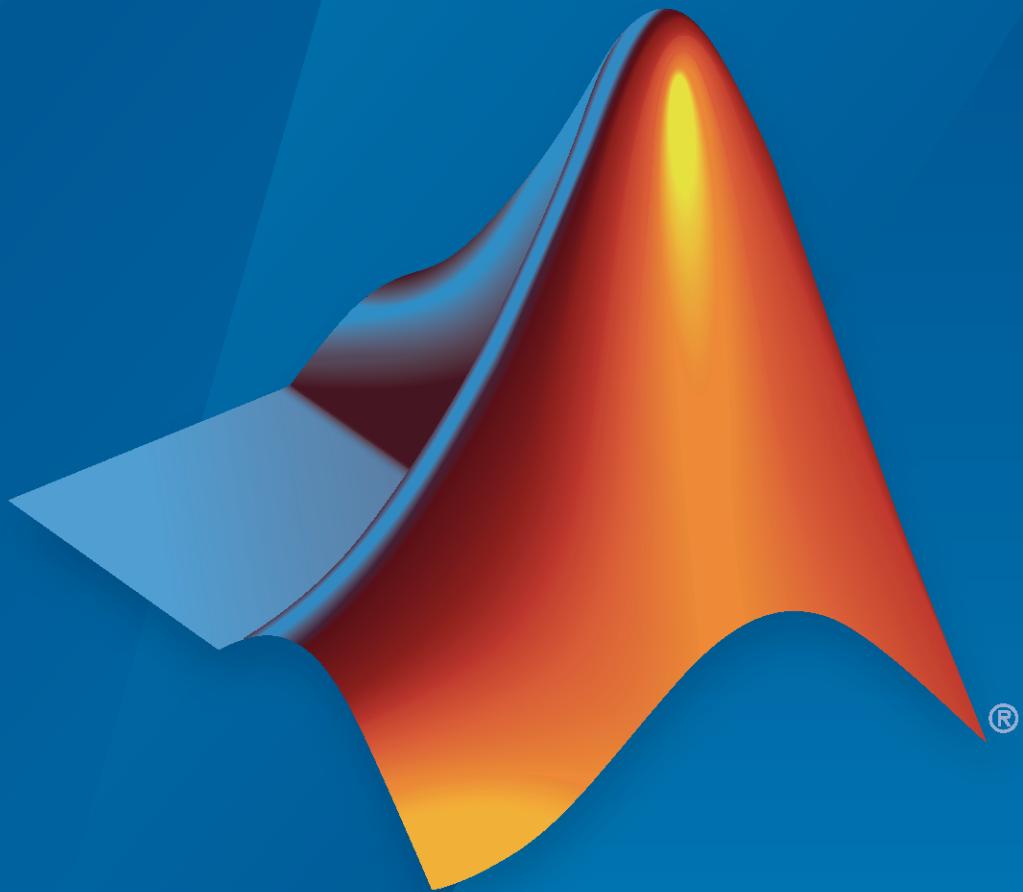


# Deep Learning Toolbox™

## 用户指南

*Mark Hudson Beale  
Martin T. Hagan  
Howard B. Demuth*



# MATLAB®

R2021a

 MathWorks®

# 如何联系 MathWorks



最新动态: [www.mathworks.com](http://www.mathworks.com)  
销售和服务: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
用户社区: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
技术支持: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



电话: 010-59827000



迈斯沃克软件(北京)有限公司  
北京市朝阳区望京东园四区6号楼  
北望金辉大厦16层1604

Deep Learning Toolbox™ 用户指南

© COPYRIGHT 1992–2021 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

**FEDERAL ACQUISITION:** This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

## 商标

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## 专利

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## 修订历史记录

1992 年 6 月	第一次印刷	
1993 年 4 月	第二次印刷	
1997 年 1 月	第三次印刷	
1997 年 7 月	第四次印刷	
1998 年 1 月	第五次印刷	版本 3 (版本 11) 中的修订内容
2000 年 9 月	第六次印刷	版本 4 (版本 12) 中的修订内容
2001 年 6 月	第七次印刷	少量修订内容 (版本 12.1)
2002 年 7 月	仅限在线版本	少量修订内容 (版本 13)
2003 年 1 月	仅限在线版本	少量修订内容 (版本 13SP1)
2004 年 6 月	仅限在线版本	版本 4.0.3 (版本 14) 中的修订内容
2004 年 10 月	仅限在线版本	版本 4.0.4 (版本 14SP1) 中的修订内容
2004 年 10 月	第八次印刷	版本 4.0.4 中的修订内容
2005 年 3 月	仅限在线版本	版本 4.0.5 (版本 14SP2) 中的修订内容
2006 年 3 月	仅限在线版本	版本 5.0 (版本 2006a) 中的修订内容
2006 年 9 月	第九次印刷	少量修订内容 (版本 2006b)
2007 年 3 月	仅限在线版本	少量修订内容 (版本 2007a)
2007 年 9 月	仅限在线版本	版本 5.1 (版本 2007b) 中的修订内容
2008 年 3 月	仅限在线版本	版本 6.0 (版本 2008a) 中的修订内容
2008 年 10 月	仅限在线版本	版本 6.0.1 (版本 2008b) 中的修订内容
2009 年 3 月	仅限在线版本	版本 6.0.2 (版本 2009a) 中的修订内容
2009 年 9 月	仅限在线版本	版本 6.0.3 (版本 2009b) 中的修订内容
2010 年 3 月	仅限在线版本	版本 6.0.4 (版本 2010a) 中的修订内容
2010 年 9 月	仅限在线版本	版本 7.0 (版本 2010b) 中的修订内容
2011 年 4 月	仅限在线版本	版本 7.0.1 (版本 2011a) 中的修订内容
2011 年 9 月	仅限在线版本	版本 7.0.2 (版本 2011b) 中的修订内容
2012 年 3 月	仅限在线版本	版本 7.0.3 (版本 2012a) 中的修订内容
2012 年 9 月	仅限在线版本	版本 8.0 (版本 2012b) 中的修订内容
2013 年 3 月	仅限在线版本	版本 8.0.1 (版本 2013a) 中的修订内容
2013 年 9 月	仅限在线版本	版本 8.1 (版本 2013b) 中的修订内容
2014 年 3 月	仅限在线版本	版本 8.2 (版本 2014a) 中的修订内容
2014 年 10 月	仅限在线版本	版本 8.2.1 (版本 2014b) 中的修订内容
2015 年 3 月	仅限在线版本	版本 8.3 (版本 2015a) 中的修订内容
2015 年 9 月	仅限在线版本	版本 8.4 (版本 2015b) 中的修订内容
2016 年 3 月	仅限在线版本	版本 9.0 (版本 2016a) 中的修订内容
2016 年 9 月	仅限在线版本	版本 9.1 (版本 2016b) 中的修订内容
2017 年 3 月	仅限在线版本	版本 10.0 (版本 2017a) 中的修订内容
2017 年 9 月	仅限在线版本	版本 11.0 (版本 2017b) 中的修订内容
2018 年 3 月	仅限在线版本	版本 11.1 (版本 2018a) 中的修订内容
2018 年 9 月	仅限在线版本	版本 12.0 (版本 2018b) 中的修订内容
2019 年 3 月	仅限在线版本	版本 12.1 (版本 2019a) 中的修订内容
2019 年 9 月	仅限在线版本	版本 13 (版本 2019b) 中的修订内容
2020 年 3 月	仅限在线版本	版本 14 (版本 2020a) 中的修订内容
2020 年 9 月	仅限在线版本	14.1 版 (版本 2020b) 中的修订内容
2021 年 3 月	仅限在线版本	版本 14.2 (版本 2021a) 中的修订内容



## 深度网络

1

<b>在 MATLAB 中进行深度学习</b>	1-2
什么是深度学习?	1-2
使用 10 行 MATLAB 代码尝试深度学习	1-4
使用迁移学习更快地开始深度学习	1-5
使用从预训练网络中提取的特征训练分类器	1-6
在并行运行的 CPU、GPU 上和云上使用大数据进行深度学习	1-6
<b>预训练的深度神经网络</b>	1-8
比较预训练网络	1-8
加载预训练网络	1-9
可视化预训练网络	1-10
特征提取	1-12
迁移学习	1-12
导入和导出网络	1-13
音频应用的预训练网络	1-14
<b>了解卷积神经网络</b>	1-16
<b>深度学习层列表</b>	1-18
深度学习层	1-18
<b>指定卷积神经网络的层</b>	1-25
图像输入层	1-25
卷积层	1-26
批量归一化层	1-30
ReLU 层	1-30
跨通道归一化（局部响应归一化）层	1-30
最大池化层和平均池化层	1-31
丢弃层	1-31
全连接层	1-31
输出层	1-32
<b>设置参数并训练卷积神经网络</b>	1-35
指定求解器和最大轮数	1-35
指定和修改学习率	1-35
指定验证数据	1-36
选择硬件资源	1-36
保存检查点网络并继续训练	1-36
在卷积层和全连接层中设置参数	1-37
训练网络	1-37
<b>长短期记忆网络</b>	1-38
LSTM 网络架构	1-38
网络层	1-41

分类、预测和预报 . . . . .	1-41
序列填充、截断和拆分 . . . . .	1-42
归一化序列数据 . . . . .	1-45
无法放入内存的数据 . . . . .	1-45
可视化 . . . . .	1-45
LSTM 层架构 . . . . .	1-45

## 深度网络设计器

2

使用深度网络设计器进行迁移学习 . . . . .	2-2
<b>使用深度网络设计器构建网络 . . . . .</b>	<b>2-15</b>
打开 App 和导入网络 . . . . .	2-15
创建和编辑网络 . . . . .	2-16
检查网络 . . . . .	2-18
训练网络 . . . . .	2-19
导出网络 . . . . .	2-19
<b>使用深度网络设计器创建简单的序列分类网络 . . . . .</b>	<b>2-21</b>

## 图像深度学习

3

<b>使用深度学习对网络摄像头图像进行分类 . . . . .</b>	<b>3-2</b>
<b>训练深度学习网络以对新图像进行分类 . . . . .</b>	<b>3-6</b>
<b>训练残差网络进行图像分类 . . . . .</b>	<b>3-13</b>
<b>使用 GoogLeNet 对图像进行分类 . . . . .</b>	<b>3-22</b>
<b>使用预训练网络提取图像特征 . . . . .</b>	<b>3-27</b>
<b>使用 AlexNet 进行迁移学习 . . . . .</b>	<b>3-32</b>
<b>创建简单的深度学习网络以用于分类 . . . . .</b>	<b>3-39</b>
<b>针对回归训练卷积神经网络 . . . . .</b>	<b>3-44</b>
<b>将分类网络转换为回归网络 . . . . .</b>	<b>3-52</b>
<b>训练生成对抗网络 (GAN) . . . . .</b>	<b>3-57</b>
<b>训练变分自编码器 (VAE) 以生成图像 . . . . .</b>	<b>3-68</b>

**4**

使用深度学习进行序列分类 . . . . .	4-2
使用深度学习进行时序预测 . . . . .	4-9
使用深度学习进行语音命令识别 . . . . .	4-16
使用深度学习进行“序列到序列”分类 . . . . .	4-33
使用深度学习进行“序列到序列”回归 . . . . .	4-38
使用深度学习对视频进行分类 . . . . .	4-45
使用深度学习对文本数据进行分类 . . . . .	4-55
使用卷积神经网络对文本数据进行分类 . . . . .	4-63
使用深度学习生成文本 . . . . .	4-72
《傲慢与偏见》与 MATLAB . . . . .	4-78
使用深度学习进行逐单词文本生成 . . . . .	4-83

**深度学习调整和可视化****5**

使用 GoogLeNet 的 Deep Dream 图像 . . . . .	5-2
使用贝叶斯优化进行深度学习 . . . . .	5-8
并行训练深度学习网络 . . . . .	5-17
监控深度学习训练进度 . . . . .	5-22
自定义深度学习网络训练期间的输出 . . . . .	5-26
使用类激活映射调查网络预测 . . . . .	5-30
可视化卷积神经网络的激活区域 . . . . .	5-35
可视化卷积神经网络的特征 . . . . .	5-46

**6****通过并行计算和云进行深度学习****7**

使用自动并行支持功能在云中训练网络 . . . . .	7-2
使用 <code>parfeval</code> 训练多个深度学习网络 . . . . .	7-6
将深度学习批处理作业发送到群集 . . . . .	7-13
使用自动多 GPU 支持训练网络 . . . . .	7-16
使用 <code>parfor</code> 训练多个深度学习网络 . . . . .	7-20
将深度学习数据上传到云 . . . . .	7-27

**计算机视觉示例****8**

使用 YOLO v2 深度学习进行目标检测 . . . . .	8-2
使用深度学习进行语义分割 . . . . .	8-11
使用扩张卷积进行语义分割 . . . . .	8-26
使用深度学习对多光谱图像进行语义分割 . . . . .	8-31
使用深度学习进行三维脑肿瘤分割 . . . . .	8-48
定义使用 Tversky 损失的自定义像素分类层 . . . . .	8-59
使用 R-CNN 深度学习训练目标检测器 . . . . .	8-66
使用 Faster R-CNN 深度学习进行目标检测 . . . . .	8-78

**图像处理示例****9**

使用预训练的神经网络去除彩色图像中的噪声 . . . . .	9-2
使用深度学习执行单图像超分辨率 . . . . .	9-8

使用深度学习进行 JPEG 图像去块 . . . . .	9-21
使用深度学习进行图像处理算子逼近 . . . . .	9-33
使用深度学习的神经样式迁移 . . . . .	9-47

## 自动驾驶示例

10

训练深度学习车辆检测器 . . . . .	10-2
使用单目相机和语义分割创建占据栅格 . . . . .	10-12

## 激光雷达示例

11

## 信号处理示例

12

使用深度学习进行波形分割 . . . . .	12-2
使用长短期记忆网络对 ECG 信号进行分类 . . . . .	12-22
使用小波分析和深度学习对时序分类 . . . . .	12-38

## 无线通信示例

13

使用深度学习进行调制分类 . . . . .	13-2
------------------------	------

## 音频示例

14

基于小波散射和深度学习的口述数字识别 . . . . .	14-2
基于深度学习网络的鸡尾酒会信源分离 . . . . .	14-16
使用深度学习检测噪声中的语音活动 . . . . .	14-37

使用深度学习网络对语音去噪 ..... 14-58

使用 GRU 网络进行性别分类 ..... 14-76

## 强化学习示例

15

## 预测性维护示例

16

## 导入、导出和自定义

17

定义自定义深度学习层 ..... 17-2

    层模板 ..... 17-2

    中间层架构 ..... 17-5

    检查层的有效性 ..... 17-10

    在网络中包含层 ..... 17-11

    输出层架构 ..... 17-11

指定自定义权重初始化函数 ..... 17-16

比较层权重初始化函数 ..... 17-22

基于预训练的 Keras 层组合网络 ..... 17-28

定义自定义训练循环、损失函数和网络 ..... 17-33

    为自定义训练循环定义深度学习网络 ..... 17-33

    指定损失函数 ..... 17-35

    使用自动微分更新可学习参数 ..... 17-36

## 深度学习数据预处理

18

预处理图像以进行深度学习 ..... 18-2

    使用重新缩放和裁剪调整图像大小 ..... 18-2

    用随机几何变换增强图像进行训练 ..... 18-3

    使用内置数据存储执行附加图像处理运算 ..... 18-4

    通过组合和变换来应用自定义图像处理管道 ..... 18-4

为图像到图像的回归准备数据存储 ..... 18-6

使用无法放入内存的序列数据训练网络 ..... 18-14

使用序列数据的自定义小批量数据存储来训练网络 .....	18-18
使用深度学习对无法放入内存的文本数据进行分类 .....	18-21
使用自定义小批量数据存储对无法放入内存的文本数据进行分类 .....	18-27

## 深度学习代码生成

19

深度学习网络的代码生成 .....	19-2
语义分割网络的代码生成 .....	19-10
使用 GPU Coder 优化车道检测 .....	19-14
为使用 YOLO v2 的目标检测生成代码 .....	19-24
交通标志检测和识别 .....	19-27
徽标识别网络 .....	19-35
行人检测 .....	19-39
去噪深度神经网络的代码生成 .....	19-45
训练和部署用于语义分割的全卷积网络 .....	19-49
使用 U-Net 的语义分割网络的代码生成 .....	19-60
ARM 目标上的深度学习代码生成 .....	19-67
通过 ARM 计算使用 codegen 进行深度学习预测 .....	19-71
针对不同批量大小在 Intel 目标上进行深度学习代码生成 .....	19-76

## 神经网络设计书籍

20

神经网络设计的工作流 .....	20-2
创建神经网络对象 .....	20-3

**多层浅层神经网络与反向传播训练****21**

<b>多层浅层神经网络与反向传播训练</b> .....	<b>21-2</b>
<b>多层浅层神经网络架构</b> .....	<b>21-3</b>
神经元模型 (logsig、tansig、purelin) .....	21-3
前馈神经网络 .....	21-4
<b>选择神经网络输入输出处理函数</b> .....	<b>21-6</b>
表示未知或不关心的目标 .....	21-7
<b>划分数据以实现最优神经网络训练</b> .....	<b>21-8</b>
<b>创建、配置和初始化多层浅层神经网络</b> .....	<b>21-9</b>
其他相关架构 .....	21-9
初始化权重 (init) .....	21-10
<b>训练与应用多层浅层神经网络</b> .....	<b>21-11</b>
训练算法 .....	21-11
训练示例 .....	21-12
使用网络 .....	21-14
<b>分析训练后的浅层神经网络性能</b> .....	<b>21-15</b>
改进结果 .....	21-18

**动态神经网络****22**

<b>设计时序 NARX 反馈神经网络</b> .....	<b>22-2</b>
多个外部变量 .....	22-6
<b>多步神经网络预测</b> .....	<b>22-8</b>
在开环模式下设置 .....	22-8
基于初始条件进行多步闭环预测 .....	22-8
遵循已知序列的多步闭环预测 .....	22-9
在闭环仿真后进行开环仿真 .....	22-10

**控制系统****23**

<b>在 Simulink 中设计神经网络预测控制器</b> .....	<b>23-2</b>
系统识别 .....	23-2
预测控制 .....	23-3

**径向基神经网络****24**

<b>径向基神经网络</b> .....	<b>24-2</b>
神经元模型 .....	24-2
网络架构 .....	24-3
精确设计 (newrbe) .....	24-3
更高效的设计 (newrb) .....	24-4
示例 .....	24-5

**自组织和学习向量量化网络****25**

<b>自组织映射神经网络的聚类</b> .....	<b>25-2</b>
拓扑 (gridtop、hextop、randtop) .....	25-3
距离函数 (dist、linkdist、mandist、boxdist) .....	25-6
架构 .....	25-8
创建自组织映射神经网络 (selforgmap) .....	25-8
训练 (learnsomb) .....	25-10
示例 .....	25-11

**自适应滤波器和自适应训练****26****高级主题****27**

<b>使用并行和 GPU 计算的浅层神经网络</b> .....	<b>27-2</b>
并行机制模式 .....	27-2
分布式计算 .....	27-2
单 GPU 计算 .....	27-4
分布式 GPU 计算 .....	27-6
并行时序 .....	27-7
并行可用性、回退和反馈 .....	27-7

<b>优化神经网络训练速度和内存</b> .....	<b>27-9</b>
内存减少 .....	27-9
快速 Elliot sigmoid .....	27-9

<b>选择多层神经网络训练函数</b> .....	<b>27-13</b>
SIN 数据集 .....	27-13

PARITY 数据集	27-15
ENGINE 数据集	27-16
CANCER 数据集	27-18
CHOLESTEROL 数据集	27-19
DIABETES 数据集	27-21
总结	27-22
<b>提高浅层神经网络泛化能力，避免过拟合</b>	<b>27-24</b>
重新训练神经网络	27-25
多个神经网络	27-25
早停法	27-26
索引数据划分 (divideind)	27-27
随机数据划分 (dividerand)	27-27
分块数据划分 (divideblock)	27-27
交错数据划分 (divideint)	27-27
正则化	27-28
早停法和正则化的摘要与讨论	27-29
训练后分析 (regression)	27-30

## 历史神经网络

**28**

<b>感知器神经网络</b>	<b>28-2</b>
神经元模型	28-2
感知器架构	28-3
创建感知器	28-4
感知器学习规则 (learnp)	28-5
训练 (train)	28-7
限制和注意事项	28-10

## 神经网络对象引用

**29**

<b>神经网络对象属性</b>	<b>29-2</b>
常规	29-2
架构	29-2
子对象结构体	29-4
函数	29-6
权重和偏置值	29-8

## 函数逼近、聚类和控制示例

**30**

<b>体脂估计</b>	<b>30-2</b>
<b>螃蟹分类</b>	<b>30-8</b>

<b>葡萄酒分类</b>	30-16
<b>癌症检测</b>	30-22
<b>字符识别</b>	30-28
<b>训练堆叠自编码器进行图像分类</b>	30-32
<b>鸢尾花聚类</b>	30-41
<b>基因表达分析</b>	30-49
<b>磁悬浮建模</b>	30-57
<b>竞争学习</b>	30-66
<b>一维自组织映射</b>	30-69
<b>二维自组织映射</b>	30-71
<b>径向基逼近</b>	30-74
<b>径向基神经元欠叠</b>	30-78
<b>径向基神经元过叠</b>	30-80
<b>GRNN 函数逼近</b>	30-82
<b>PNN 分类</b>	30-86
<b>学习向量量化</b>	30-90
<b>线性预测设计</b>	30-93
<b>自适应线性预测</b>	30-97
<b>用双输入感知器分类</b>	30-101
<b>离群值输入向量</b>	30-106
<b>归一化感知器规则</b>	30-112
<b>线性不可分向量</b>	30-118

**数学表示法****A****用于 Simulink 环境的神经网络模块****B**

<b>神经网络 Simulink 模块库</b> .....	<b>B-2</b>
传递函数模块 .....	B-2
净输入模块 .....	B-3
权重模块 .....	B-3
处理模块 .....	B-3

**代码说明****C**

# 深度网络

---

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “预训练的深度神经网络” (第 1-8 页)
- “了解卷积神经网络” (第 1-16 页)
- “深度学习层列表” (第 1-18 页)
- “指定卷积神经网络的层” (第 1-25 页)
- “设置参数并训练卷积神经网络” (第 1-35 页)
- “长短期记忆网络” (第 1-38 页)

# 在 MATLAB 中进行深度学习

## 本节内容

- “什么是深度学习？”（第 1-2 页）
- “使用 10 行 MATLAB 代码尝试深度学习”（第 1-4 页）
- “使用迁移学习更快地开始深度学习”（第 1-5 页）
- “使用从预训练网络中提取的特征训练分类器”（第 1-6 页）
- “在并行运行的 CPU、GPU 上和云上使用大数据进行深度学习”（第 1-6 页）

## 什么是深度学习？

深度学习是机器学习的一个分支，它训练计算机像人类一样自然地做事情，即：从经验中学习。机器学习算法使用计算方法直接从数据“学习”信息，而不依赖预先确定的方程作为模型。深度学习尤其适用于图像识别，这对于解决人脸识别、运动检测和许多高级驾驶员辅助技术（如自动驾驶、车道检测、行人检测和自动泊车）等问题非常重要。

Deep Learning Toolbox 提供简单的 MATLAB 命令来创建和互连深度神经网络的各个层。示例和预训练网络使得使用 MATLAB 进行深度学习变得很容易，即使没有高级计算机视觉算法或神经网络方面的知识也不会感觉困难。

有关实际深度学习方法的免费实践课程，请参阅深度学习入门之旅。

您想要执行什么操作？	了解更多信息
执行迁移学习以用您的数据微调网络	<p>“使用迁移学习更快地开始深度学习”（第 1-5 页）</p> <p><b>提示</b> 微调预训练网络来学习一项新任务通常比训练新网络要快得多，也容易得多。</p>
使用预训练网络对图像进行分类	“预训练的深度神经网络”（第 1-8 页）
新建一个用于分类或回归的深度神经网络	<p>“创建简单的深度学习网络以用于分类”（第 3-39 页）</p> <p>“针对回归训练卷积神经网络”（第 3-44 页）</p>
调整图像大小、旋转图像或预处理图像以进行训练或预测	“预处理图像以进行深度学习”（第 18-2 页）
根据文件夹名称自动标注图像数据，或使用 App 以交互方式标注图像数据	<p>“Train Network for Image Classification”</p> <p>Image Labeler (Computer Vision Toolbox)</p>
创建深度学习网络以处理序列和时序数据。	<p>“使用深度学习进行序列分类”（第 4-2 页）</p> <p>“使用深度学习进行时序预测”（第 4-9 页）</p>
对图像的每个像素进行分类（例如，道路、汽车、行人）	“Getting Started with Semantic Segmentation Using Deep Learning” (Computer Vision Toolbox)
检测和识别图像中的目标	“Recognition, Object Detection, and Semantic Segmentation” (Computer Vision Toolbox)

您想要执行什么操作?	了解更多信息
对文本数据进行分类	"使用深度学习对文本数据进行分类" (第 4-55 页)
对音频数据进行分类以实现语音识别	"使用深度学习进行语音命令识别" (第 4-16 页)
可视化网络学习到的特征	"使用 GoogLeNet 的 Deep Dream 图像" (第 5-2 页) "可视化卷积神经网络的激活区域" (第 5-35 页)
在桌面环境的 CPU、GPU 或并行运行的多个 GPU 上或者云中的群集上进行训练，并处理因太大而无法放入内存的数据集	

要了解有关深度学习应用领域（包括自动驾驶）的更多信息，请参阅“深度学习应用”。

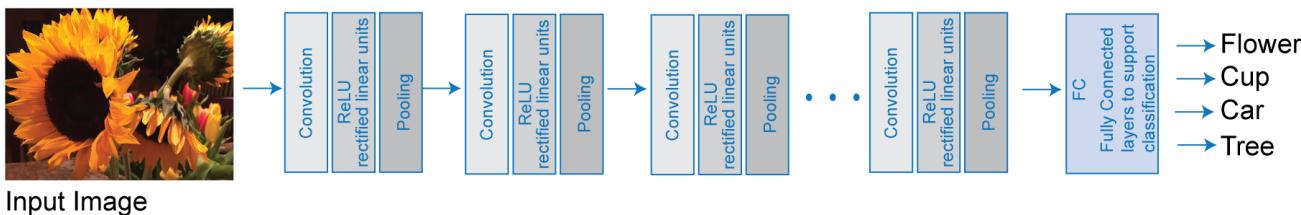
要选择是使用预训练网络还是创建新的深度网络，请参考下表中的情形。

	使用预训练网络进行迁移学习	创建新的深度网络
训练数据	数百到数千张已标注图像（小）	数干到数百万张已标注图像
计算量	中等计算量（可选择使用 GPU）	计算量大（需要使用 GPU 来提高速度）
训练时间	数秒到数分钟	解决真实问题需要几天到几周的时间
模型准确度	好，取决于预训练模型	高，但对于较小的数据集，可能会出现过拟合的情况

有关详细信息，请参阅“Choose Network Architecture”。

深度学习使用神经网络直接从数据中学习有用的特征表示。神经网络结合了多个非线性处理层，它使用并行运行的简单元素并受到生物神经系统的启发。深度学习模型在对象分类方面可以达到非常高的准确度，有时甚至超过人类的水平。

您在训练模型时会使用大量标注数据和神经网络架构，这些架构包含许多层，通常包括一些卷积层。训练这些模型需要进行大量计算，通常可以使用高性能 GPU 来加快训练速度。下图显示卷积神经网络如何将自动从许多图像中学习特征的各层组合起来，以对新图像进行分类。



许多深度学习应用程序都会使用图像文件，有时甚至会用到数百万个图像文件。为了高效地访问大量图像文件进行深度学习，MATLAB 提供了 `imageDatastore` 函数。使用此函数可以：

- 自动读取成批图像，以便在机器学习和计算机视觉应用程序中加快处理速度

- 从图像集合中导入太大而无法放入内存的数据
- 根据文件夹名称自动标注图像数据

## 使用 10 行 MATLAB 代码尝试深度学习

此示例说明如何仅通过 10 行 MATLAB 代码使用深度学习来识别实时网络摄像头画面中的对象。尝试此示例以了解在 MATLAB 中使用深度学习有多么简单。

- 1 运行以下命令，根据需要获取下载，连接到网络摄像头，并获得预训练的神经网络。

```
camera = webcam; % Connect to the camera  
net = alexnet; % Load the neural network
```

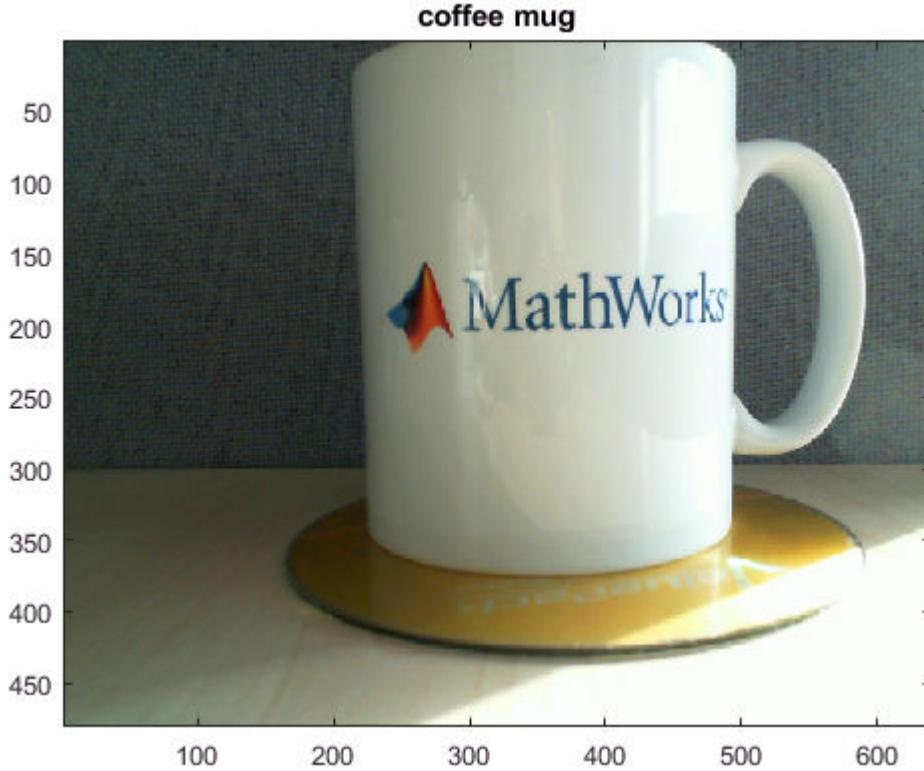
如果您需要安装 `webcam` 和 `alexnet` 附加功能，对于每个函数都会显示一条带链接的消息，帮助您使用附加功能资源管理器下载免费的附加功能。或者，请参阅 Deep Learning Toolbox Model for AlexNet Network 和 MATLAB Support Package for USB Webcams。

安装 Deep Learning Toolbox Model for AlexNet Network 后，可以使用它对图像进行分类。AlexNet 是预训练的卷积神经网络 (CNN)，已基于超过一百万个图像进行训练，可以将图像分为 1000 个对象类别（例如键盘、鼠标、咖啡杯、铅笔和多种动物）。

- 2 运行以下代码来显示和分类实时图像。将网络摄像头对准一个对象，神经网络会报告它认为网络摄像头正在显示哪类对象。网络会持续进行图像分类，直到您按下 **Ctrl+C** 为止。该代码使用 `imresize` 针对网络调整图像的大小。

```
while true  
    im = snapshot(camera); % Take a picture  
    image(im); % Show the picture  
    im = imresize(im,[227 227]); % Resize the picture for alexnet  
    label = classify(net,im); % Classify the picture  
    title(char(label)); % Show the class label  
    drawnow  
end
```

在此示例中，网络对咖啡杯进行了正确分类。用您周围的物体进行试验，看看该网络的准确度如何。



要观看此示例的视频，请参阅使用 11 行 MATLAB 代码进行深度学习。

要了解如何扩展此示例并显示类的概率分数，请参阅“使用深度学习对网络摄像头图像进行分类”（第 3-2 页）。

在深度学习的后续步骤中，您可以将预训练网络用于其他任务。通过迁移学习或特征提取解决新的图像数据分类问题。有关示例，请参阅“使用迁移学习更快地开始深度学习”（第 1-5 页）和“使用从预训练网络中提取的特征训练分类器”（第 1-6 页）。要尝试其他预训练网络，请参阅“预训练的深度神经网络”（第 1-8 页）。

## 使用迁移学习更快地开始深度学习

深度学习应用中常常用到迁移学习。您可以采用预训练的网络，基于它学习新任务。用迁移学习来微调网络比从头开始训练要快得多，也容易得多。您可以使用较少数量的训练图像快速让网络学习新任务。迁移学习的优势在于，预训练网络已学习一系列丰富的特征，这些特征可以应用于其他各种类似任务。

例如，如果您利用一个基于数千或数百万个图像进行过训练的网络，可以只使用数百个图像重新训练它来进行新目标检测。您可以用比原始训练数据小得多的数据集有效地微调预训练网络。如果您的数据集非常大，则迁移学习可能不会比训练新网络更快。

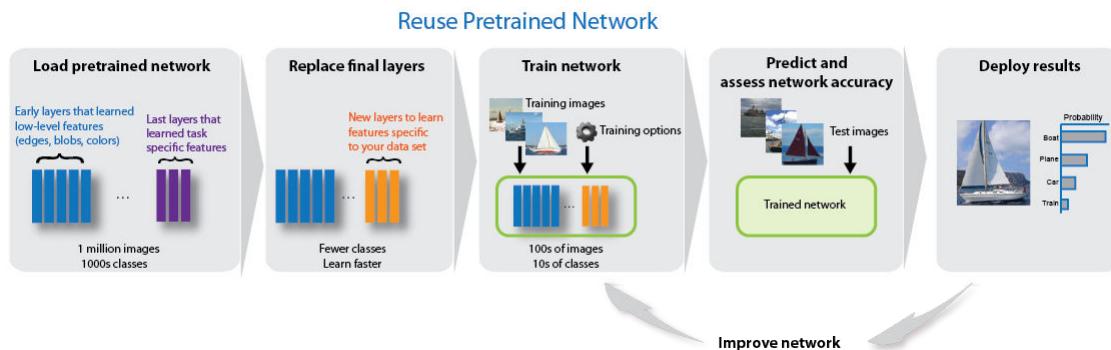
迁移学习使您能够：

- 将学习到的预训练网络的特征迁移到新问题上
- 迁移学习比训练新网络更快更容易
- 减少训练时间和数据集大小

- 无需了解如何创建全新网络，即可进行深度学习

有关交互方式的示例，请参阅“使用深度网络设计器进行迁移学习”（第 2-2 页）。

有关编程方式的示例，请参阅“训练深度学习网络以对新图像进行分类”（第 3-6 页）。



## 使用从预训练网络中提取的特征训练分类器

特征提取让您利用预训练网络的强大功能，而无需在训练方面投入时间和精力。特征提取可能是使用深度学习的最快方法。您从预训练网络中提取学习到的特征，并使用这些特征来训练分类器，例如，支持向量机（SVM - 需要 Statistics and Machine Learning Toolbox™）。例如，如果使用 `alexnet` 训练过的 SVM 能够基于您的训练集和验证集达到 90% 以上的准确度，则可能不值得为再提高一点准确度而通过迁移学习进行微调。如果您对较小的数据集进行微调，可能还存在过拟合的风险。如果 SVM 无法达到对您的应用来说足够好的准确度，您可以通过微调来提高准确度。

有关示例，请参阅“使用预训练网络提取图像特征”（第 3-27 页）。

## 在并行运行的 CPU、GPU 上和云上使用大数据进行深度学习

神经网络本质上是并行算法。您可以通过 Parallel Computing Toolbox™ 来利用这种并行机制，在多核 CPU、图形处理单元 (GPU) 以及具有多个 CPU 和 GPU 的计算机群集之间分配训练。

训练深度网络的计算量非常巨大，使用高性能 GPU 通常可以加快训练速度。如果您没有合适的 GPU，您可以在一个或多个 CPU 内核上进行训练。您可以在单个 GPU 或 CPU 上或者在多个 GPU 或 CPU 内核上或在群集上以并行方式训练卷积神经网络。使用 GPU 或并行选项需要 Parallel Computing Toolbox。

您不需要使用多台计算机来解决数据集太大而无法放入内存的问题。您可以使用 `imageDatastore` 函数处理批量数据，而无需使用计算机群集。但是，如果您有可用的群集，将您的代码放入数据存储库会很有帮助，这样可以避免到处移动大量数据。

要了解有关深度学习硬件和内存设置的详细信息，请参阅。

## 另请参阅

### 相关示例

- “使用深度学习对网络摄像头图像进行分类”（第 3-2 页）
- “使用深度网络设计器进行迁移学习”（第 2-2 页）
- “训练深度学习网络以对新图像进行分类”（第 3-6 页）

- “预训练的深度神经网络” (第 1-8 页)
- “创建简单的深度学习网络以用于分类” (第 3-39 页)
- 
- “Recognition, Object Detection, and Semantic Segmentation” (Computer Vision Toolbox)
- “使用深度学习对文本数据进行分类” (第 4-55 页)
- “Deep Learning Tips and Tricks”

## 预训练的深度神经网络

本节内容
“比较预训练网络”（第 1-8 页）
“加载预训练网络”（第 1-9 页）
“可视化预训练网络”（第 1-10 页）
“特征提取”（第 1-12 页）
“迁移学习”（第 1-12 页）
“导入和导出网络”（第 1-13 页）
“音频应用的预训练网络”（第 1-14 页）

您可以采用预训练的图像分类网络，它已学会从自然图像中提取功能强大且包含丰富信息的特征，并以此作为学习新任务的起点。大多数预训练网络是基于 ImageNet 数据库 [1] 的子集进行训练的，该数据库用于 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [2] 中。这些网络已经对超过一百万个图像进行了训练，可以将图像分为 1000 个对象类别，例如键盘、咖啡杯、铅笔和多种动物。通常来说，使用预训练网络进行迁移学习比从头开始训练网络更快更容易。

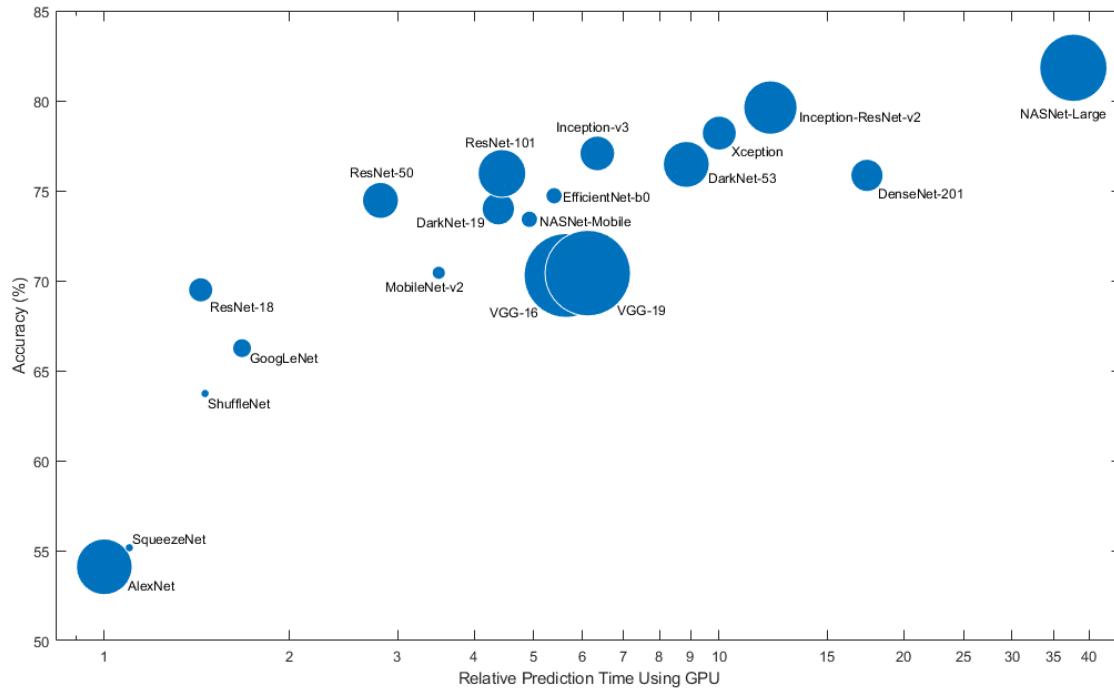
您可以将之前训练过的网络用于以下任务：

目的	说明
分类	将预训练网络直接应用于分类问题。要对新图像进行分类，请使用 <code>classify</code> 。有关如何使用预训练网络进行分类的示例，请参阅“使用 GoogLeNet 对图像进行分类”（第 3-22 页）。
特征提取	通过使用层激活作为特征，使用预训练网络作为特征提取器。您可以使用这些激活作为特征来训练另一个机器学习模型，例如支持向量机 (SVM)。有关详细信息，请参阅“特征提取”（第 1-12 页）。有关示例，请参阅“使用预训练网络提取图像特征”（第 3-27 页）。
迁移学习	从基于大型数据集训练的网络中提取层，并基于新数据集进行微调。有关详细信息，请参阅“迁移学习”（第 1-12 页）。有关简单的示例，请参阅“迁移学习快速入门”。要尝试更多预训练网络，请参阅“训练深度学习网络以对新图像进行分类”（第 3-6 页）。

## 比较预训练网络

当选择适用于您的问题的网络时，预训练网络具有不同的重要特征。最重要的特征是网络的准确度、速度和规模。选择网络时通常需要在这些特征之间进行权衡。使用下图比较 ImageNet 验证准确度和使用网络进行预测所需的时间。

**提示** 要开始迁移学习，请尝试选择一个更快的网络，例如 SqueezeNet 或 GoogLeNet。然后，您可以快速迭代并尝试不同设置，如数据预处理步骤和训练选项。一旦您感觉到哪些设置运行良好，请尝试更准确的网络，例如 Inception-v3 或 ResNet，看看这是否能改进您的结果。



**注意** 上图仅显示不同网络的相对速度。准确的预测和训练迭代时间取决于您使用的硬件和小批量大小。

理想的网络具有高准确度并且速度很快。该图显示的是使用现代 GPU (NVIDIA® Tesla® P100) 和大小为 128 的小批量时分类准确度对预测时间的结果。预测时间是相对于最快的网络来测量的。每个标记的面积与网络在磁盘上的大小成正比。

ImageNet 验证集上的分类准确度是衡量在 ImageNet 上训练的网络准确度的最常见方法。如果您的网络在 ImageNet 上准确，则当您使用迁移学习或特征提取将网络应用于其他自然图像数据集时，您的网络通常也是准确的。这种泛化之所以可行，是因为网络已学会从自然图像中提取强大的信息特征，这些特征可以泛化到其他类似的数据集。但是，在 ImageNet 上的高准确度并不能始终直接迁移到其他任务，因此最好尝试多个网络。

如果您要使用受限制的硬件执行预测或通过 Internet 分发网络，则还要考虑网络在磁盘上和内存中的大小。

## 网络准确度

可以使用多种方法来计算基于 ImageNet 验证集的分类准确度，不同数据源使用不同的方法。有时使用包含多个模型的集合，有时使用多次裁剪对每个图像进行多次计算。有时会引用 top-5 准确度，而不是标准 (top-1) 准确度。由于这些差异，通常无法直接比较不同数据源的准确度。Deep Learning Toolbox 中预训练网络的准确度是使用单一模型和单一中心图像裁剪的标准 (top-1) 准确度。

## 加载预训练网络

要加载 SqueezeNet 网络，请在命令行中键入 `squeezezenet`。

```
net = squeezeNet;
```

对于其他网络，请使用 `googlenet` 等函数来获取链接，以便从附加功能资源管理器下载预训练网络。

下表列出了基于 ImageNet 训练的可用预训练网络以及这些网络的一些属性。网络深度定义为从输入层到输出层的路径中顺序卷积层或全连接层的最大数量。所有网络的输入均为 RGB 图像。

网络	深度	大小	参数（单位为百万）	图像输入大小
<b>squeezeNet</b>	18	5.2 MB	1.24	227×227
<b>googlenet</b>	22	27 MB	7.0	224×224
<b>inceptionv3</b>	48	89 MB	23.9	299×299
<b>densenet201</b>	201	77 MB	20.0	224×224
<b>mobilenetv2</b>	53	13 MB	3.5	224×224
<b>resnet18</b>	18	44 MB	11.7	224×224
<b>resnet50</b>	50	96 MB	25.6	224×224
<b>resnet101</b>	101	167 MB	44.6	224×224
<b>xception</b>	71	85 MB	22.9	299×299
<b>inceptionresnetv2</b>	164	209 MB	55.9	299×299
<b>shufflenet</b>	50	5.4 MB	1.4	224×224
<b>nasnetmobile</b>	*	20 MB	5.3	224×224
<b>nasnetlarge</b>	*	332 MB	88.9	331×331
<b>darknet19</b>	19	78 MB	20.8	256×256
<b>darknet53</b>	53	155 MB	41.6	256×256
<b>efficientnetb0</b>	82	20 MB	5.3	224×224
<b>alexnet</b>	8	227 MB	61.0	227×227
<b>vgg16</b>	16	515 MB	138	224×224
<b>vgg19</b>	19	535 MB	144	224×224

\*NASNet-Mobile 和 NASNet-Large 网络不是由模块的线性序列构成的。

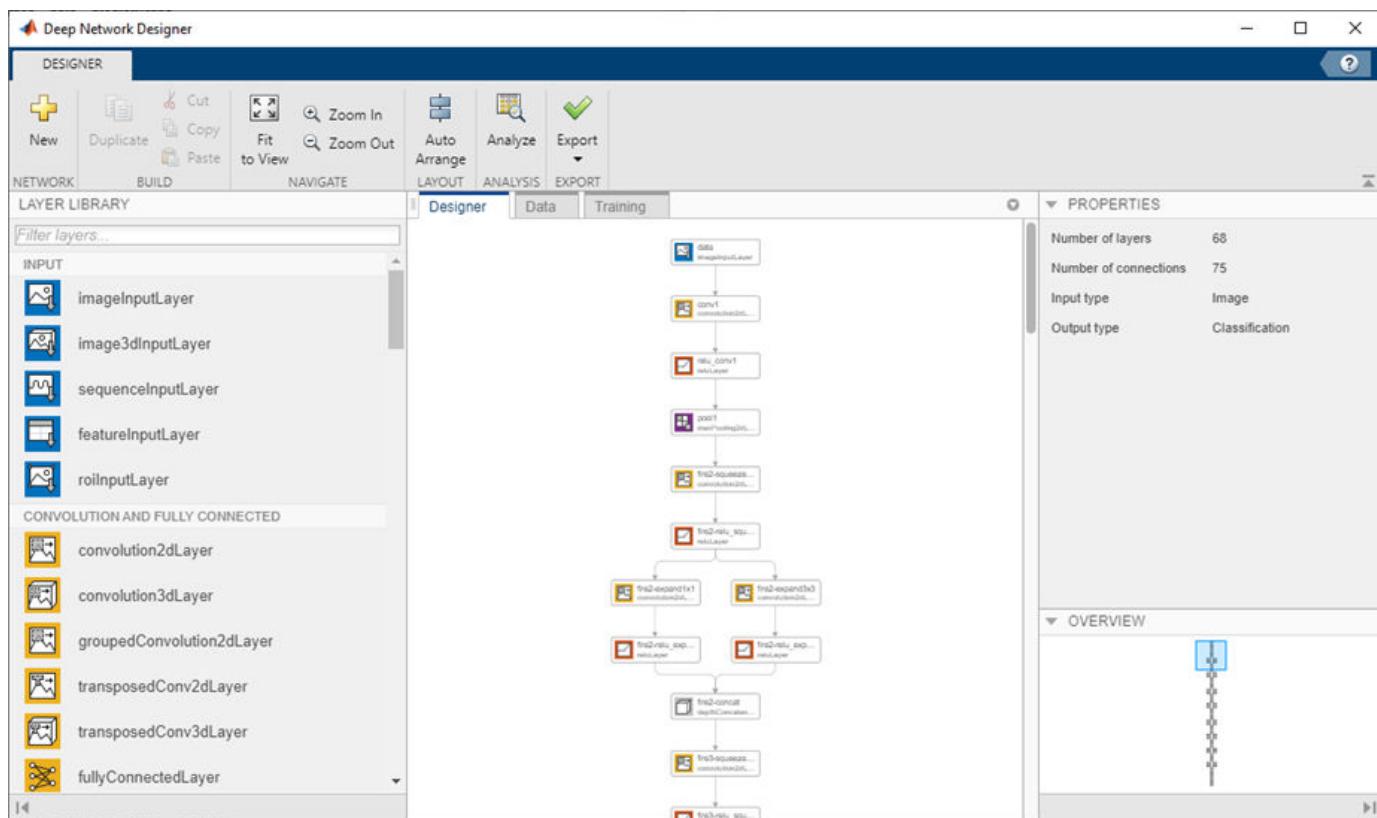
### 基于 Places365 训练的 GoogLeNet

标准 GoogLeNet 网络是基于 ImageNet 数据集进行训练的，但您也可以加载基于 Places365 数据集训练的网络 [3] [4]。基于 Places365 训练的网络将图像分为 365 个不同位置类别，例如田野、公园、跑道和大厅。要加载基于 Places365 数据集训练的预训练 GoogLeNet 网络，请使用 `googlenet('Weights','places365')`。在执行迁移学习以执行新任务时，最常见的方法是使用基于 ImageNet 预训练的网络。如果新任务类似于场景分类，则使用基于 Places365 训练的网络可以提供更高的准确度。

## 可视化预训练网络

您可以使用深度网络设计器加载和可视化预训练网络。

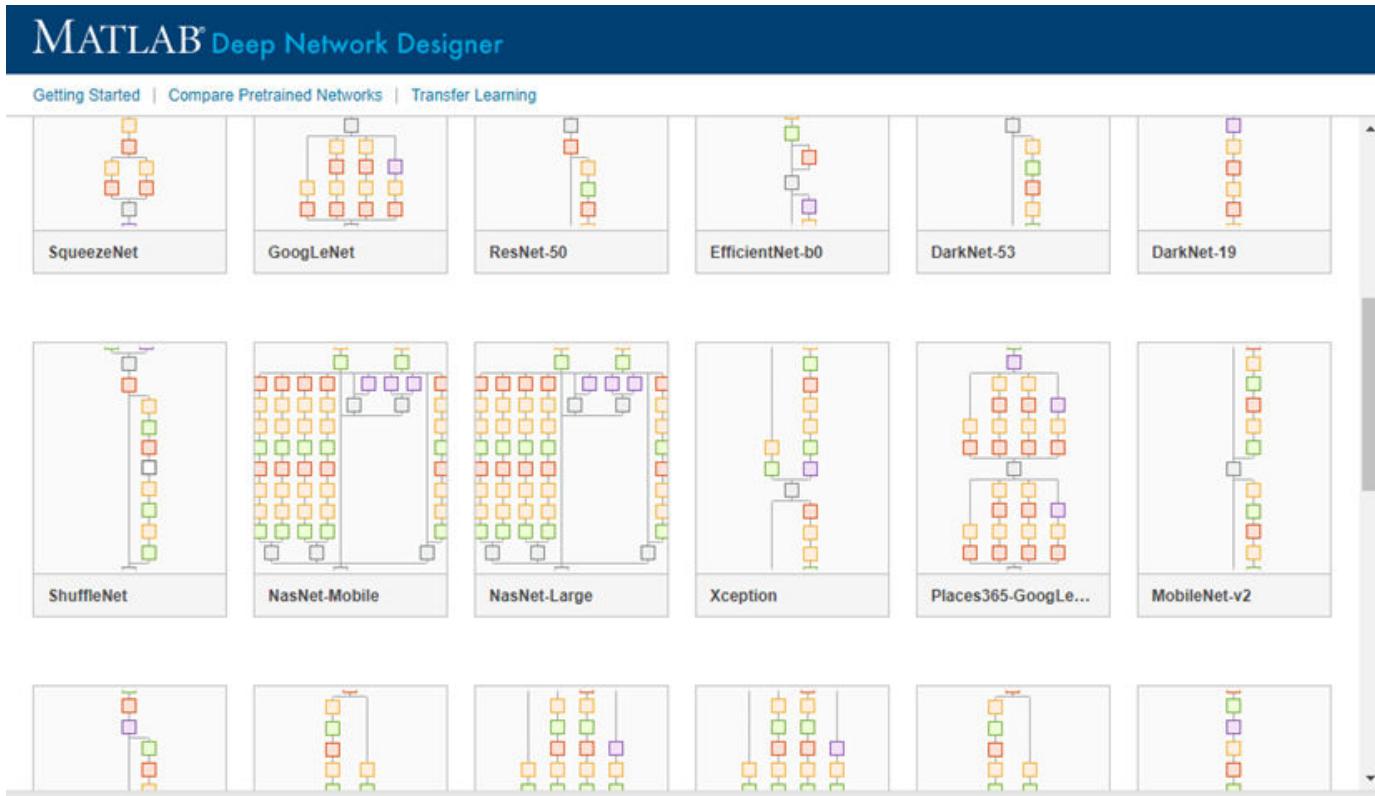
```
deepNetworkDesigner(squeezeNet)
```



要查看和编辑层属性，请选择一个层。有关层属性的信息，请点击层名称旁边的帮助图标。

This screenshot shows a detailed view of the Deep Network Designer's Properties panel. A specific layer, 'conv2-norm2 crossChannelNormaliz...', is selected and highlighted with a blue border. The Properties panel on the right shows its configuration with fields for Name (set to 'conv2-norm2'), WindowChannelSize (5), Alpha (0.0001), Beta (0.75), and K (1). A help icon is visible next to the Name field. The left side of the interface shows a portion of the neural network diagram with arrows indicating the flow of data between layers.

通过点击 **New**，在深度网络设计器中浏览其他预训练网络。



如果您需要下载网络，可点击[安装](#)，打开附加功能资源管理器。

## 特征提取

特征提取可以简单快捷地利用深度学习的强大功能，而无需投入时间和精力来训练完整网络。由于它只需遍历一次训练图像，因此如果您没有 GPU，特征提取会特别有用。您使用预训练网络提取学习到的图像特征，然后使用这些特征来训练分类器，例如使用 `fitcsvm` 的支持向量机。

当您的新数据集很小时，请尝试使用特征提取。由于您仅基于提取的特征来训练简单的分类器，因此训练速度很快。由于几乎没有数据可供学习，因此微调网络的更深层也不太可能提高准确度。

- 如果您的数据与原始数据非常相似，则在网络的更深层提取的更具体的特征可能对新任务有用。
- 如果您的数据与原始数据相差很大，则在网络的更深层提取的特征可能对您的任务用处不大。请尝试基于从较浅网络层提取的更一般特征来训练最终的分类器。如果新数据集很大，则您也可以尝试从头开始训练网络。

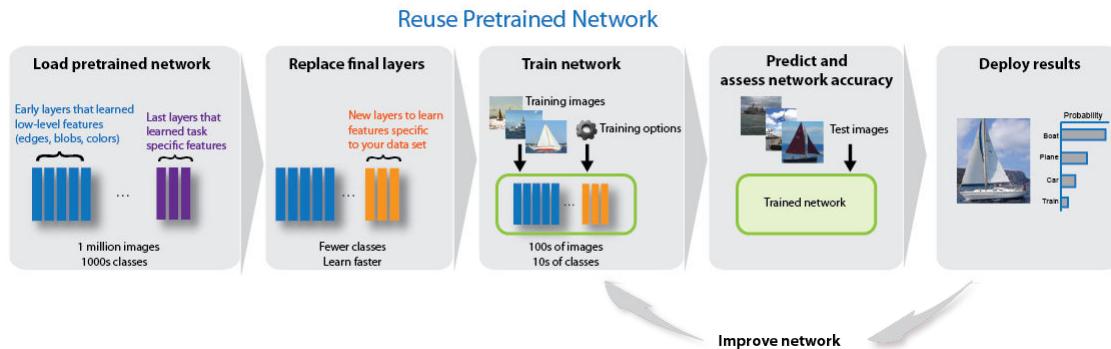
ResNet 通常是合适的特征提取器。有关如何使用预训练网络进行特征提取的示例，请参阅“[使用预训练网络提取图像特征](#)”（第 3-27 页）。

## 迁移学习

您可以通过基于新数据集对网络进行训练来微调网络中的更深层，并以该预训练网络为起点。通过迁移学习来微调网络通常比构建和训练新网络更快更容易。网络已学习到一系列丰富的图像特征，但当您微调网络时，它可以学习特定于您的新数据集的特征。如果您有超大型数据集，则迁移学习可能不会比从头开始训练更快。

**提示** 微调网络通常能达到最高的准确度。对于非常小的数据集（每个类不到 20 个图像），请尝试使用特征提取。

与简单的特征提取相比，微调网络会更慢，需要完成的工作更多，但由于网络可以学习提取不同的特征集，最终的网络通常更准确。只要新数据集不是特别小，微调通常比特征提取效果更好，因为微调时网络有数据可供学习新特征。有关如何执行迁移学习的示例，请参阅“使用深度网络设计器进行迁移学习”（第 2-2 页）和“训练深度学习网络以对新图像进行分类”（第 3-6 页）。



## 导入和导出网络

您可以从 TensorFlow®-Keras、Caffe 和 ONNX™（开放式神经网络交换）模型格式导入网络和网络架构。您还可以将经过训练的网络导出为 ONNX 模型格式。

### 从 Keras 导入

使用 `importKerasNetwork` 从 TensorFlow-Keras 导入预训练网络。您可以从同一个 HDF5 (.h5) 文件或单独的 HDF5 和 JSON (.json) 文件导入网络和权重。有关详细信息，请参阅 `importKerasNetwork`。

使用 `importKerasLayers` 从 TensorFlow-Keras 导入网络架构。您可以导入网络架构，使用或不使用权重均可。您可以从同一个 HDF5 (.h5) 文件或单独的 HDF5 和 JSON (.json) 文件导入网络架构和权重。有关详细信息，请参阅 `importKerasLayers`。

### 从 Caffe 导入

使用 `importCaffeNetwork` 函数从 Caffe 中导入预训练网络。Caffe Model Zoo 中提供许多可用的预训练网络 [5]。下载所需的 .prototxt 和 .caffemodel 文件，并使用 `importCaffeNetwork` 将预训练网络导入 MATLAB。有关详细信息，请参阅 `importCaffeNetwork`。

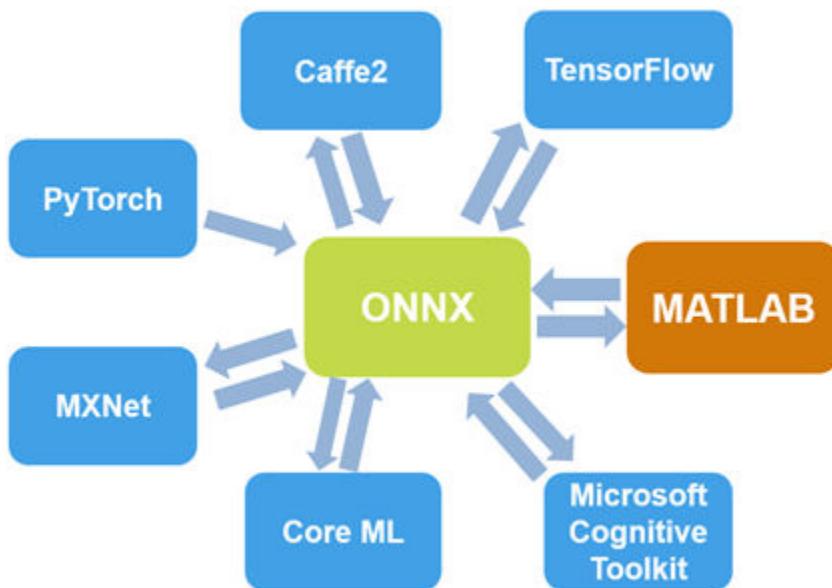
您可以导入 Caffe 网络的网络架构。下载所需的 .prototxt 文件，并使用 `importCaffeLayers` 将网络层导入 MATLAB。有关详细信息，请参阅 `importCaffeLayers`。

### 在 ONNX 中导入和导出

通过使用 ONNX 作为中间格式，您可以与支持 ONNX 模型导出或导入的其他深度学习框架进行互操作，这些框架包括 TensorFlow、PyTorch、Caffe2、Microsoft® Cognitive Toolkit (CNTK)、Core ML 和 Apache MXNet。

使用 `exportONNXNetwork` 函数将经过训练的 Deep Learning Toolbox 网络导出为 ONNX 模型格式。然后，您可以将 ONNX 模型导入支持 ONNX 模型导入的其他深度学习框架中。

使用 `importONNXNetwork` 从 ONNX 导入预训练网络，并使用 `importONNXLayers` 导入带或不带权重的网络架构。



## 音频应用的预训练网络

通过将 Deep Learning Toolbox 与 Audio Toolbox™ 结合使用，将预训练网络用于音频和语音处理应用。

Audio Toolbox 提供预训练的 VGGish 和 YAMNet 网络。使用 `vggish` 和 `yamnet` 函数直接与预训练网络交互。`classifySound` 函数为 YAMNet 执行所需的预处理和后处理，以便您可以定位声音并将其划分到 521 个类别中的一个。您可以使用 `yamnetGraph` 函数探查 YAMNet 本体。`vggishFeatures` 函数为 VGGish 执行必要的预处理和后处理，以便您可以提取特征嵌入，以输入到机器学习和深度学习系统。有关在音频应用中使用深度学习的详细信息，请参阅 “Introduction to Deep Learning for Audio Applications” (Audio Toolbox)。

使用 VGGish 和 YAMNet 执行迁移学习和特征提取。有关示例，请参阅 “Transfer Learning with Pretrained Audio Networks” (Audio Toolbox)。

## 参考

- [1] *ImageNet*. <http://www.image-net.org>
- [2] Russakovsky, O., Deng, J., Su, H., et al. "ImageNet Large Scale Visual Recognition Challenge." *International Journal of Computer Vision (IJCV)*. Vol 115, Issue 3, 2015, pp. 211–252
- [3] Zhou, Bolei, Aditya Khosla, Agata Lapedriza, Antonio Torralba, and Aude Oliva. "Places: An image database for deep scene understanding." arXiv preprint arXiv:1610.02055 (2016).
- [4] *Places*. <http://places2.csail.mit.edu/>
- [5] *Caffe Model Zoo*. [http://caffe.berkeleyvision.org/model\\_zoo.html](http://caffe.berkeleyvision.org/model_zoo.html)

## 另请参阅

`alexnet | googlenet | inceptionv3 | densenet201 | darknet19 | darknet53 | resnet18 |  
resnet50 | resnet101 | vgg16 | vgg19 | shufflenet | nasnetmobile | nasnetlarge | mobilenetv2 |  
xception | inceptionresnetv2 | squeezezenet | importCaffeNetwork | importCaffeLayers |  
importKerasLayers | importKerasNetwork | exportONNXNetwork | importONNXLayers |  
importONNXNetwork | 深度网络设计器`

## 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “使用深度网络设计器进行迁移学习” (第 2-2 页)
- “使用预训练网络提取图像特征” (第 3-27 页)
- “使用 GoogLeNet 对图像进行分类” (第 3-22 页)
- “训练深度学习网络以对新图像进行分类” (第 3-6 页)
- “可视化卷积神经网络的特征” (第 5-46 页)
- “可视化卷积神经网络的激活区域” (第 5-35 页)
- “使用 GoogLeNet 的 Deep Dream 图像” (第 5-2 页)

## 了解卷积神经网络

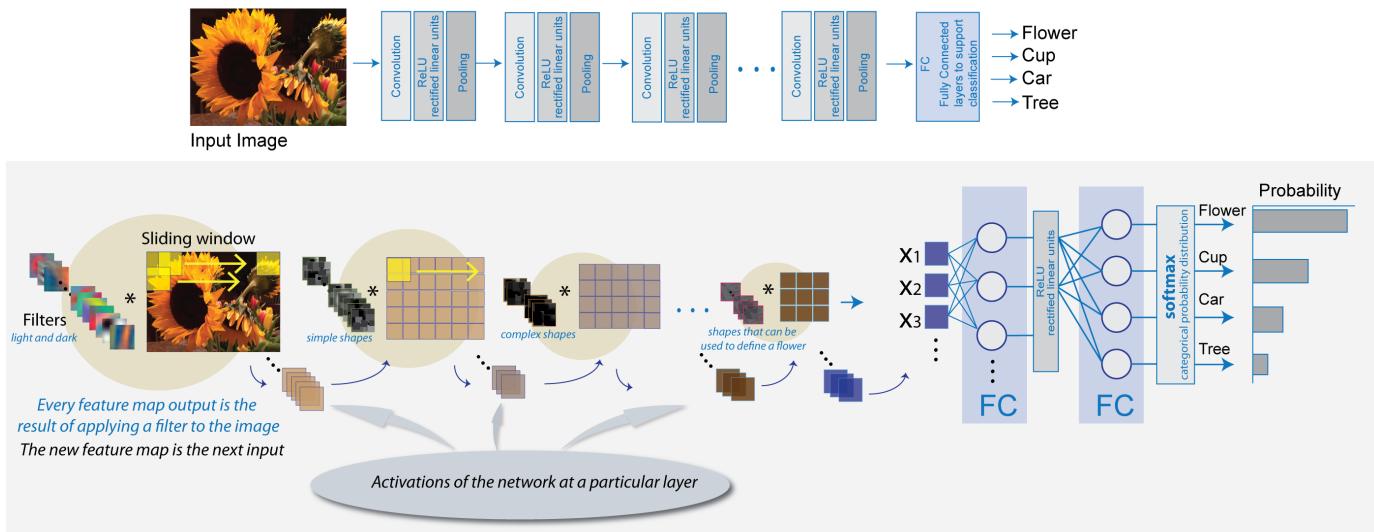
卷积神经网络 (ConvNet) 是深度学习中广泛使用的工具。它们特别适用于作为输入的图像，但它们也适用于其他应用情形，如文本、信号和其他连续响应。卷积神经网络在几个方面不同于其他类型的神经网络：

卷积神经网络的灵感来自视觉皮层的生物结构，视觉皮层包含简单和复杂的细胞排列 [1]。人们发现这些细胞基于视野的子区域而激活。这些子区域称为感受野。受这项研究发现的启发，卷积层中的神经元连接到该层之前各层的子区域，而不是像其他类型的神经网络那样进行全连接。神经元对图像中这些子区域之外的区域没有响应。

这些子区域可能重叠，因此 ConvNet 的神经元会产生在空间上相关的结果，而在其他类型的神经网络中，神经元不共享任何连接并产生独立的结果。

此外，在具有全连接神经元的神经网络中，参数（权重）的数量可能随着输入大小的增加而快速增加。卷积神经网络通过减少连接数量、共享权重和下采样来减少参数个数。

一个 ConvNet 由多个层组成，例如卷积层、最大池化层或平均池化层以及全连接层。



ConvNet 的每层中的神经元以三维方式排列，从而将三维输入变换为三维输出。例如，对于图像输入，第一层（输入层）将图像保留为三维输入，维度分别为图像的高度、宽度和颜色通道。第一个卷积层中的神经元连接到这些图像的区域，并将它们变换为一个三维输出。每层中的隐藏单元（神经元）学习原始输入的非线性组合，这称为特征提取 [2]。这些从一个层学到的特征（也称为激活）成为下一个层的输入。最后，学到的特征成为网络末尾的分类器或回归函数的输入。

根据所包含的层的类型和数量，ConvNet 的架构可能有所不同。包含的层的类型和数量取决于特定的应用或数据。例如，如果您有分类响应，则必须有分类函数和分类层；而如果您的响应是连续的，则必须在网络末尾有回归层。只包含一个或两个卷积层的较小网络可能足以学习少量灰度图像数据。另一方面，对于具有数百万彩色图像的更复杂数据，您可能需要包含多个卷积层和全连接层的更复杂的网络。

您可以通过以下方式在 MATLAB 中串联卷积神经网络的各层：

```
layers = [imageInputLayer([28 28 1])
          convolution2dLayer(5,20)
          reluLayer]
```

```
maxPooling2dLayer(2,'Stride',2)
fullyConnectedLayer(10)
softmaxLayer
classificationLayer];
```

在定义网络的层后，您必须使用 `trainingOptions` 函数指定训练选项。例如，

```
options = trainingOptions('sgdm');
```

然后，您可以使用 `trainNetwork` 函数用您的训练数据对网络进行训练。数据、层和训练选项成为训练函数的输入。例如，

```
convnet = trainNetwork(data,layers,options);
```

有关 ConvNet 各层的详细讨论，请参阅“指定卷积神经网络的层”（第 1-25 页）。要设置训练参数，请参阅“设置参数并训练卷积神经网络”（第 1-35 页）。

## 参考

[1] Hubel, H. D. and Wiesel, T. N. "Receptive Fields of Single neurones in the Cat's Striate Cortex." *Journal of Physiology*. Vol 148, pp. 574-591, 1959.

[2] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. Cambridge, Massachusetts: The MIT Press, 2012.

## 另请参阅

`trainNetwork` | `trainingOptions`

## 详细信息

- “在 MATLAB 中进行深度学习”（第 1-2 页）
- “指定卷积神经网络的层”（第 1-25 页）
- “设置参数并训练卷积神经网络”（第 1-35 页）
- “迁移学习快速入门”
- “创建简单的深度学习网络以用于分类”（第 3-39 页）
- “预训练的深度神经网络”（第 1-8 页）

## 深度学习层列表

本页提供 MATLAB 中的深度学习层的列表。

要了解如何针对不同任务从层创建网络，请参阅以下示例。

任务	了解更多信息
为图像分类或回归创建深度学习网络。	<a href="#">“创建简单的深度学习网络以用于分类” (第 3-39 页)</a> <a href="#">“针对回归训练卷积神经网络” (第 3-44 页)</a> <a href="#">“训练残差网络进行图像分类” (第 3-13 页)</a>
创建深度学习网络以处理序列和时序数据。	<a href="#">“使用深度学习进行序列分类” (第 4-2 页)</a> <a href="#">“使用深度学习进行时序预测” (第 4-9 页)</a>
为音频数据创建深度学习网络。	<a href="#">“使用深度学习进行语音命令识别” (第 4-16 页)</a>
为文本数据创建深度学习网络。	<a href="#">“使用深度学习对文本数据进行分类” (第 4-55 页)</a> <a href="#">“使用深度学习生成文本” (第 4-72 页)</a>

## 深度学习层

使用以下函数创建不同层类型。或者，使用**深度网络设计器**以交互方式创建网络。

要了解如何定义您自己的自定义层，请参阅 “[定义自定义深度学习层](#)” (第 17-2 页)。

### 输入层

层	说明
 <code>imageInputLayer</code>	图像输入层向网络输入二维图像，并应用数据归一化。
 <code>image3dInputLayer</code>	三维图像输入层向网络输入三维图像或三维体，并应用数据归一化。
 <code>sequenceInputLayer</code>	序列输入层向网络输入序列数据。
 <code>featureInputLayer</code>	特征输入层将特征数据输入网络并应用数据归一化。当您有表示特征的数值标量数据集（数据没有空间和时间维度）时，请使用此层。
 <code>roiInputLayer</code>	ROI 输入层将图像输入到 Fast R-CNN 目标检测网络。

## 卷积和全连接层

层	说明
 convolution2dLayer	二维卷积层对输入应用滑动卷积滤波器。
 convolution3dLayer	三维卷积层将滑动立方体卷积滤波器应用于三维输入。
 groupedConvolution2dLayer	二维分组卷积层将输入通道分成各个组，并应用滑动卷积滤波器。使用分组卷积层进行按通道可分离（也称为按深度可分离）卷积。
 transposedConv2dLayer	转置的二维卷积层对特征图进行上采样。
 transposedConv3dLayer	转置的三维卷积层对三维特征图进行上采样。
 fullyConnectedLayer	全连接层将输入乘以权重矩阵，然后添加偏置向量。

## 序列层

层	说明
 sequenceInputLayer	序列输入层向网络输入序列数据。
 lstmLayer	LSTM 层学习时序和序列数据中时间步之间的长期相关性。
 bilstmLayer	双向 LSTM (BiLSTM) 层学习时序或序列数据的时间步之间的双向长期相关性。当您希望网络在每个时间步从完整时序中学习时，这些相关性会很有用。
 gruLayer	GRU 层学习时序和序列数据中时间步之间的相关性。
 sequenceFoldingLayer	序列折叠层将一批图像序列转换为一批图像。使用序列折叠层独立地对图像序列的时间步执行卷积运算。
 sequenceUnfoldingLayer	序列展开层在序列折叠后还原输入数据的序列结构。
 flattenLayer	扁平化层将输入的空间维度折叠成通道维度。
 wordEmbeddingLayer	单词嵌入层将单词索引映射到向量。

## 激活层

层	说明
 <b>reluLayer</b>	ReLU 层对输入的每个元素执行阈值运算，其中任何小于零的值都设置为零。
 <b>leakyReluLayer</b>	泄漏 ReLU 层执行阈值运算，其中小于零的任何输入值都乘以固定标量。
 <b>clippedReluLayer</b>	裁剪 ReLU 层执行阈值运算，其中任何小于零的输入值都设置为零，裁剪上限以上的任何值都设置为该裁剪上限。
 <b>eluLayer</b>	ELU 激活层对正输入执行单位运算，对负输入执行指数非线性运算。
 <b>tanhLayer</b>	双曲正切 (tanh) 激活层对层输入应用 tanh 函数。
 <b>swishLayer</b>	swish 激活层对层输入应用 swish 函数。
 <b>preluLayer (自定义层示例)</b>	PReLU 层执行阈值运算，其中对于每个通道，任何小于零的输入值都乘以在训练时学习到的标量。

## 归一化、丢弃和裁剪层

层	说明
 <b>batchNormalizationLayer</b>	批量归一化层独立地对每个通道的所有观测值的小批量数据进行归一化。为了加快卷积神经网络的训练并降低对网络初始化的敏感度，请在卷积层和非线性部分之间使用批量归一化层，如 ReLU 层。
 <b>groupNormalizationLayer</b>	组归一化层独立地对每个观测值的通道分组子集的小批量数据进行归一化。为了加快卷积神经网络的训练并降低对网络初始化的敏感度，请在卷积层和非线性部分之间使用组归一化层，如 ReLU 层。
 <b>instanceNormalizationLayer</b>	实例归一化层独立地对每个观测值的每个通道上的小批量数据进行归一化。为了改进训练卷积神经网络的收敛性并降低对网络超参数的敏感度，请在卷积层和非线性部分之间使用实例归一化层，如 ReLU 层。
 <b>layerNormalizationLayer</b>	层归一化层独立地对每个观测值的所有通道上的小批量数据进行归一化。为了加快循环和多层次感知神经网络的训练并降低对网络初始化的敏感度，请在可学习层后使用层归一化层，例如 LSTM 层和全连接层。
 <b>crossChannelNormalizationLayer</b>	按通道局部响应（跨通道）归一化层执行按通道归一化。
 <b>dropoutLayer</b>	丢弃层以给定的概率将输入元素随机设置为零。

层	说明
 crop2dLayer	二维裁剪层对输入应用二维裁剪。
 crop3dLayer	三维裁剪层将三维体裁剪到输入特征图的大小。
 resize2dLayer	二维调整大小层根据缩放因子将二维输入的大小调整到指定的高度和宽度，或调整到引用输入特征图的大小。
 resize3dLayer	三维调整大小层根据缩放因子将三维输入的大小调整到指定的高度、宽度和深度，或调整到引用输入特征图的大小。

### 池化和去池化层

层	说明
 averagePooling2dLayer	平均池化层通过将输入划分为矩形池化区域并计算每个区域的平均值来执行下采样。
 averagePooling3dLayer	三维平均池化层通过将三维输入划分为立方体池化区域并计算每个区域的平均值来执行下采样。
 globalAveragePooling2dLayer	全局平均池化层通过计算输入的高度和宽度维度的均值来执行下采样。
 globalAveragePooling3dLayer	三维全局平均池化层通过计算输入的高度、宽度和深度维度的均值来执行下采样。
 maxPooling2dLayer	最大池化层通过将输入划分为矩形池化区域并计算每个区域的最大值来执行下采样。
 maxPooling3dLayer	三维最大池化层通过将三维输入划分为立方体池化区域并计算每个区域的最大值来执行下采样。
 globalMaxPooling2dLayer	全局最大池化层通过计算输入的高度和宽度维度的最大值来执行下采样。
 globalMaxPooling3dLayer	三维全局最大池化层通过计算输入的高度、宽度和深度维度的最大值来执行下采样。
 maxUnpooling2dLayer	最大去池化层对最大池化层的输出进行去池化。

### 组合层

层	说明
 additionLayer	相加层按元素将来自多个神经网络层的输入相加。
 multiplicationLayer	相乘层按元素将来自多个神经网络层的输入相乘。
 depthConcatenationLayer	深度串联层接受具有相同高度和宽度的输入，并沿第三个维度（通道维度）串联它们。

层	说明
 concatenationLayer	串联层接受输入并沿指定维度串联它们。除串联维度外，所有其他维度中的输入必须具有相同的大 小。
 weightedAdditionLayer (自定义层示例)	加权相加层按元素对来自多个神经网络层的输入进 行缩放和相加。

### 目标检测层

层	说明
 roiInputLayer	ROI 输入层将图像输入到 Fast R-CNN 目标检测网 络。
 roiMaxPooling2dLayer	ROI 最大池化层对输入特征图中的每个矩形 ROI 输 出固定大小的特征图。使用此层创建一个 Fast 或 Faster R-CNN 目标检测网络。
 roiAlignLayer	ROI 对齐层对输入特征图中的每个矩形 ROI 输出固 定大小的特征图。使用此层创建一个 Mask-RCNN 网络。
 anchorBoxLayer	锚框层存储用于目标检测网络中特征图的锚框。
 regionProposalLayer	区域提议层是 Faster R-CNN 中区域提议网络 (RPN) 的一部分，该层输出图像中潜在目标周围的 边界框。
 ssdMergeLayer	SSD 合并层合并特征图的输出，用于后续回归和分 类损失计算。
 spaceToDepthLayer	空间到深度层将输入的空间模块置换到深度维度。 当您需要在不丢弃任何特征数据的情况下合并不同 大小的特征图时，请使用此层。
 depthToSpace2dLayer	二维深度到空间层将数据从深度维度置换到二维空 间数据块。
 rpnSoftmaxLayer	区域提议网络 (RPN) softmax 层对输入应用 softmax 激活函数。使用此层创建一个 Faster R- CNN 目标检测网络。
 focalLossLayer	焦点损失层使用焦点损失预测目标类。
 rpnClassificationLayer	区域提议网络 (RPN) 分类层通过使用交叉熵损失函 数将图像区域分类为目标或背景。使用此层创建一 个 Faster R-CNN 目标检测网络。
 rcnnBoxRegressionLayer	框回归层通过使用平滑 L1 损失函数来细化边界框位 置。使用此层创建一个 Fast 或 Faster R-CNN 目标 检测网络。

## 生成对抗网络层

层	说明
 <b>projectAndReshapeLayer</b> (第 3-57 页) (自定义层示例)	投影和重构层接受 $1 \times 1 \times \text{numLatentInputs}$ 数组作为输入，并将它们转换为指定大小的图像。使用投影和重构层来重构 GAN 的噪声输入。
 <b>embedAndReshapeLayer</b> (自定义层示例)	嵌入和重构层接受分类元素的数值索引作为输入，并将其转换为指定大小的图像。使用嵌入和重构层将分类数据输入到条件 GAN 中。

## 输出层

层	说明
 <b>softmaxLayer</b>	softmax 层对输入应用 softmax 函数。
 <b>sigmoidLayer</b>	sigmoid 层对输入应用 sigmoid 函数，使得输出在区间 (0,1) 内有界。
 <b>classificationLayer</b>	分类层计算具有互斥类的分类和加权分类任务的交叉熵损失。
 <b>regressionLayer</b>	回归层计算回归任务的平均方误差损失。
 <b>pixelClassificationLayer</b>	像素分类层为每个图像像素或体素提供分类标签。
 <b>dicePixelClassificationLayer</b>	Dice 像素分类层使用广义 Dice 损失为每个图像像素或体素提供分类标签。
 <b>focalLossLayer</b>	焦点损失层使用焦点损失预测目标类。
 <b>rpnSoftmaxLayer</b>	区域提议网络 (RPN) softmax 层对输入应用 softmax 激活函数。使用此层创建一个 Faster R-CNN 目标检测网络。
 <b>rpnClassificationLayer</b>	区域提议网络 (RPN) 分类层通过使用交叉熵损失函数将图像区域分类为目标或背景。使用此层创建一个 Faster R-CNN 目标检测网络。
 <b>rcnnBoxRegressionLayer</b>	框回归层通过使用平滑 L1 损失函数来细化边界框位置。使用此层创建一个 Fast 或 Faster R-CNN 目标检测网络。
 <b>tverskyPixelClassificationLayer</b> (第 8-59 页) (自定义层示例)	Tversky 像素分类层使用 Tversky 损失为每个图像像素或体素提供分类标签。
 <b>sseClassificationLayer</b> (自定义层示例)	分类 SSE 层计算分类问题的误差损失平方和。
 <b>maeRegressionLayer</b> (自定义层示例)	回归 MAE 层计算回归问题的均值绝对误差损失。

## 另请参阅

[trainingOptions](#) | [trainNetwork](#) | [深度网络设计器](#)

## 详细信息

- “使用深度网络设计器构建网络” (第 2-15 页)
- “指定卷积神经网络的层” (第 1-25 页)
- “设置参数并训练卷积神经网络” (第 1-35 页)
- “定义自定义深度学习层” (第 17-2 页)
- “创建简单的深度学习网络以用于分类” (第 3-39 页)
- “使用深度学习进行序列分类” (第 4-2 页)
- “预训练的深度神经网络” (第 1-8 页)
- “Deep Learning Tips and Tricks”

# 指定卷积神经网络的层

## 本节内容

- “图像输入层”（第 1-25 页）
- “卷积层”（第 1-26 页）
- “批量归一化层”（第 1-30 页）
- “ReLU 层”（第 1-30 页）
- “跨通道归一化（局部响应归一化）层”（第 1-30 页）
- “最大池化层和平均池化层”（第 1-31 页）
- “丢弃层”（第 1-31 页）
- “全连接层”（第 1-31 页）
- “输出层”（第 1-32 页）

创建和训练新卷积神经网络 (ConvNet) 的第一步是定义网络架构。本主题说明 ConvNet 层的细节，以及这些层在 ConvNet 中出现的顺序。有关深度学习层的完整列表以及如何创建它们，请参阅“深度学习层列表”（第 1-18 页）。要了解序列分类和回归的 LSTM 网络，请参阅“长短期记忆网络”（第 1-38 页）。要了解如何创建自己的自定义层，请参阅“定义自定义深度学习层”（第 17-2 页）。

根据所包含的层的类型和数量，网络架构可能有所不同。包含的层的类型和数量取决于特定的应用或数据。例如，如果您有分类响应，您必须有 softmax 层和分类层，而如果您的响应是连续的，您必须在网络末尾有回归层。只包含一个或两个卷积层的较小网络可能基于少量灰度图像数据进行学习就足够了。另一方面，对于具有数百万彩色图像的更复杂数据，您可能需要包含多个卷积层和全连接层的更复杂的网络。

要指定所有层按顺序连接的深度网络的架构，请直接创建一个层数组。例如，要创建一个将  $28 \times 28$  灰度图像分为 10 个类的深度网络，请指定层数组。

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,16,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,32,'Padding',1)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

`layers` 是 `Layer` 对象组成的数组。然后，您可以使用 `layers` 作为训练函数 `trainNetwork` 的输入。

要指定所有层按顺序连接的神经网络的架构，请直接创建一个层数组。要指定层可以有多个输入或输出的网络架构，请使用 `LayerGraph` 对象。

## 图像输入层

使用 `imageInputLayer` 创建一个图像输入层。

图像输入层将图像输入到网络并应用数据归一化。

使用 `inputSize` 参数指定图像大小。图像的大小对应于该图像的高度、宽度和颜色通道的数量。例如，对于灰度图像，通道数为 1，而对于彩色图像，通道数为 3。

## 卷积层

二维卷积层对输入应用滑动卷积滤波器。使用 `convolution2dLayer` 创建一个二维卷积层。

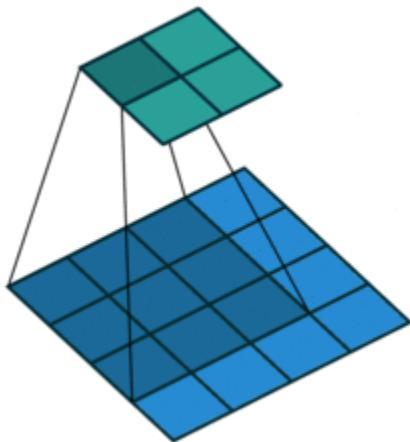
卷积层包含各种组成部分。<sup>1</sup>

### 滤波器和步幅

卷积层由连接到输入图像的子区域或连接到前一层的输出的神经元组成。该层在扫描图像的同时学习由这些区域所局部化的特征。使用 `convolution2dLayer` 函数创建层时，您可以使用 `filterSize` 输入参数指定这些区域的大小。

对于每个区域，`trainNetwork` 函数计算权重和输入的点积，然后加上偏置项。应用于图像中某个区域的一组权重称为滤波器。滤波器沿输入图像垂直和水平移动，对每个区域重复相同的计算。换句话说，滤波器对输入进行卷积。

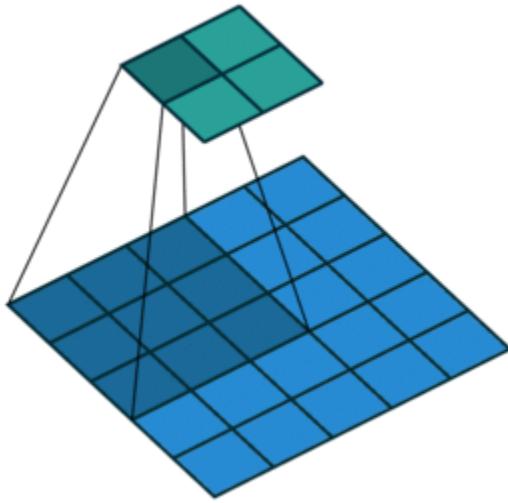
下图显示一个扫描输入的  $3 \times 3$  滤波器。下部子图表示输入，上部子图表示输出。



滤波器移动的步长大小称为步幅。您可以使用 `Stride` 名称-值对组参数指定步长大小。神经元连接的局部区域可能会重叠，具体取决于 `filterSize` 和 'Stride' 值。

下图显示一个扫描输入的  $3 \times 3$  滤波器，步幅为 2。下部子图表示输入，上部子图表示输出。

1. Image credit: Convolution arithmetic (License)



滤波器中的权重数为  $h \times w \times c$ ，其中  $h$  是高度， $w$  是滤波器的宽度， $c$  是输入中的通道数。例如，如果输入是彩色图像，则颜色通道的数量是 3。滤波器的数量决定卷积层输出中的通道数量。使用 `numFilters` 参数和 `convolution2dLayer` 函数指定滤波器的数量。

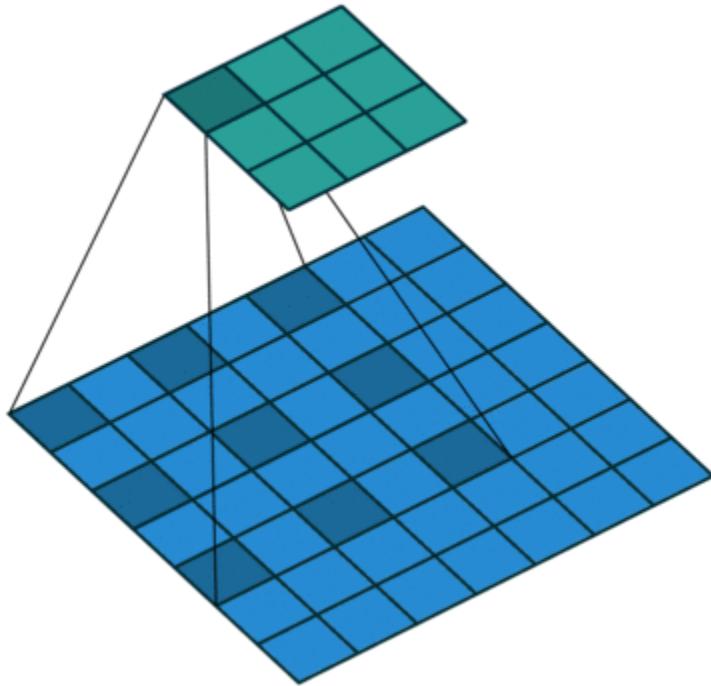
### 扩张卷积

在扩张卷积中，通过在滤波器元素之间插入空白来扩展滤波器。使用 `'DilationFactor'` 属性指定扩张系数。

使用扩张卷积可增加层的感受野（层可以看到的输入区域），而不增加参数个数或计算量。

层通过在每个滤波器元素之间插入零来扩展滤波器。扩张系数决定对输入进行采样的步长大小，或等效地决定滤波器的上采样因子。它对应的有效滤波器大小为： $(\text{Filter Size} - 1) \cdot \text{Dilation Factor} + 1$ 。例如，扩张系数为 [2 2] 的  $3 \times 3$  滤波器等效于在元素间插入零的  $5 \times 5$  滤波器。

下图显示一个使用扩张系数 2 扫描输入来扩张的  $3 \times 3$  滤波器。下部子图表示输入，上部子图表示输出。



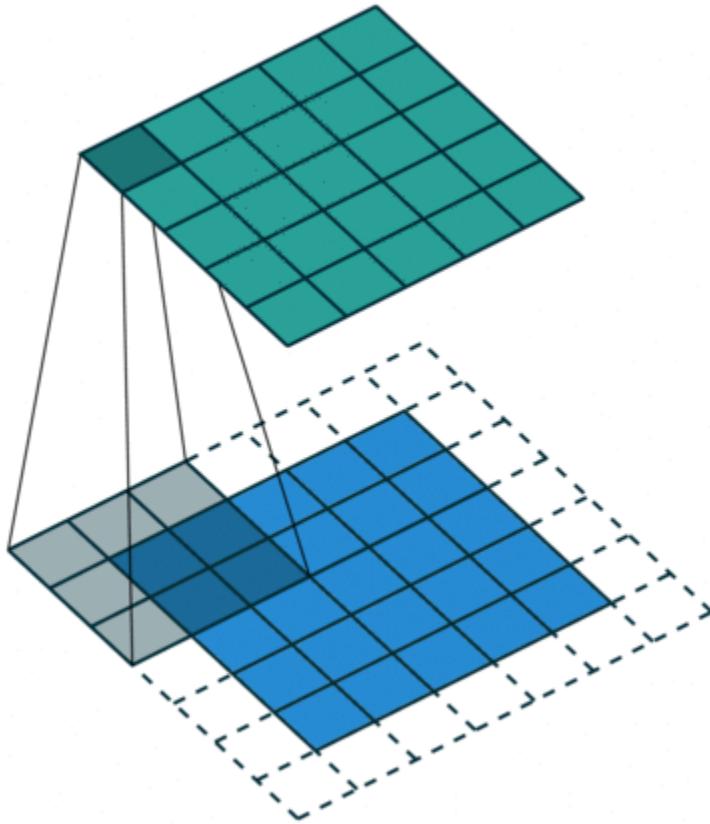
### 特征图

当滤波器扫描输入时，它对卷积使用相同的权重和偏置，从而形成特征图。每个特征图均为使用不同权重集和不同偏置的卷积结果。因此，特征图的数量等于滤波器的数量。卷积层中的参数总数为  $((h \times w \times c + 1) \times \text{Number of Filters})$ ，其中 1 为偏置。

### 填充

您还可以使用 'Padding' 名称-值对组参数在垂直和水平方向上对输入图像边界应用填充。填充是指将向输入图像的边界追加值以增大其大小。通过调整零填充，您可以控制层的输出大小。

下图显示一个用于扫描输入的  $3 \times 3$  滤波器，其零填充大小为 1。下部子图表示输入，上部子图表示输出。



## 输出大小

卷积层的输出高度和宽度为  $(\text{Input Size} - ((\text{Filter Size} - 1) * \text{Dilation Factor} + 1) + 2 * \text{Padding}) / \text{Stride} + 1$ 。此值必须为整数，才能完全覆盖整个图像。如果这些选项的组合无法完全覆盖图像，则默认情况下，软件会忽略卷积中沿图像右边缘和下边缘的其余部分。

## 神经元数量

输出高度和宽度的乘积是特征图中神经元的总数，即 Map Size。卷积层中的神经元总数（输出大小）是 Map Size \* Number of Filters。

例如，假设输入图像是一个  $32 \times 32 \times 3$  彩色图像。对于具有 8 个滤波器且滤波器大小为  $5 \times 5$  的卷积层，每个滤波器的权重数是  $5 \times 5 \times 3 = 75$ ，并且该层中的参数总数是  $(75 + 1) * 8 = 608$ 。如果每个方向的步幅为 2，并且指定的填充大小为 2，则每个特征图为  $16 \times 16$ 。这是因为  $(32 - 5 + 2 * 2)/2 + 1 = 16.5$ ，图像右侧和底部一些最外层的填充被丢弃。最后，该层中神经元的总数是  $16 * 16 * 8 = 2048$ 。

通常，来自这些神经元的结果以某种非线性（例如修正线性单元 (ReLU)）形式进行传递。

## 学习参数

在定义卷积层时，您可以使用名称-值对组参数来调整层的学习率和正则化选项。如果您选择不指定这些选项，则 `trainNetwork` 将使用通过 `trainingOptions` 函数定义的全局训练选项。有关全局和层训练选项的详细信息，请参阅“设置参数并训练卷积神经网络”（第 1-35 页）。

## 层数

一个卷积神经网络可以包含一个或多个卷积层。卷积层的数量取决于数据的量和复/实性。

## 批量归一化层

使用 **batchNormalizationLayer** 创建一个批量归一化层。

批量归一化层独立地对每个通道的所有观测值的小批量数据进行归一化。为了加快卷积神经网络的训练并降低对网络初始化的敏感度，请在卷积层和非线性部分之间使用批量归一化层，如 ReLU 层。

该层首先通过减去小批量均值并除以小批量标准差来归一化每个通道的激活。然后，该层按可学习偏移量  $\beta$  对输入进行偏移，并按可学习缩放因子  $\gamma$  对其进行缩放。 $\beta$  和  $\gamma$  本身就是在网络训练期间会更新的可学习参数。

批量归一化层对神经网络中的激活值和梯度传播进行归一化，使网络训练成为更简单的优化问题。为了充分利用这一点，您可以尝试提高学习率。由于优化问题更容易，参数更新可以更大，网络学习的速度更快。您也可以尝试减少  $L_2$  和丢弃正则化。对于批量归一化层，特定图像在训练期间的激活取决于哪些图像碰巧出现在同一小批量中。为了充分利用这种正则化效果，请尝试在每轮训练之前对训练数据进行乱序处理。要指定训练期间数据乱序的频率，请使用 **trainingOptions** 的 'Shuffle' 名称-值对组参数。

## ReLU 层

使用 **reluLayer** 创建一个 ReLU 层。

ReLU 层对输入的每个元素执行阈值运算，其中任何小于零的值都设置为零。

卷积层和批量归一化层通常后跟非线性激活函数，例如由 ReLU 层指定的修正线性单元 (ReLU)。ReLU 层对每个元素执行阈值运算，其中任何小于零的输入值都设置为零，即

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

ReLU 层不更改其输入的大小。

还有其他执行不同运算的非线性激活层，这些层可以提高某些应用的网络准确度。有关激活层的列表，请参阅“激活层”（第 1-20 页）。

## 跨通道归一化（局部响应归一化）层

使用 **crossChannelNormalizationLayer** 创建一个跨通道归一化层。

按通道局部响应（跨通道）归一化层执行按通道归一化。

此层执行按通道局部响应归一化。它通常在 ReLU 激活层之后。此层将每个元素替换为它使用来自一定数量的相邻通道的元素（归一化窗口中的元素）获得的归一化值。也就是说，对于输入中的每个元素  $x$ ，**trainNetwork** 使用

$$x' = \frac{x}{\left(K + \frac{\alpha * ss}{windowChannelSize}\right)^{\beta}}$$

计算归一化值  $x'$  其中  $K$ 、 $\alpha$  和  $\beta$  是归一化中的超参数， $ss$  是归一化窗口中元素的平方和 [2]。您必须使用 `crossChannelNormalizationLayer` 函数的 `windowChannelSize` 参数来指定归一化窗口的大小。您还可以使用 `Alpha`、`Beta` 和 `K` 名称-值对组参数来指定超参数。

前面的归一化公式与 [2] 中给出的公式略有不同。您可以通过将 `alpha` 值乘以 `windowChannelSize` 获得等效公式。

## 最大池化层和平均池化层

最大池化层通过将输入划分为矩形池化区域并计算每个区域的最大值来执行下采样。使用 `maxPooling2dLayer` 创建一个最大池化层。

平均池化层通过将输入划分为矩形池化区域并计算每个区域的平均值来执行下采样。使用 `averagePooling2dLayer` 创建一个平均池化层。

池化层在卷积层之后，用于下采样，因此可减少与后续层的连接数量。池化层自身不执行任何学习，但可减少要在后续层中学习的参数的数量。它们也有助于减少过拟合。

最大池化层返回其输入的矩形区域的最大值。矩形区域的大小由 `maxPoolingLayer` 的 `poolSize` 参数决定。例如，如果 `poolSize` 等于 `[2,3]`，则层返回高度为 2 和宽度为 3 的区域中的最大值。平均池化层输出其输入的矩形区域的平均值。矩形区域的大小由 `averagePoolingLayer` 的 `poolSize` 参数决定。例如，如果 `poolSize` 是 `[2,3]`，则层返回高度为 2、宽度为 3 的区域的平均值。

池化层按照一定的步长在水平和垂直方向扫描输入，您可以使用 '`Stride`' 名称-值对组参数指定步长大小。如果池大小小于或等于步幅，则池化区域不会重叠。

对于非重叠区域（Pool Size 和 Stride 相等），如果池化层的输入是  $n \times n$ ，池化区域大小是  $h \times h$ ，则池化层按  $h$  对区域进行下采样 [6]。也就是说，卷积层的一个通道的最大或平均池化层的输出是  $n/h \times n/h$ 。对于重叠区域，池化层的输出是  $(\text{Input Size} - \text{Pool Size} + 2 * \text{Padding})/\text{Stride} + 1$ 。

## 丢弃层

使用 `dropoutLayer` 创建一个丢弃层。

丢弃层以给定的概率将输入元素随机设置为零。

在训练时，层随机将输入元素设置为由丢弃掩膜 `rand(size(X)) < Probability` 给出的零，其中 `X` 是层输入，然后按  $1/(1-\text{Probability})$  缩放其余元素。该运算实际上更改了迭代之间的底层网络架构，有助于防止网络过拟合 [7]、[2]。数字越大，训练过程中丢弃的元素越多。在预测时，层的输出等于其输入。

与最大池化层或平均池化层相似，此层不进行学习。

## 全连接层

使用 `fullyConnectedLayer` 创建一个全连接层。

全连接层将输入乘以权重矩阵，然后添加偏置向量。

卷积层（和下采样层）后跟一个或多个全连接层。

顾名思义，全连接层中的所有神经元都连接到前一个层中的所有神经元。该层将先前各层在图像中学习的所有特征（局部信息）组合在一起，以识别较大的模式。对于分类问题，最后一个全连接层将特征组合在

一起来对图像进行分类。因此，网络的最后一个全连接层的 `outputSize` 参数等于数据集的类的数目。对于回归问题，输出大小必须等于响应变量的数目。

在创建全连接层时，您还可以使用相关的名称-值对组参数来调整该层的学习率和正则化参数。如果您选择不调整它们，则 `trainNetwork` 将使用 `trainingOptions` 函数定义的全局训练参数。有关全局和层训练选项的详细信息，请参阅“设置参数并训练卷积神经网络”（第 1-35 页）。

全连接层将输入乘以权重矩阵  $W$ ，然后与偏置向量  $b$  相加。

如果层的输入是序列（例如，在 LSTM 网络中），则全连接层独立作用于每个时间步。例如，如果全连接层之前的层输出大小为  $D \times N \times S$  的数组  $X$ ，则全连接层输出大小为  $outputSize \times N \times S$  的数组  $Z$ 。在时间步  $t$  中， $Z$  的对应项是  $WX_t + b$ ，其中  $X_t$  表示  $X$  的时间步  $t$ 。

## 输出层

### softmax 层和分类层

softmax 层对输入应用 softmax 函数。使用 `softmaxLayer` 创建一个 softmax 层。

分类层计算具有互斥类的分类和加权分类任务的交叉熵损失。使用 `classificationLayer` 创建一个分类层。

对于分类问题，必须在最终全连接层后跟 softmax 层，然后跟分类层。

输出单元激活函数是 softmax 函数：

$$y_r(x) = \frac{\exp(a_r(x))}{\sum_{j=1}^k \exp(a_j(x))}$$

其中  $0 \leq y_r \leq 1$  并且  $\sum_{j=1}^k y_j = 1$ 。

对于多类分类问题，softmax 函数是最后一个全连接层后的输出单元激活函数：

$$P(c_r|x, \theta) = \frac{P(x, \theta|c_r)P(c_r)}{\sum_{j=1}^k P(x, \theta|c_j)P(c_j)} = \frac{\exp(a_r(x, \theta))}{\sum_{j=1}^k \exp(a_j(x, \theta))}$$

其中  $0 \leq P(c_r|x, \theta) \leq 1$  且  $\sum_{j=1}^k P(c_j|x, \theta) = 1$ 。此外， $a_r = \ln(P(x, \theta|c_r)P(c_r))$ ，其中  $P(x, \theta|c_r)$  是给定类  $r$  的样本的条件概率， $P(c_r)$  是类先验概率。

softmax 函数也称为归一化指数，可视为逻辑 sigmoid 函数的多类泛化 [8]。

对于典型的分类网络，分类层必须在 softmax 层之后。在分类层中，`trainNetwork` 接受来自 softmax 函数的值，使用交叉熵函数将每个输入赋给  $K$  个互斥类之一以实现 1-of- $K$  编码方案 [8]：

$$\text{loss} = -\frac{1}{N} \sum_{n=1}^N \sum_{i=1}^K w_i t_{ni} \ln y_{ni}$$

其中  $N$  是样本数， $K$  是类数， $w_i$  是类  $i$  的权重， $t_{ni}$  指示第  $n$  个样本属于第  $i$  个类， $y_{ni}$  是类  $i$  的样本  $n$  的输出，在本例中它是来自 softmax 函数的值。换句话说， $y_{ni}$  是网络将第  $n$  个输入与类  $i$  相关联的概率。

## 回归层

使用 **regressionLayer** 创建一个回归层。

回归层计算回归任务的平均方误差损失。对于典型的回归问题，回归层必须位于最终全连接层之后。

对于单个观测值，均方误差由下式给出：

$$\text{MSE} = \sum_{i=1}^R \frac{(t_i - y_i)^2}{R},$$

其中， $R$  是响应的数目， $t_i$  是目标输出， $y_i$  是网络对响应  $i$  的预测。

对于图像和“序列到单个”回归网络，回归层的损失函数是预测响应的平均方误差，未由  $R$  进行归一化：

$$\text{loss} = \frac{1}{2} \sum_{i=1}^R (t_i - y_i)^2.$$

对于图像到图像回归网络，回归层的损失函数是每个像素的预测响应的平均方误差，未由  $R$  进行归一化：

$$\text{loss} = \frac{1}{2} \sum_{p=1}^{HWC} (t_p - y_p)^2,$$

其中  $H$ 、 $W$  和  $C$  分别表示输出的高度、宽度和通道数， $p$  对  $t$  和  $y$  的每个元素（像素）进行线性索引。

对于“序列到序列”回归网络，回归层的损失函数是每个时间步的预测响应的平均方误差，未由  $R$  进行归一化：

$$\text{loss} = \frac{1}{2S} \sum_{i=1}^S \sum_{j=1}^R (t_{ij} - y_{ij})^2,$$

其中  $S$  是序列长度。

在训练时，软件计算小批量中各观测值的均值损失。

## 参考

[1] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. Cambridge, Massachusetts: The MIT Press, 2012.

[2] Krizhevsky, A., I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *Advances in Neural Information Processing Systems*. Vol 25, 2012.

- [3] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D., et al. "Handwritten Digit Recognition with a Back-propagation Network." In *Advances of Neural Information Processing Systems*, 1990.
- [4] LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based Learning Applied to Document Recognition." *Proceedings of the IEEE*. Vol 86, pp. 2278–2324, 1998.
- [5] Nair, V. and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines." In Proc. 27th International Conference on Machine Learning, 2010.
- [6] Nagi, J., F. Ducatelle, G. A. Di Caro, D. Ciresan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, L. M. Gambardella. "Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition". *IEEE International Conference on Signal and Image Processing Applications (ICSI/PA2011)*, 2011.
- [7] Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research*. Vol. 15, pp. 1929-1958, 2014.
- [8] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 2006.
- [9] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." preprint, arXiv:1502.03167 (2015).

## 另请参阅

convolution2dLayer | batchNormalizationLayer | dropoutLayer | averagePooling2dLayer | maxPooling2dLayer | classificationLayer | regressionLayer | softmaxLayer | crossChannelNormalizationLayer | fullyConnectedLayer | reluLayer | leakyReluLayer | clippedReluLayer | imageInputLayer | trainingOptions | trainNetwork

## 详细信息

- “深度学习层列表” (第 1-18 页)
- “了解卷积神经网络” (第 1-16 页)
- “设置参数并训练卷积神经网络” (第 1-35 页)
- “Resume Training from Checkpoint Network”
- “创建简单的深度学习网络以用于分类” (第 3-39 页)
- “预训练的深度神经网络” (第 1-8 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 设置参数并训练卷积神经网络

## 本节内容

- “指定求解器和最大轮数”（第 1-35 页）
- “指定和修改学习率”（第 1-35 页）
- “指定验证数据”（第 1-36 页）
- “选择硬件资源”（第 1-36 页）
- “保存检查点网络并继续训练”（第 1-36 页）
- “在卷积层和全连接层中设置参数”（第 1-37 页）
- “训练网络”（第 1-37 页）

按照“指定卷积神经网络的层”（第 1-25 页）中所述定义神经网络的层后，下一步是为网络设置训练选项。使用 **trainingOptions** 函数定义全局训练参数。要训练网络，请使用 **trainingOptions** 返回的对象作为 **trainNetwork** 函数的输入参数。例如：

```
options = trainingOptions('adam');
trainedNet = trainNetwork(data,layers,options);
```

具有可学习参数的层也有用于调整学习参数的选项。有关详细信息，请参阅“在卷积层和全连接层中设置参数”（第 1-37 页）。

## 指定求解器和最大轮数

**trainNetwork** 可以使用随机梯度下降的不同变体来训练网络。使用 **trainingOptions** 的 **solverName** 参数指定优化算法。为了最小化损失，这些算法通过在损失函数的负梯度方向上采用小步长来更新网络参数。

'adam'（派生自适应矩估计）求解器通常适合作为首先尝试的优化器。您也可以尝试 'rmsprop'（均方根传播）和 'sgdm'（带动量的随机梯度下降）优化器，看看这是否能改进训练。不同的问题最好采用不同的求解器。有关不同求解器的详细信息，请参阅“*Stochastic Gradient Descent*”。

求解器在每一步都使用数据的一个子集来更新参数。此子集称为小批量。您可以使用 **trainingOptions** 的 'MinibatchSize' 名称-值对组参数来指定小批量的大小。每次参数更新都称为一次迭代。对整个数据集的一次完整遍历称为一轮训练。您可以使用 **trainingOptions** 的 'MaxEpochs' 名称-值对组参数来指定要训练的最大轮数。默认值为 30，但您可以为小型网络或微调和迁移学习选择较少的轮数，因为大多数学习已完成。

默认情况下，软件会在训练前对数据进行一次乱序处理。您可以使用 'Shuffle' 名称-值对组参数来更改此设置。

## 指定和修改学习率

您可以使用 **trainingOptions** 的 'InitialLearnRate' 名称-值对组参数来指定全局学习率。默认情况下，**trainNetwork** 在整个训练过程中使用此值。您可以选择在每经过一定的轮数后，就将学习率乘以一个因子来修改学习率。您可以在训练开始时选择较大的学习率，并在优化过程中逐渐降低该值，而不是在整个训练过程中使用较小的固定学习率。这样做可以缩短训练时间，同时随着训练的进行，使得在接近损失最小值的过程中步长不断变小。

**提示** 如果训练期间的小批量损失变为 NaN，则学习率可能过高。请尝试降低学习率，例如降低为三分之一，并重新开始网络训练。

要逐渐降低学习率，请使用 'LearnRateSchedule','piecewise' 名称-值对组参数。如果您选择此选项，`trainNetwork` 会每经过 10 轮训练就将初始学习率乘以因子 0.1。您可以分别使用 'LearnRateDropFactor' 和 'LearnRateDropPeriod' 名称-值对组参数来指定降低初始学习率和轮数的因子。

## 指定验证数据

要在训练期间执行网络验证，请使用 `trainingOptions` 的 'ValidationData' 名称-值对组参数指定验证数据。默认情况下，`trainNetwork` 通过预测验证数据的响应并计算验证损失和准确度（回归网络的均方根误差），每 50 次迭代验证一次网络。您可以使用 'ValidationFrequency' 名称-值对组参数更改验证频率。如果您的网络中的层在预测过程中的行为不同于在训练过程中的行为（例如，丢弃层），则验证准确度可能高于训练（小批量）准确度。您还可以使用验证数据来实现在验证损失停止减少时自动停止训练。要启用自动验证停止，请使用 'ValidationPatience' 名称-值对组参数。

在训练期间以固定间隔执行验证有助于确定您的网络是否在对训练数据进行过拟合。一个常见的问题是，网络只是简单地“记忆”训练数据，而不是学习使网络能够对新数据作出准确预测的一般特征。要检查您的网络是否正在过拟合，请将训练损失和准确度与对应的验证指标进行比较。如果训练损失明显低于验证损失，或训练准确度明显高于验证准确度，则说明您的网络正在过拟合。

为了减少过拟合，您可以尝试添加数据增强功能。使用 `augmentedImageDatastore` 对输入图像执行随机变换。这有助于防止网络记忆目标的准确位置和方向。您还可以尝试使用 'L2Regularization' 名称-值对组参数来增加 L<sub>2</sub> 正则化，在卷积层后使用批量归一化层，以及添加丢弃层。

## 选择硬件资源

默认情况下，如果 GPU 可用，则 `trainNetwork` 会将其用于训练。否则，`trainNetwork` 将使用 CPU。您也可以使用 'ExecutionEnvironment' 名称-值对组参数指定所需的执行环境。您可以指定单个 CPU ('cpu')、单个 GPU ('gpu')、多个 GPU ('multi-gpu')，或本地并行池或计算群集 ('parallel')。除 'cpu' 以外的所有选项都需要 Parallel Computing Toolbox。在 GPU 上训练需要支持的 GPU 设备。有关受支持的设备的信息，请参阅“GPU Support by Release” (Parallel Computing Toolbox)。

## 保存检查点网络并继续训练

Deep Learning Toolbox 使您能够在训练期间的每轮训练后将网络另存为 .mat 文件。当您有大型网络或大型数据集并且训练需要很长时间时，这种定期保存特别有用。如果训练因某种原因中断，您可以从上次保存的检查点网络继续训练。如果希望 `trainNetwork` 保存检查点网络，则您必须使用 `trainingOptions` 的 'CheckpointPath' 名称-值对组参数指定路径名称。如果您指定的路径不存在，则 `trainingOptions` 会返回错误。

`trainNetwork` 自动为检查点网络文件分配唯一名称。在示例名称 `net_checkpoint_351_2018_04_12_18_09_52.mat` 中，351 是迭代编号，2018\_04\_12 是日期，18\_09\_52 是 `trainNetwork` 保存网络的时间。您可以通过双击检查点网络文件或在命令行中使用 `load` 命令来加载该文件。例如：

```
load net_checkpoint_351_2018_04_12_18_09_52.mat
```

然后，您可以使用网络的层作为 `trainNetwork` 的输入参数，继续进行训练。例如：

```
trainNetwork(XTrain,YTrain,net.Layers,options)
```

您必须手动指定训练选项和输入数据，因为检查点网络不包含此信息。有关示例，请参阅“Resume Training from Checkpoint Network”。

## 在卷积层和全连接层中设置参数

您可以在具有可学习参数的层（如卷积层和全连接层）中，将学习参数设置为不同于 `trainingOptions` 指定的全局值。例如，要调整偏置或权重的学习率，您可以分别为层的 `BiasLearnRateFactor` 或 `WeightLearnRateFactor` 属性指定值。`trainNetwork` 函数将您使用 `trainingOptions` 指定的学习率乘以这些因子。同样，您也可以通过分别指定 `BiasL2Factor` 和 `WeightL2Factor` 属性，为这些层中的权重和偏置指定 L<sub>2</sub> 正则化因子。然后，`trainNetwork` 将您使用 `trainingOptions` 指定的 L<sub>2</sub> 正则化因子乘以这些因子。

### 初始化卷积层和全连接层中的权重

层权重是可学习参数。您可以直接使用层的 `Weights` 属性指定权重的初始值。训练网络时，如果层的 `Weights` 属性不为空，则 `trainNetwork` 使用 `Weights` 属性作为初始值。如果 `Weights` 属性为空，则 `trainNetwork` 使用该层的 `WeightsInitializer` 属性指定的初始值。

## 训练网络

指定网络的层和训练参数后，可以使用训练数据训练网络。数据、层和训练选项均为 `trainNetwork` 函数的输入参数，如此示例中所示。

```
layers = [imageInputLayer([28 28 1])
          convolution2dLayer(5,20)
          reluLayer
          maxPooling2dLayer(2,'Stride',2)
          fullyConnectedLayer(10)
          softmaxLayer
          classificationLayer];
options = trainingOptions('adam');
convnet = trainNetwork(data,layers,options);
```

训练数据可以是数组、表或 `ImageDatastore` 对象。有关详细信息，请参阅 `trainNetwork` 函数参考页。

## 另请参阅

`trainingOptions` | `trainNetwork` | `Convolution2dLayer` | `FullyConnectedLayer`

## 详细信息

- “了解卷积神经网络”（第 1-16 页）
- “指定卷积神经网络的层”（第 1-25 页）
- “创建简单的深度学习网络以用于分类”（第 3-39 页）
- “Resume Training from Checkpoint Network”

## 长短期记忆网络

本主题说明如何使用长短期记忆 (LSTM) 网络处理分类和回归任务的序列和时序数据。有关如何使用 LSTM 网络对序列数据进行分类的示例，请参阅“使用深度学习进行序列分类”（第 4-2 页）。

LSTM 网络是一种循环神经网络 (RNN)，可以学习序列数据的时间步之间的长期依存关系。

### LSTM 网络架构

LSTM 网络的核心组件是序列输入层和 LSTM 层。序列输入层将序列或时序数据输入网络中。LSTM 层学习序列数据的时间步之间的长期相关性。

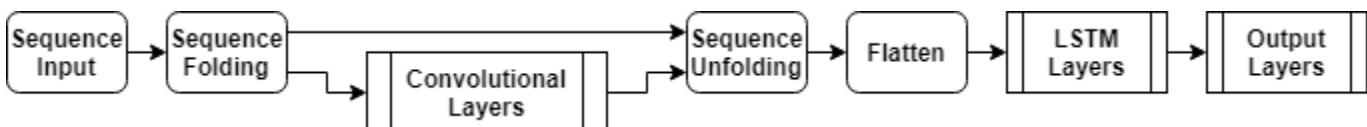
下图说明用于分类的简单 LSTM 网络的架构。该网络从一个序列输入层开始，后跟一个 LSTM 层。为了预测类标签，该网络的末尾是一个全连接层、一个 softmax 层和一个分类输出层。



下图说明用于回归的简单 LSTM 网络的架构。该网络从一个序列输入层开始，后跟一个 LSTM 层。该网络的末尾是一个全连接层和一个回归输出层。



下图说明用于视频分类的网络的架构。要将图像序列输入到网络，请使用序列输入层。要使用卷积层来提取特征，也就是说，要将卷积运算独立地应用于视频的每帧，请使用一个序列折叠层，后跟一个卷积层，然后是一个序列展开层。要使用 LSTM 层从向量序列中学习，请使用一个扁平化层，后跟 LSTM 层和输出层。



### 分类 LSTM 网络

要创建针对“序列到标签”分类的 LSTM 网络，请创建一个层数组，其中包含一个序列输入层、一个 LSTM 层、一个全连接层、一个 softmax 层和一个分类输出层。

将序列输入层的大小设置为输入数据的特征数量。将全连接层的大小设置为类的数量。您不需要指定序列长度。

对于 LSTM 层，指定隐含单元的数量和输出模式 'last'。

```

numFeatures = 12;
numHiddenUnits = 100;
numClasses = 9;
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits, 'OutputMode', 'last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
  
```

有关说明如何针对“序列到标签”分类训练 LSTM 网络和对新数据进行分类的示例，请参阅“使用深度学习进行序列分类”（第 4-2 页）。

要针对“序列到序列”分类创建一个 LSTM 网络，请使用与“序列到标签”分类相同的架构，但将 LSTM 层的输出模式设置为 'sequence'。

```
numFeatures = 12;
numHiddenUnits = 100;
numClasses = 9;
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

## 回归 LSTM 网络

要针对“序列到单个”回归创建一个 LSTM 网络，请创建一个层数组，其中包含一个序列输入层、一个 LSTM 层、一个全连接层和一个回归输出层。

将序列输入层的大小设置为输入数据的特征数量。将全连接层的大小设置为响应的数量。您不需要指定序列长度。

对于 LSTM 层，指定隐含单元的数量和输出模式 'last'。

```
numFeatures = 12;
numHiddenUnits = 125;
numResponses = 1;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

要针对“序列到序列”回归创建一个 LSTM 网络，请使用与“序列到单个”回归相同的架构，但将 LSTM 层的输出模式设置为 'sequence'。

```
numFeatures = 12;
numHiddenUnits = 125;
numResponses = 1;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

有关说明如何针对“序列到序列”回归训练 LSTM 网络和对新数据进行预测的示例，请参阅“使用深度学习进行“序列到序列”回归”（第 4-38 页）。

## 视频分类网络

要针对包含图像序列的数据（如视频数据和医学图像）创建一个深度学习网络，请使用序列输入层指定图像序列输入。

要使用卷积层来提取特征，也就是说，要将卷积运算独立地应用于视频的每帧，请使用一个序列折叠层，后跟一个卷积层，然后是一个序列展开层。要使用 LSTM 层从向量序列中学习，请使用一个扁平化层，后跟 LSTM 层和输出层。

```
inputSize = [28 28 1];
filterSize = 5;
numFilters = 20;
numHiddenUnits = 200;
numClasses = 10;

layers = [...
    sequenceInputLayer(inputSize,'Name','input')

    sequenceFoldingLayer('Name','fold')

    convolution2dLayer(filterSize,numFilters,'Name','conv')
    batchNormalizationLayer('Name','bn')
    reluLayer('Name','relu')

    sequenceUnfoldingLayer('Name','unfold')
    flattenLayer('Name','flatten')

    lstmLayer(numHiddenUnits,'OutputMode','last','Name','lstm')

    fullyConnectedLayer(numClasses, 'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','classification')];
```

将这些层转换为一个层次图，并将序列折叠层的 `miniBatchSize` 输出连接到序列展开层的对应输入。

```
lgraph = layerGraph(layers);
lgraph = connectLayers(lgraph,'fold/miniBatchSize','unfold/miniBatchSize');
```

有关说明如何针对视频分类训练深度学习网络的示例，请参阅“使用深度学习对视频进行分类”（第 4-45 页）。

## 更深的 LSTM 网络

您可以通过在 LSTM 层之前插入具有输出模式 '`sequence`' 的额外 LSTM 层来加大 LSTM 网络的深度。为了防止过拟合，可以在 LSTM 层后插入丢弃层。

对于“序列到标签”分类网络，最后一个 LSTM 层的输出模式必须为 '`last`'。

```
numFeatures = 12;
numHiddenUnits1 = 125;
numHiddenUnits2 = 100;
numClasses = 9;
layers = [...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits1,'OutputMode','sequence')
    dropoutLayer(0.2)
    lstmLayer(numHiddenUnits2,'OutputMode','last')
    dropoutLayer(0.2)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

对于“序列到序列”分类网络，最后一个 LSTM 层的输出模式必须为 'sequence'。

```
numFeatures = 12;
numHiddenUnits1 = 125;
numHiddenUnits2 = 100;
numClasses = 9;
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits1,'OutputMode','sequence')
    dropoutLayer(0.2)
    lstmLayer(numHiddenUnits2,'OutputMode','sequence')
    dropoutLayer(0.2)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

## 网络层

层	说明
 sequenceInputLayer	序列输入层向网络输入序列数据。
 lstmLayer	LSTM 层学习时序和序列数据中时间步之间的长期相关性。
 bilstmLayer	双向 LSTM (BiLSTM) 层学习时序或序列数据的时间步之间的双向长期相关性。当您希望网络在每个时间步从完整时序中学习时，这些相关性会很有用。
 gruLayer	GRU 层学习时序和序列数据中时间步之间的相关性。
 sequenceFoldingLayer	序列折叠层将一批图像序列转换为一批图像。使用序列折叠层独立地对图像序列的时间步执行卷积运算。
 sequenceUnfoldingLayer	序列展开层在序列折叠后还原输入数据的序列结构。
 flattenLayer	扁平化层将输入的空间维度折叠成通道维度。
 wordEmbeddingLayer	单词嵌入层将单词索引映射到向量。

## 分类、预测和预报

要对新数据进行分类或预测，请使用 `classify` 和 `predict`。

LSTM 网络可以记住各次预测之间的网络状态。如果您事先没有完整的时序，或您要对一个长时序进行多次预测，则网络状态会很有用。

要对一个时序的多个部分进行预测和分类并更新网络状态，请使用 `predictAndUpdateState` 和 `classifyAndUpdateState`。要在各次预测之间重置网络状态，请使用 `resetState`。

有关如何预测序列的将来时间步的示例，请参阅“使用深度学习进行时序预测”（第 4-9 页）。

## 序列填充、截断和拆分

LSTM 网络支持具有不同序列长度的输入数据。当使数据通过网络时，软件会填充、截断或拆分序列，以便每个小批量中的所有序列都具有指定的长度。您可以使用 `trainingOptions` 中的 `SequenceLength` 和 `SequencePaddingValue` 名称-值对组参数来指定序列长度和用于填充序列的值。

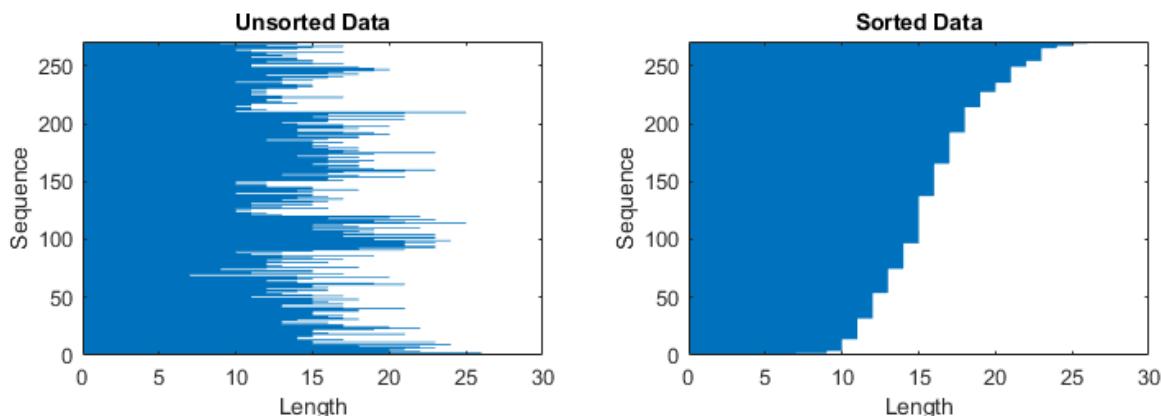
训练网络后，在使用 `classify`、`predict`、`classifyAndUpdateState`、`predictAndUpdateState` 和 `activations` 函数时，使用相同的小批量大小和填充选项。

### 按长度对序列排序

要在填充或截断序列时减少填充或丢弃的数据量，请尝试按序列长度对数据进行排序。要按序列长度对数据进行排序，首先使用 `cellfun` 对每个序列应用 `size(X,2)` 来获得每个序列的列数。然后使用 `sort` 对序列长度进行排序，并使用第二个输出对原始序列重新排序。

```
sequenceLengths = cellfun(@(X) size(X,2), XTrain);
[sequenceLengthsSorted, idx] = sort(sequenceLengths);
XTrain = XTrain(idx);
```

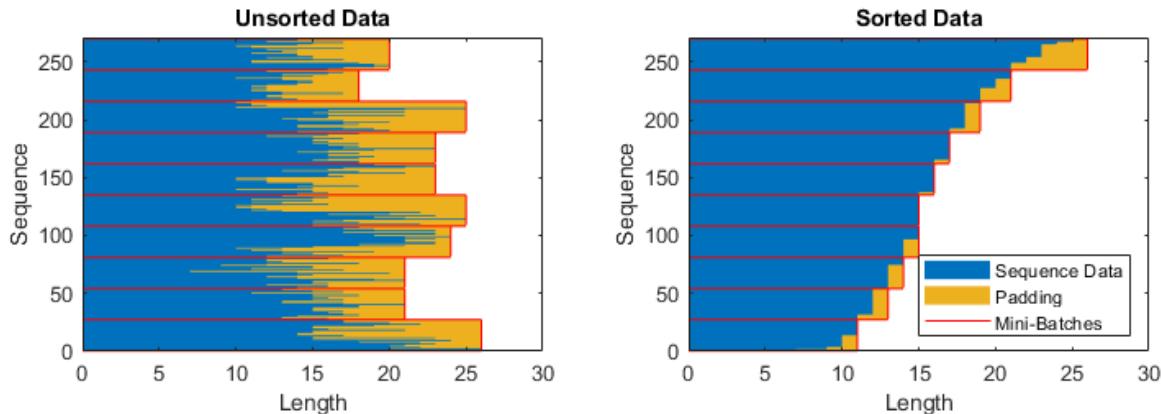
以下各图中以条形图显示了已排序和未排序数据的序列长度。



### 填充序列

如果您指定序列长度 '`longest`'，则软件会填充序列，使小批量中的所有序列具有与小批量中的最长序列相同的长度。此选项是默认选项。

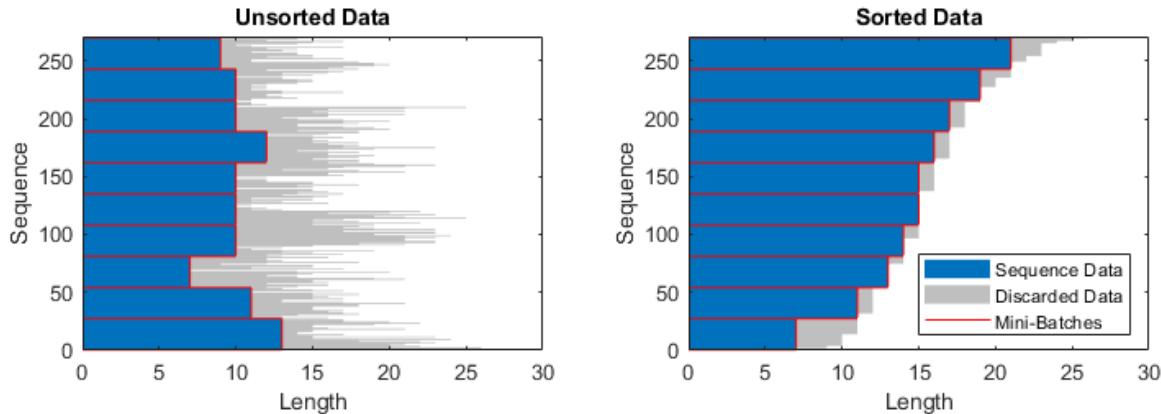
下图说明将 '`SequenceLength`' 设置为 '`longest`' 的效果。



### 截断序列

如果您指定序列长度 'shortest'，则软件会截断序列，使小批量中的所有序列具有与该小批量中的最短序列相同的长度。序列中的其余数据被丢弃。

下图说明将 'SequenceLength' 设置为 'shortest' 的效果。



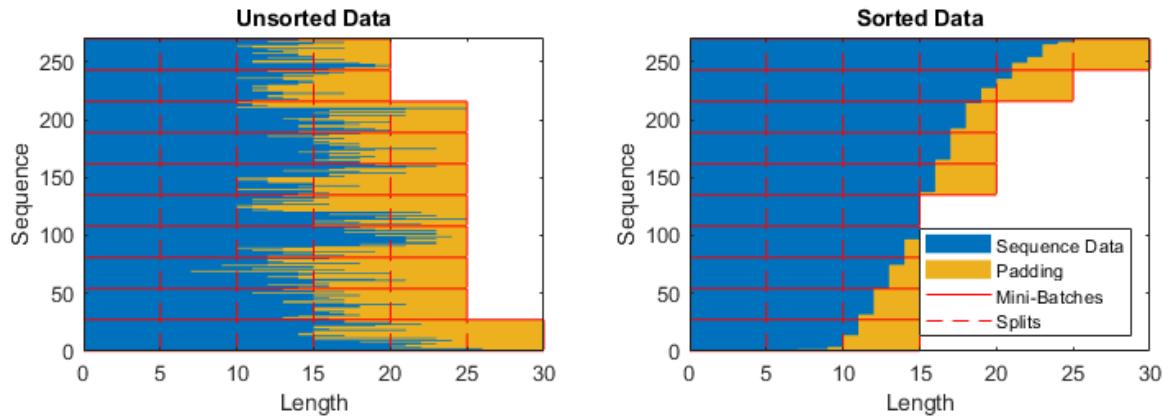
### 拆分序列

如果将序列长度设置为整数值，则软件会将小批量中的所有序列填充为大于该小批量中最长序列长度的指定长度的最接近倍数。然后，软件将每个序列拆分为指定长度的较小序列。如果发生拆分，则软件会创建额外的小批量。

如果整个序列无法放入内存中，请使用此选项。您也可以尝试通过将 `trainingOptions` 中的 '`MiniBatchSize`' 选项设置为较低的值来减少每个小批量的序列数。

如果您将序列长度指定为正整数，则软件会在连续的迭代中处理较小的序列。网络会在拆分的序列之间更新网络状态。

下图说明将 'SequenceLength' 设置为 5 的效果。



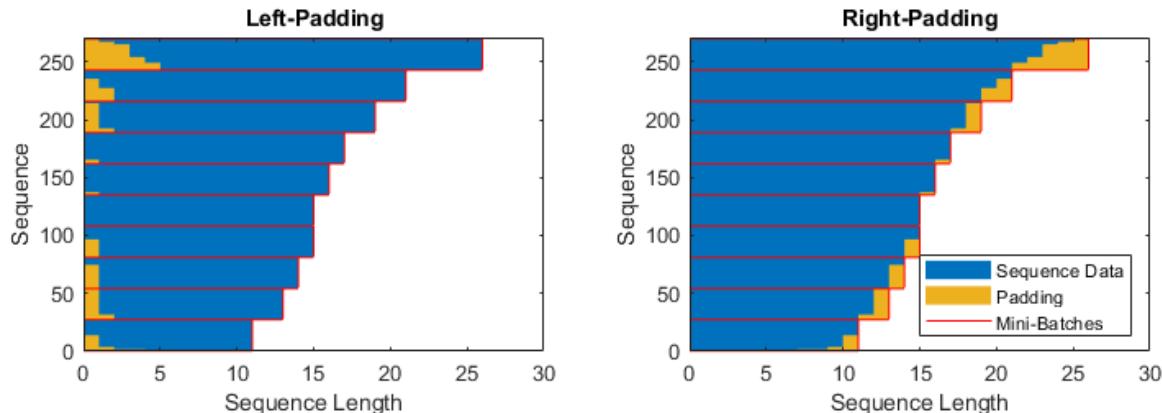
### 指定填充方向

填充和截断的位置会影响训练、分类和预测准确度。请尝试将 `trainingOptions` 中的 `'SequencePaddingDirection'` 选项设置为 `'left'` 或 `'right'`，看看哪个最适合您的数据。

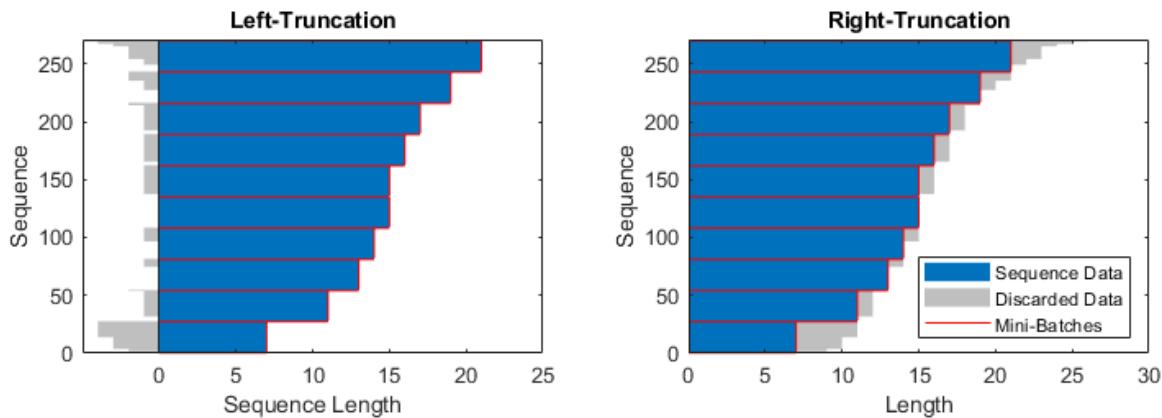
由于 LSTM 层一次处理一个时间步的序列数据，当层 `OutputMode` 属性为 `'last'` 时，最终时间步中的任何填充都会对层输出产生负面影响。要填充或截断左侧的序列数据，请将 `'SequencePaddingDirection'` 选项设置为 `'left'`。

对于“序列到序列”网络（当每个 LSTM 层的 `OutputMode` 属性为 `'sequence'` 时），前几个时间步中的任何填充都会对较早时间步的预测产生负面影响。要填充或截断右侧的序列数据，请将 `'SequencePaddingDirection'` 选项设置为 `'right'`。

下图说明左侧和右侧的填充序列数据。



下图说明左侧和右侧的截断序列数据。



## 归一化序列数据

要在训练时使用以零为中心的归一化自动对训练数据调整中心位置，请将 `sequenceInputLayer` 的 `Normalization` 选项设置为 '`'zerocenter'`'。您也可以通过首先计算所有序列的每个特征的均值和标准差来归一化序列数据。然后，对于每个训练观测值，减去均值并除以标准差。

```
mu = mean([XTrain{:}],2);
sigma = std([XTrain{:}],0,2);
XTrain = cellfun(@(X) (X-mu)./sigma,XTrain,'UniformOutput',false);
```

## 无法放入内存的数据

如果数据太大而无法放入内存或在读取批量数据时无法执行特定操作，请对序列、时序和信号数据使用数据存储。

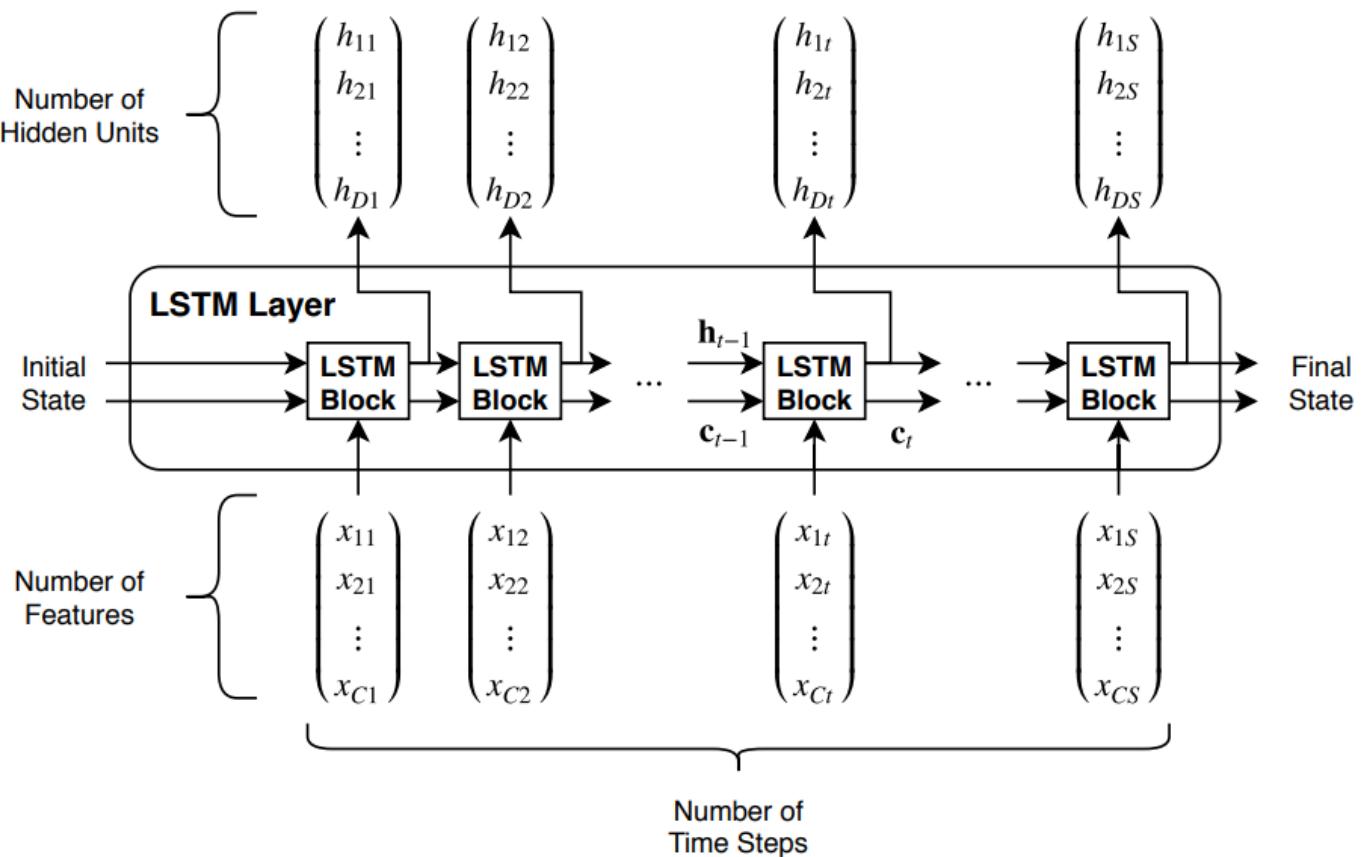
要了解详细信息，请参阅“使用无法放入内存的序列数据训练网络”（第 18-14 页）和“使用深度学习对无法放入内存的文本数据进行分类”（第 18-21 页）。

## 可视化

通过使用 `activations` 函数提取激活，调查并可视化 LSTM 网络从序列和时序数据中学习到的特征。要了解详细信息，请参阅“Visualize Activations of LSTM Network”。

## LSTM 层架构

下图说明具有 C 个长度为 S 的特征（通道）的时序 X 通过 LSTM 层的流程。在图中， $\mathbf{h}_t$  和  $\mathbf{c}_t$  分别表示在时间步 t 的输出（也称为隐藏状态）和单元状态。



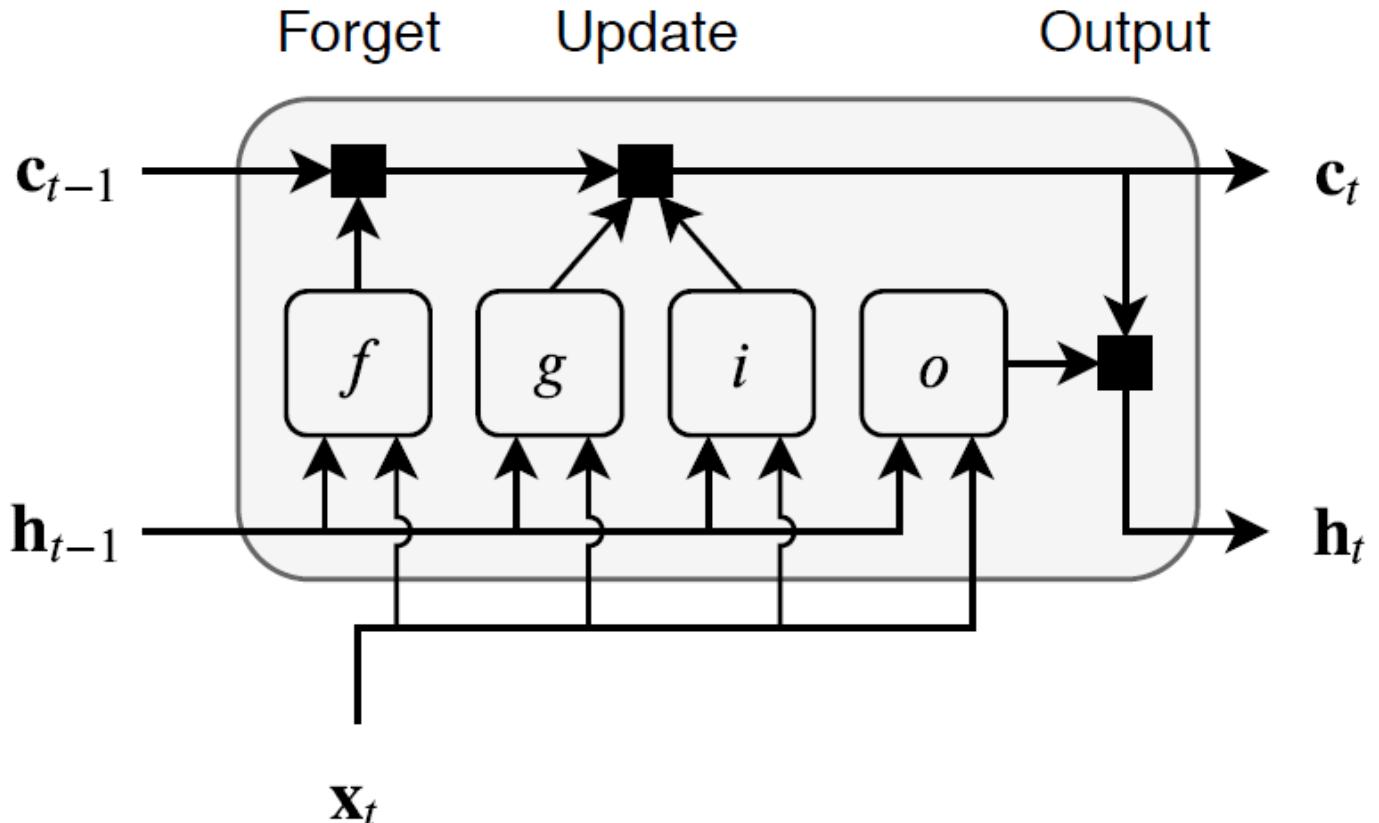
第一个 LSTM 模块使用网络的初始状态和序列的第一个时间步来计算第一个输出和更新后的单元状态。在时间步  $t$  上，该模块使用网络的当前状态  $(\mathbf{c}_{t-1}, \mathbf{h}_{t-1})$  和序列的下一个时间步来计算输出和更新后的单元状态  $\mathbf{c}_t$ 。

该层的状态由隐藏状态（也称为输出状态）和单元状态组成。时间步  $t$  处的隐藏状态包含该时间步的 LSTM 层的输出。单元状态包含从前面的时间步中获得的信息。在每个时间步，该层都会在单元状态中添加或删除信息。该层使用不同的门控制这些更新。

以下组件控制层的单元状态和隐藏状态。

组件	目的
输入门 (i)	控制单元状态更新的级别
遗忘门 (f)	控制单元状态重置（遗忘）的级别
候选单元 (g)	向单元状态添加信息
输出门 (o)	控制添加到隐藏状态的单元状态的级别

下图说明在时间步  $t$  上的数据流。该图突出显示门如何遗忘、更新和输出单元状态和隐藏状态。



LSTM 层的可学习权重包括输入权重 W (InputWeights)、循环权重 R (RecurrentWeights) 以及偏置 b (Bias)。矩阵 W、R 和 b 分别是输入权重、循环权重和每个分量的偏置的串联。这些矩阵的串联如下：

$$W = \begin{bmatrix} W_i \\ W_f \\ W_g \\ W_o \end{bmatrix}, R = \begin{bmatrix} R_i \\ R_f \\ R_g \\ R_o \end{bmatrix}, b = \begin{bmatrix} b_i \\ b_f \\ b_g \\ b_o \end{bmatrix}$$

其中 i、f、g、o 分别表示输入门、遗忘门、候选单元和输出门。

时间步 t 处的单元状态由下式给出：

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot g_t,$$

其中  $\odot$  表示 Hadamard 乘积（向量的按元素乘法）。

时间步 t 处的隐藏状态由下式给出：

$$\mathbf{h}_t = o_t \odot \sigma_c(\mathbf{c}_t),$$

其中  $\sigma_c$  表示状态激活函数。默认情况下，lstmLayer 函数使用双曲正切函数 (tanh) 计算状态激活函数。

以下公式说明时间步 t 处的组件。

组件	公式
输入门	$i_t = \sigma_g(W_i \mathbf{x}_t + R_i \mathbf{h}_{t-1} + b_i)$
遗忘门	$f_t = \sigma_g(W_f \mathbf{x}_t + R_f \mathbf{h}_{t-1} + b_f)$
候选单元	$g_t = \sigma_c(W_g \mathbf{x}_t + R_g \mathbf{h}_{t-1} + b_g)$
输出门	$o_t = \sigma_g(W_o \mathbf{x}_t + R_o \mathbf{h}_{t-1} + b_o)$

在这些计算中， $\sigma_g$  表示门激活函数。默认情况下，`lstmLayer` 函数使用  $\sigma(x) = (1 + e^{-x})^{-1}$  给出的 sigmoid 函数来计算门激活函数。

## 参考

[1] Hochreiter, S., and J. Schmidhuber. "Long short-term memory." *Neural computation*. Vol. 9, Number 8, 1997, pp.1735–1780.

## 另请参阅

`sequenceInputLayer` | `lstmLayer` | `bilstmLayer` | `gruLayer` | `classifyAndUpdateState` | `predictAndUpdateState` | `resetState` | `sequenceFoldingLayer` | `sequenceUnfoldingLayer` | `flattenLayer` | `wordEmbeddingLayer` | `activations`

## 相关示例

- “使用深度学习进行序列分类”（第 4-2 页）
- “使用深度学习进行时序预测”（第 4-9 页）
- “使用深度学习进行“序列到序列”分类”（第 4-33 页）
- “使用深度学习进行“序列到序列”回归”（第 4-38 页）
- “使用深度学习对视频进行分类”（第 4-45 页）
- “Visualize Activations of LSTM Network”
- “Develop Custom Mini-Batch Datastore”
- “在 MATLAB 中进行深度学习”（第 1-2 页）

# 深度网络设计器

---

- “使用深度网络设计器进行迁移学习” (第 2-2 页)
- “使用深度网络设计器构建网络” (第 2-15 页)
- “使用深度网络设计器创建简单的序列分类网络” (第 2-21 页)

## 使用深度网络设计器进行迁移学习

此示例说明如何使用深度网络设计器以交互方式执行迁移学习。

迁移学习指采用预训练的深度学习网络并对其进行微调以学习新任务的过程。使用迁移学习通常比从头开始训练网络更快、更简单。您可以使用更少的数据量将学习到的特征快速迁移到新任务中。

按照以下步骤进行操作，使用深度网络设计器执行迁移学习以进行图像分类：

- 1 打开深度网络设计器，选择一个预训练网络。
- 2 导入新数据集。
- 3 用适应新数据集的新层替换最终层。
- 4 设置学习率，使新层中的学习速度快于迁移的层。
- 5 使用深度网络设计器训练网络，或通过命令行导出网络进行训练。

### 提取数据

在工作区中，提取 MathWorks Merch 数据集。这是包含 75 幅 MathWorks 商品图像的小型数据集，这些商品分属五个不同类（瓶盖、魔方、扑克牌、螺丝刀和手电筒）。

```
unzip("MerchData.zip");
```

### 选择预训练网络

要打开深度网络设计器，请在 **App** 选项卡上的**机器学习和深度学习**下，点击该 App 的图标。您也可以从命令行打开该 App：

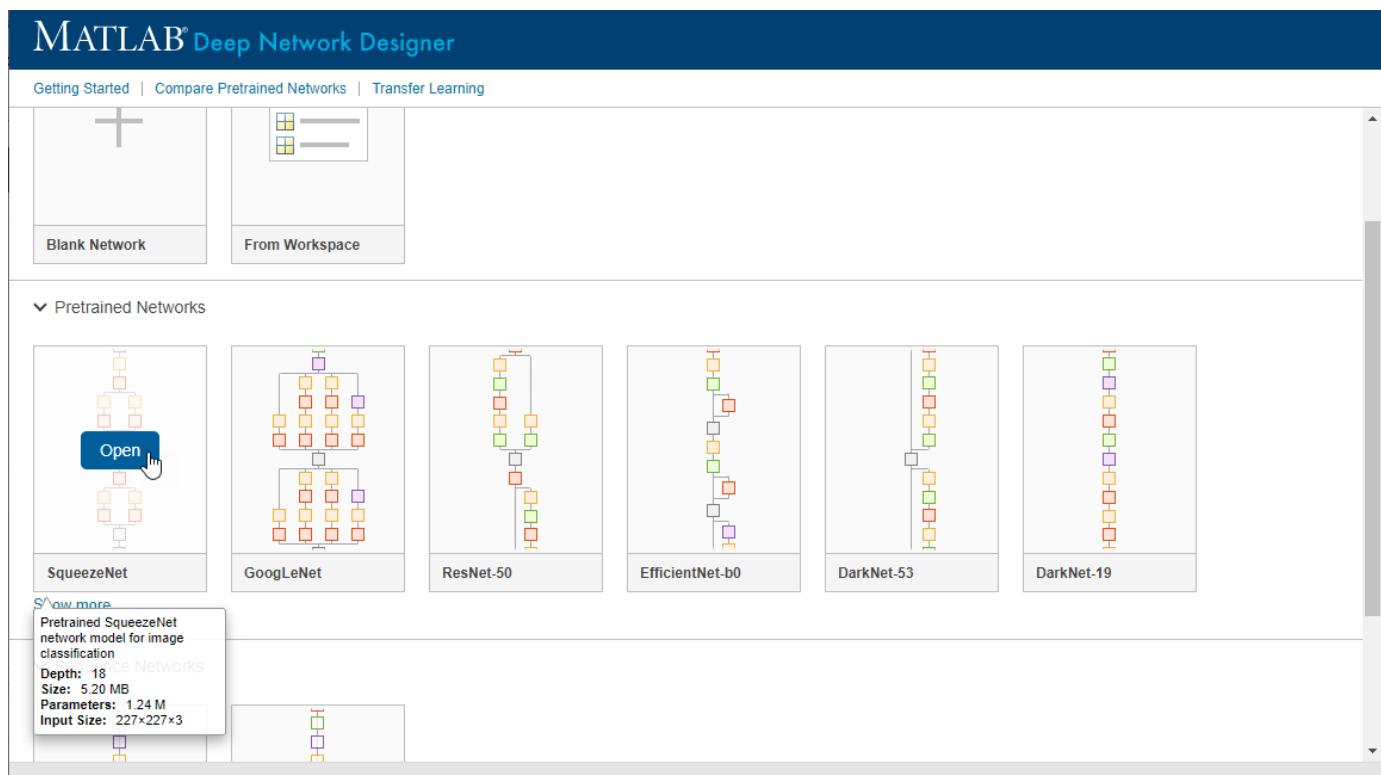
```
deepNetworkDesigner
```

深度网络设计器提供一些精选的预训练图像分类网络，这些网络已学习适用于各种图像的丰富特征表示。如果您的图像与最初用于训练网络的图像相似，迁移学习效果最好。如果您的训练图像是像 ImageNet 数据库中那样的自然图像，则任一预训练网络都合适。有关可用网络的列表以及如何比较它们，请参阅“[预训练的深度神经网络](#)”（第 1-8 页）。

如果您的数据与 ImageNet 数据相差很大（例如，如果您有很小的图像、频谱图或非图像数据），训练新网络可能效果更好。有关如何从头开始训练网络的示例，请参阅“[使用深度网络设计器创建简单的序列分类网络](#)”（第 2-21 页）和“[Train Simple Semantic Segmentation Network in Deep Network Designer](#)”。

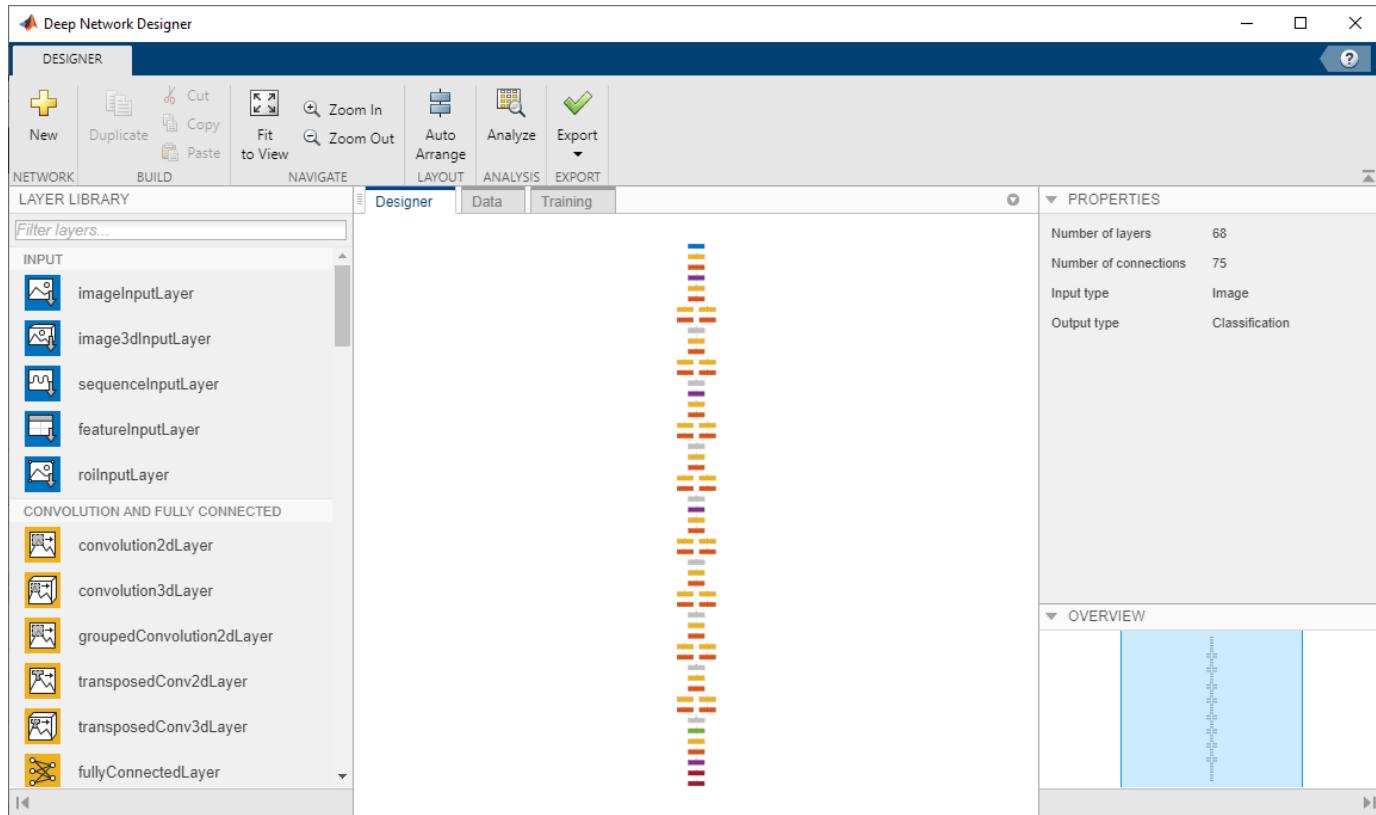
SqueezeNet 不需要额外的支持包。对于其他预训练网络，如果您没有安装所需的支持包，则该 App 提供[安装](#)选项。

从预训练网络列表中选择 **SqueezeNet**，然后点击[打开](#)。



## 浏览网络

深度网络设计器在设计器窗格中显示整个网络的缩小视图。

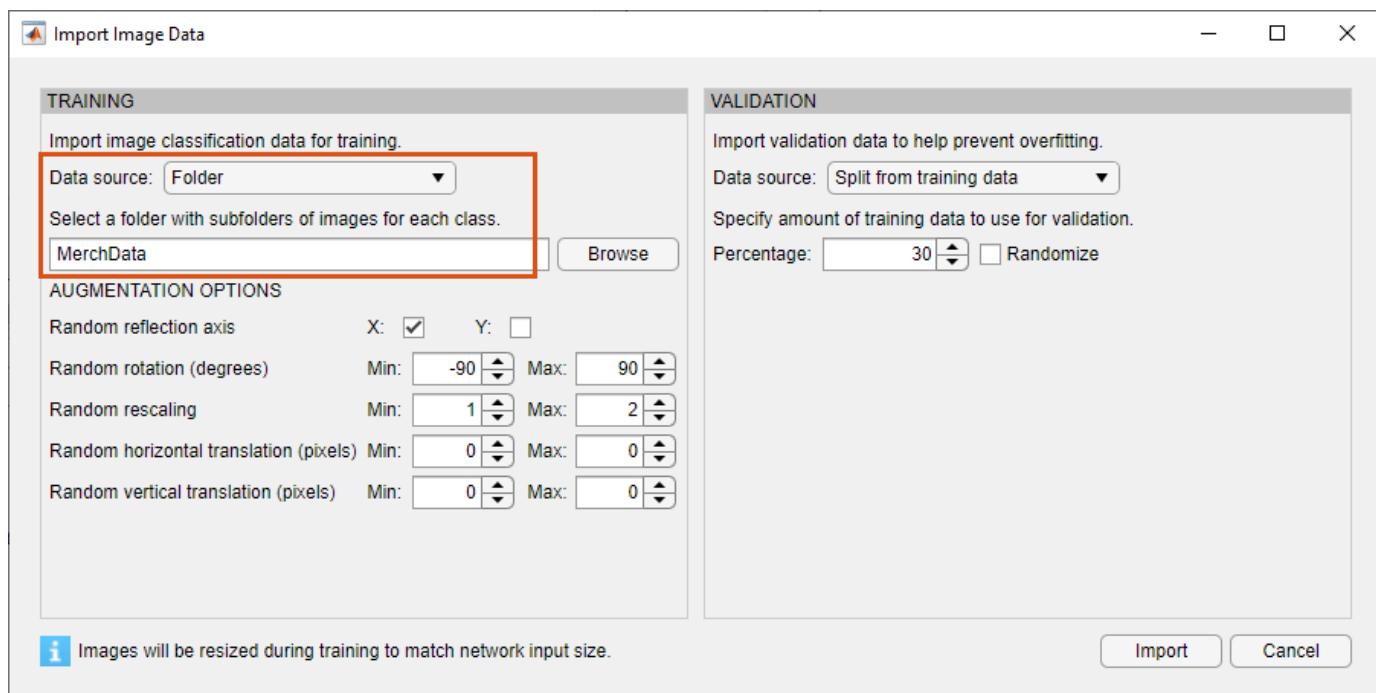


浏览网络图。要使用鼠标放大，请使用 **Ctrl + 滚轮**。要平移，请使用箭头键，或按住滚轮并拖动鼠标。选择一个层以查看其属性。取消选择所有层，以在属性窗格中查看网络摘要。

### 导入数据

要将数据加载到深度网络设计器中，请在**数据**选项卡上，点击**导入数据 > 导入图像数据**。将打开“**导入图像数据**”对话框。

在**数据源**列表中，选择**文件夹**。点击**浏览**并选择提取的 MerchData 文件夹。



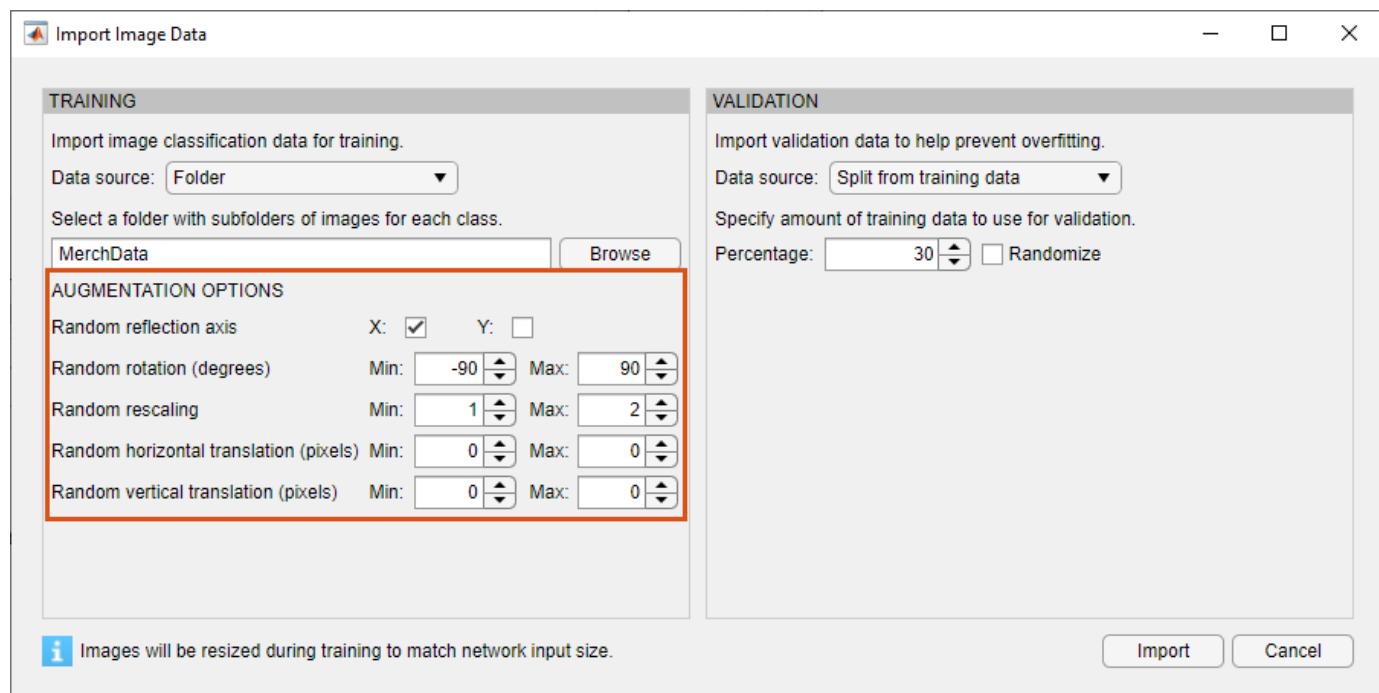
## 图像增强

您可以选择将图像增强应用于您的训练数据。深度网络设计器提供以下增强选项：

- x 轴上的随机翻转
- y 轴上的随机翻转
- 随机旋转
- 随机重新缩放
- 随机水平平移
- 随机垂直平移

通过对数据应用随机增强，您实际上是增加了训练数据量。增强还使您能够训练网络，使其在图像数据失真时仍保持稳定。例如，您可以对输入图像添加随机旋转，使网络不会因输入图像中存在旋转而改变。

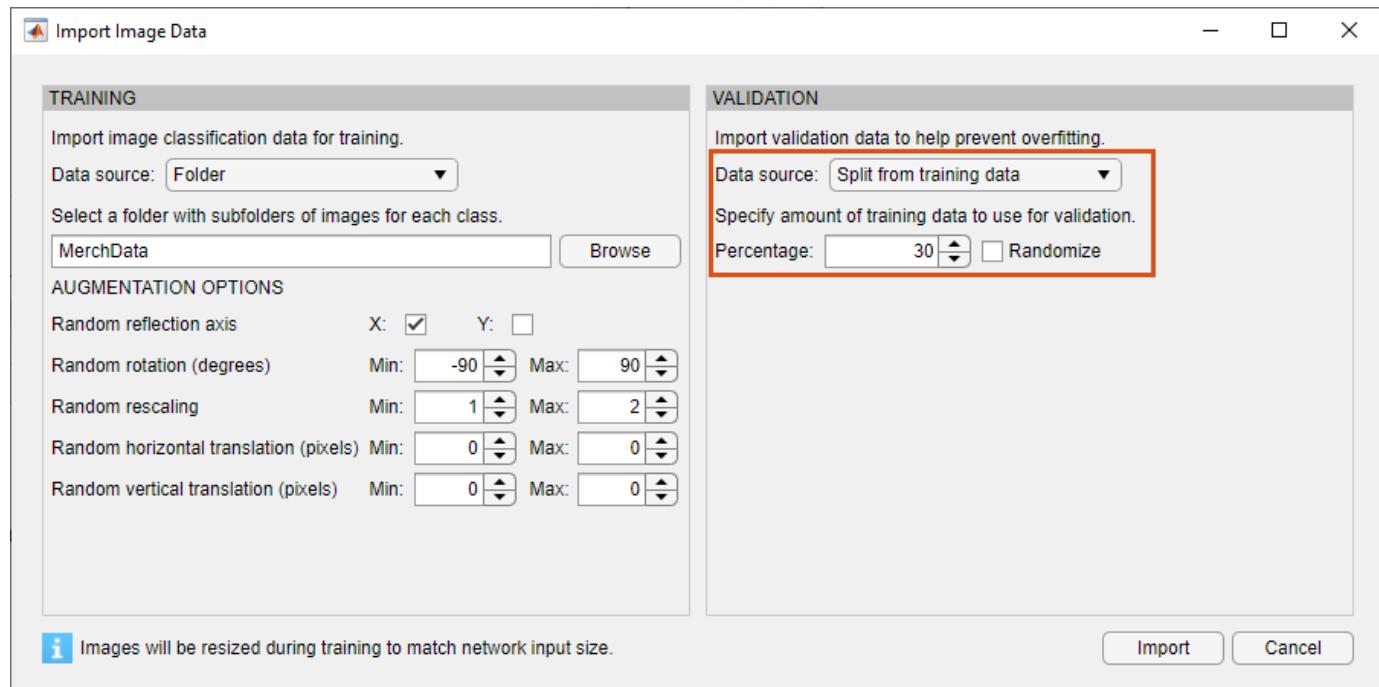
对于此示例，在 x 轴上进行随机翻转，在 [-90,90] 度范围内进行随机旋转，在 [1,2] 范围内进行随机重新缩放。



## 验证数据

您还可以选择导入验证数据，您可以从训练数据中拆分出验证数据，也可以从其他来源导入。验证会估计新数据与训练数据相比的模型性能，并帮助您监控性能和防止过拟合。

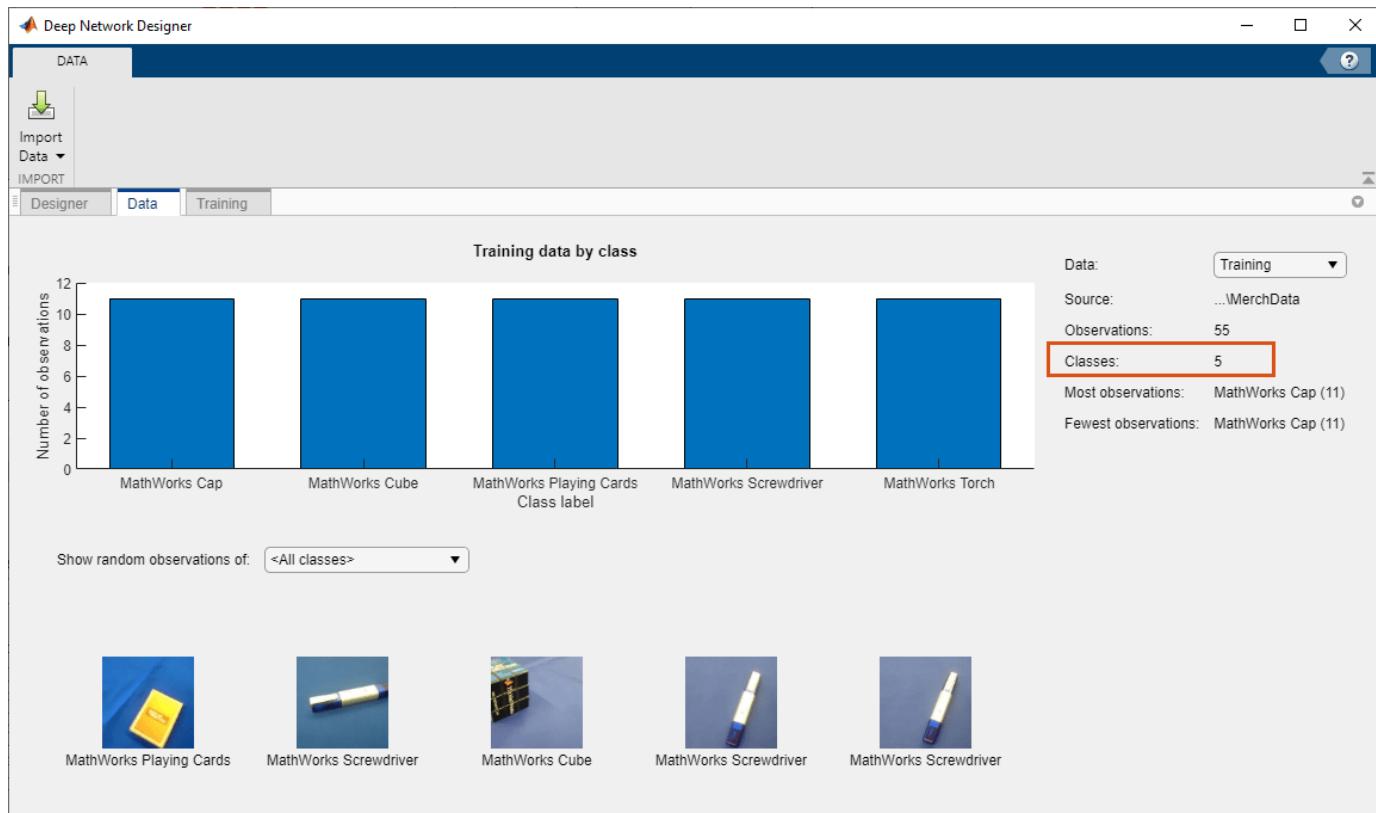
对于此示例，我们将 30% 的图像用于验证。



**点击导入**将数据导入深度网络设计器。

## 可视化数据

使用深度网络设计器，您可以在**数据**选项卡中直观地查看训练和验证数据的分布情况。您可以看到在此示例中数据集中有五个类。您也可以看到每个类的随机观测值。



## 准备要训练的网络

在设计器窗格中编辑网络，以在数据中指定新的类数量。要为迁移学习准备网络，请替换最后一个可学习层和最终分类层。

### 替换最后一个可学习层

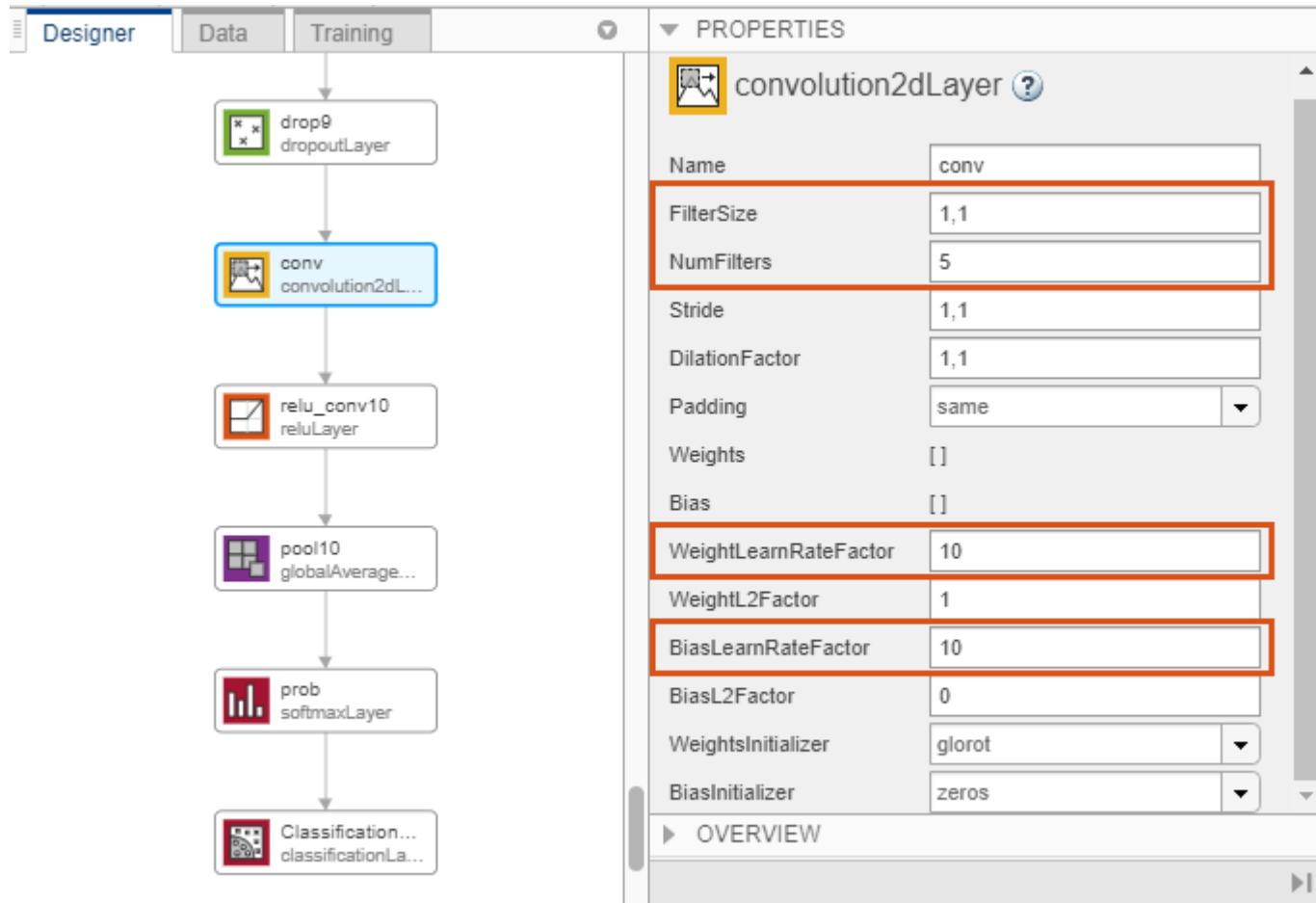
要使用预训练网络进行迁移学习，您必须更改类的数量以匹配新数据集。首先，找到网络中的最后一个可学习层。对于 SqueezeNet，最后一个可学习层是最后一个卷积层，'conv10'。在这种情况下，请将该卷积层替换为新的卷积层，其中滤波器的数量等于类的数量。

将一个新 **convolution2dLayer** 拖到画布上。要匹配原始卷积层，请将 **FilterSize** 设置为 **1,1**。

**NumFilters** 属性定义用于分类问题的类的数量。将 **NumFilters** 更改为新数据中的类数量，此示例中为 **5**。

通过将 **WeightLearnRateFactor** 和 **BiasLearnRateFactor** 设置为 **10** 来更改学习率，使新层中的学习速度快于迁移层的学习速度。

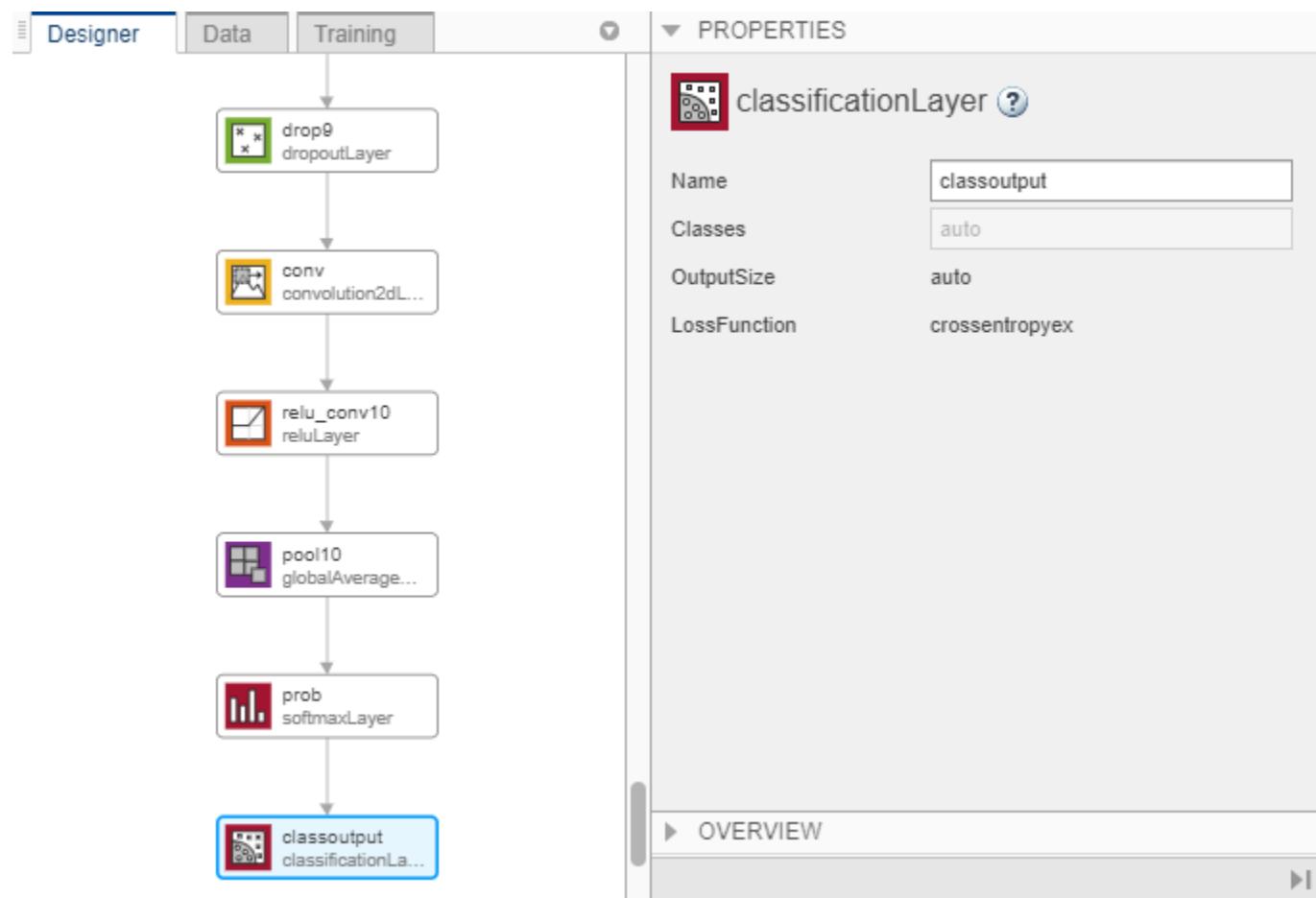
删除最后一个二维卷积层，改为连接新层。



### 替换输出层

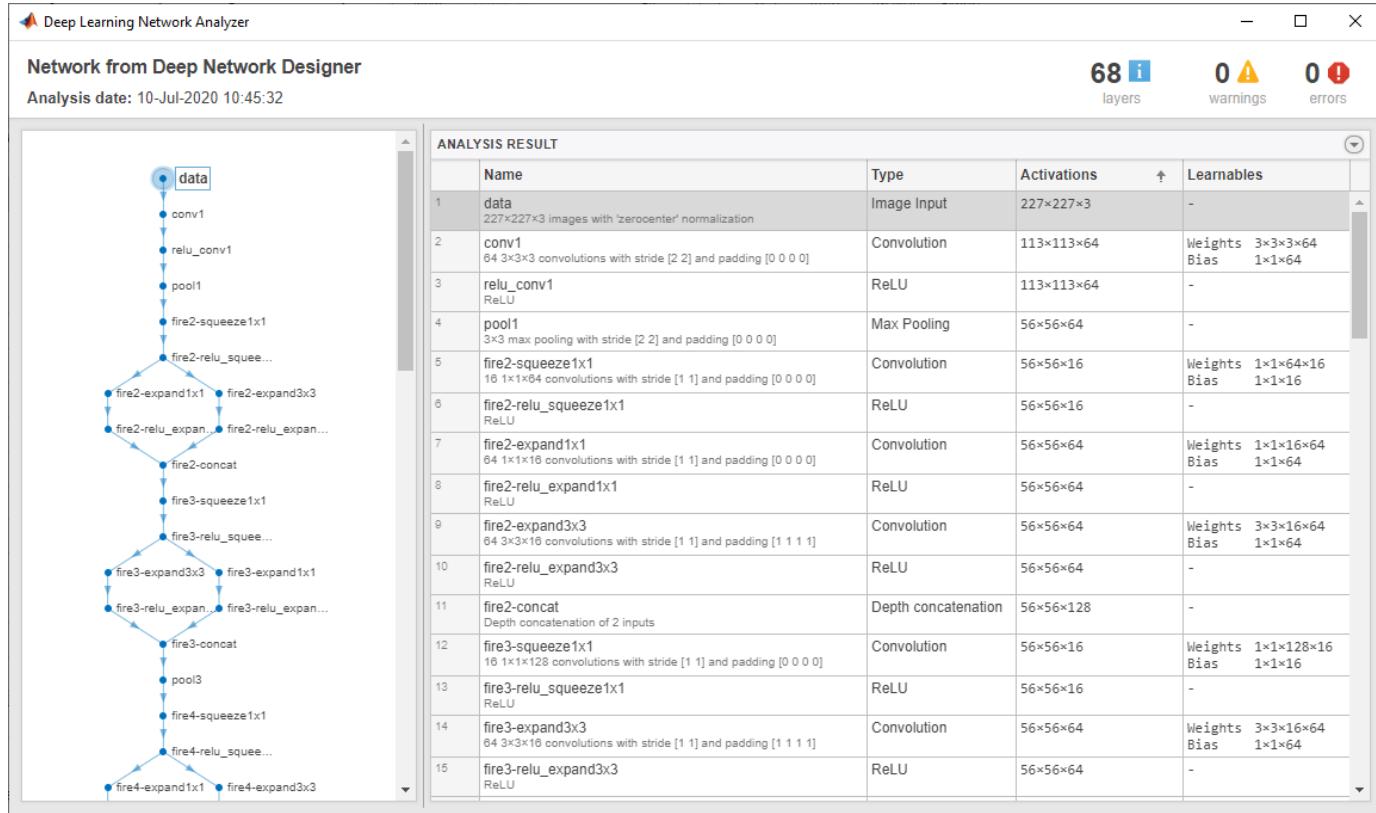
要进行迁移学习，您需要替换输出层。滚动到[网络层库](#)的末尾，将一个新的 `classificationLayer` 拖到画布上。删除原始分类层，并在其位置连接新层。

对于新输出层，您不需要设置 `OutputSize`。在训练时，深度网络设计器会根据数据自动设置层的输出类。



### 检查网络

要检查网络是否准备好进行训练，请点击**分析**。如果 Deep Learning Network Analyzer 报告零错误，则表示编辑过的网络已准备就绪可以开始训练。



## 训练网络

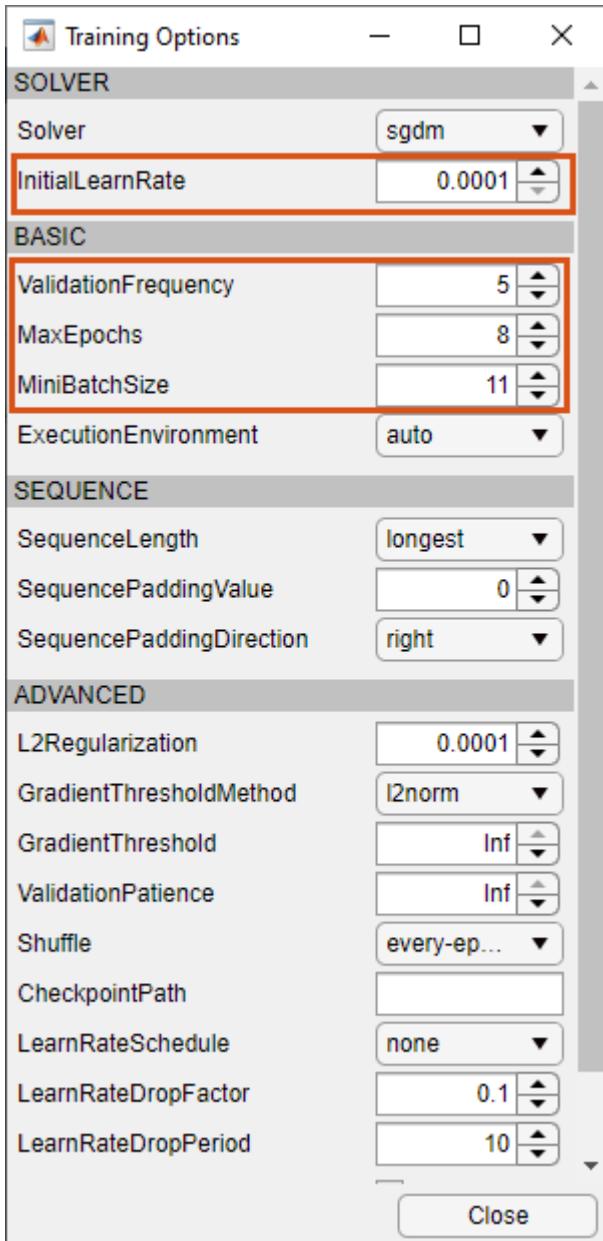
在深度网络设计器中，您可以训练在该 App 中导入或创建的网络。

要使用默认设置训练网络，请在**训练**选项卡上，点击**训练**。默认训练选项更适合大型数据集，对于小型数据集，请减少小批量大小和验证频率。

如果您要更好地控制训练，请点击**训练选项**，然后选择训练所用的设置。

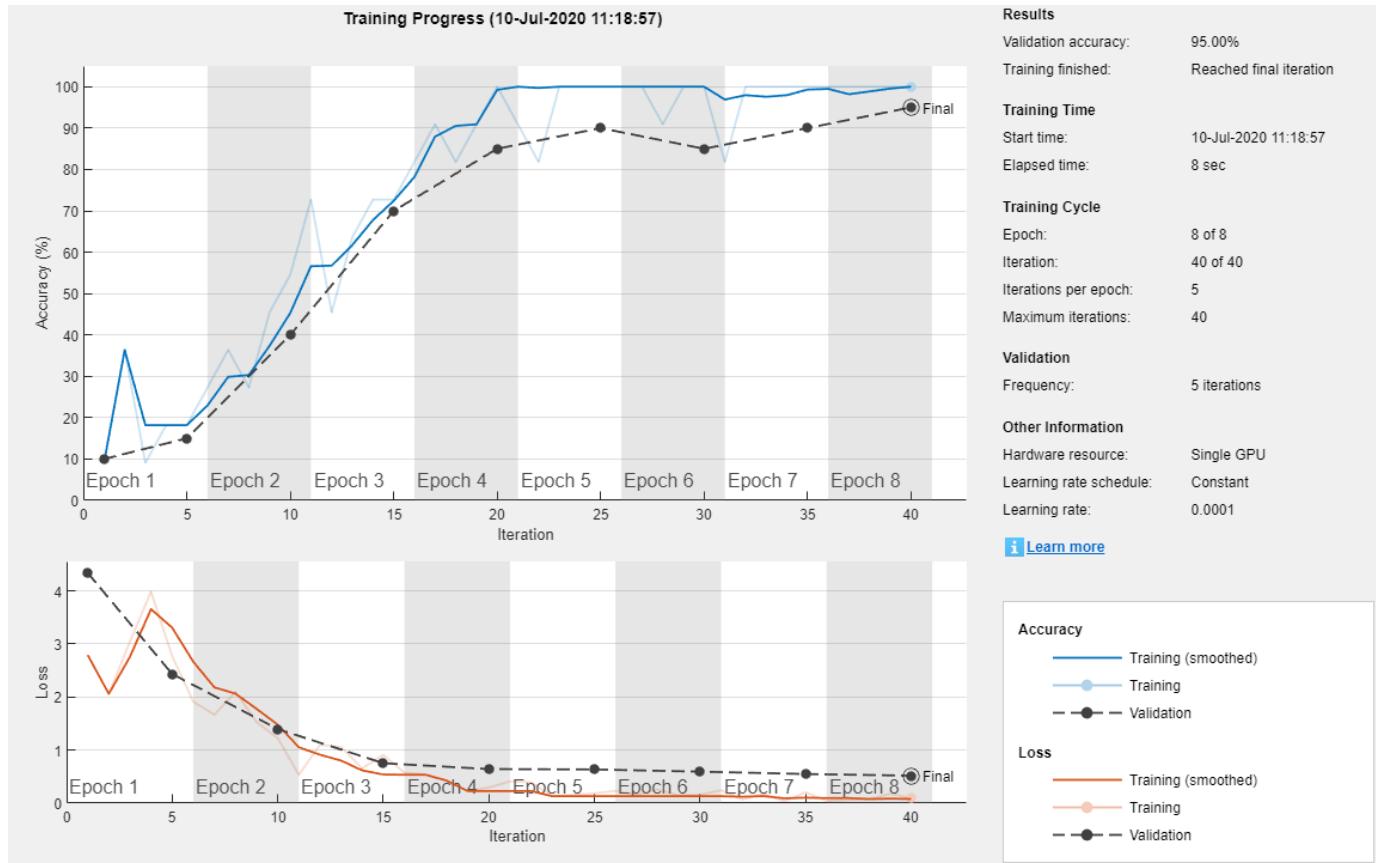
- 将初始学习率设置为较小的值以减慢迁移的层中的学习速度。
- 指定验证频率，以便每经过一轮训练就计算一次基于验证数据的准确度。
- 指定少量轮数。一轮训练是对整个训练数据集的一个完整训练周期。对于迁移学习，所需的训练轮数相对较少。
- 指定小批量大小，即每次迭代中使用多少个图像。为了确保在每轮训练中都使用整个数据集，请设置小批量大小以均分训练样本的数量。

对于此示例，将 **InitialLearnRate** 设置为 0.0001，**ValidationFrequency** 设置为 5，**MaxEpochs** 设置为 8。由于有 55 个观测值，请将 **MiniBatchSize** 设置为 11 以均分训练数据，并确保您在每轮训练期间都使用整个训练数据集。有关选择训练选项的详细信息，请参阅 **trainingOptions**。



要使用指定的训练选项训练网络，请点击**关闭**，然后点击**训练**。

深度网络设计器允许您可视化和监控训练进度。然后，如果需要，您可以编辑训练选项并重新训练网络。



## 导出结果并生成 MATLAB 代码

要导出具有训练权重的网络架构，请在**训练**选项卡上，选择**导出 > 导出经过训练的网络和结果**。深度网络设计器将经过训练的网络导出为变量 `trainedNetwork_1`，将训练信息导出为变量 `trainInfoStruct_1`。

### trainInfoStruct\_1

`trainInfoStruct_1 = struct with fields:`

```

    TrainingLoss: [1×40 double]
    TrainingAccuracy: [1×40 double]
    ValidationLoss: [4.3374 NaN NaN NaN 2.4329 NaN NaN NaN NaN 1.3966 NaN NaN NaN NaN 0.7526 NaN NaN
    ValidationAccuracy: [10 NaN NaN NaN 15 NaN NaN NaN 40 NaN NaN NaN 70 NaN NaN NaN NaN
    BaseLearnRate: [1×40 double]
    FinalValidationLoss: 0.5179
    FinalValidationAccuracy: 95
  
```

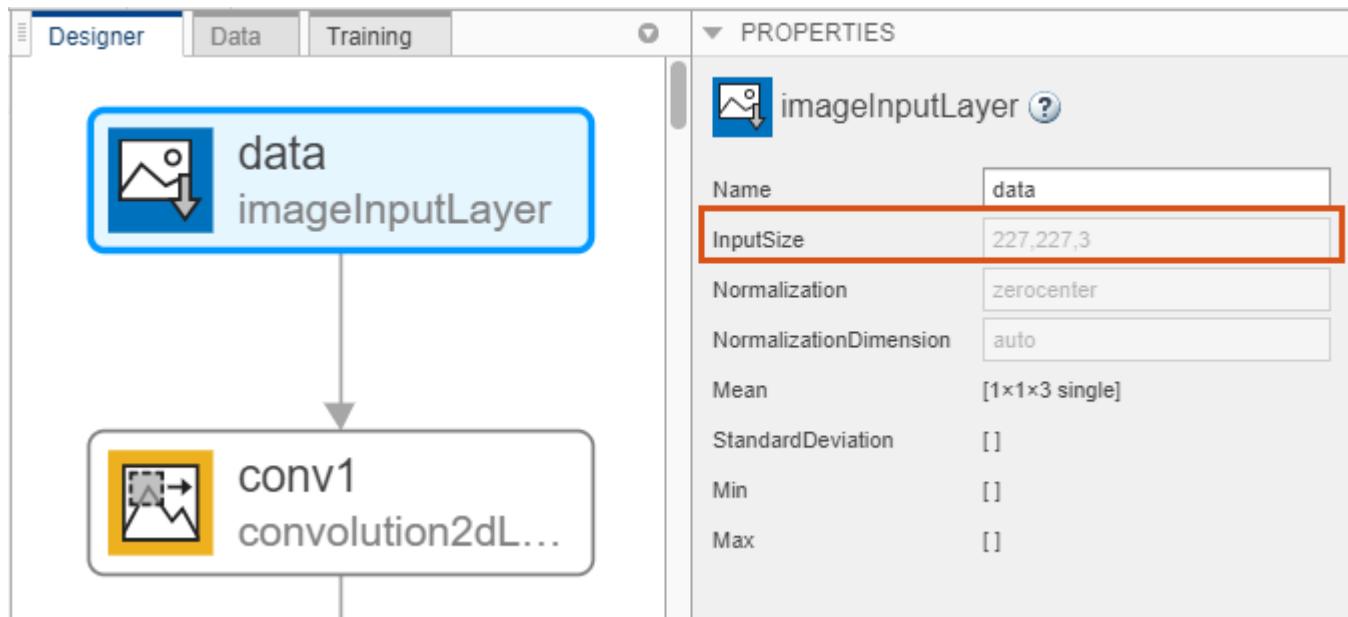
您也可以生成 MATLAB 代码，它可以重新创建所使用的网络和训练选项。在**训练**选项卡上，选择**导出 > 生成训练代码**。查看 MATLAB 代码，了解如何以编程方式准备训练数据、创建网络架构和训练网络。

## 对新图像进行分类

加载一个新图像以使用经过训练的网络对其进行分类。

```
I = imread("MerchDataTest.jpg");
```

深度网络设计器会在训练期间调整图像大小，以匹配网络输入大小。要查看网络输入大小，请转至设计器窗格，然后选择 **imageInputLayer**（第一层）。该网络的输入大小为  $227 \times 227$ 。



调整测试图像的大小以匹配网络输入大小。

```
I = imresize(I, [227 227]);
```

使用经过训练的网络对测试图像进行分类。

```
[YPred,probs] = classify(trainedNetwork_1,I);
imshow(I)
label = YPred;
title(string(label) + "," + num2str(100*max(probs),3) + "%");
```



**另请参阅**  
深度网络设计器

### 相关示例

- “使用深度网络设计器构建网络” (第 2-15 页)
- “Import Data into Deep Network Designer”
- “Generate MATLAB Code from Deep Network Designer”
- “Deep Learning Tips and Tricks”
- “深度学习层列表” (第 1-18 页)

# 使用深度网络设计器构建网络

使用深度网络设计器以交互方式构建和编辑深度学习网络。使用此 App，您可以：

- 导入和编辑网络。
- 从头开始构建新网络。
- 添加新层并创建新连接。
- 查看和编辑层属性。
- 组合网络。
- 导入自定义层。
- 生成 MATLAB 代码来创建网络架构。

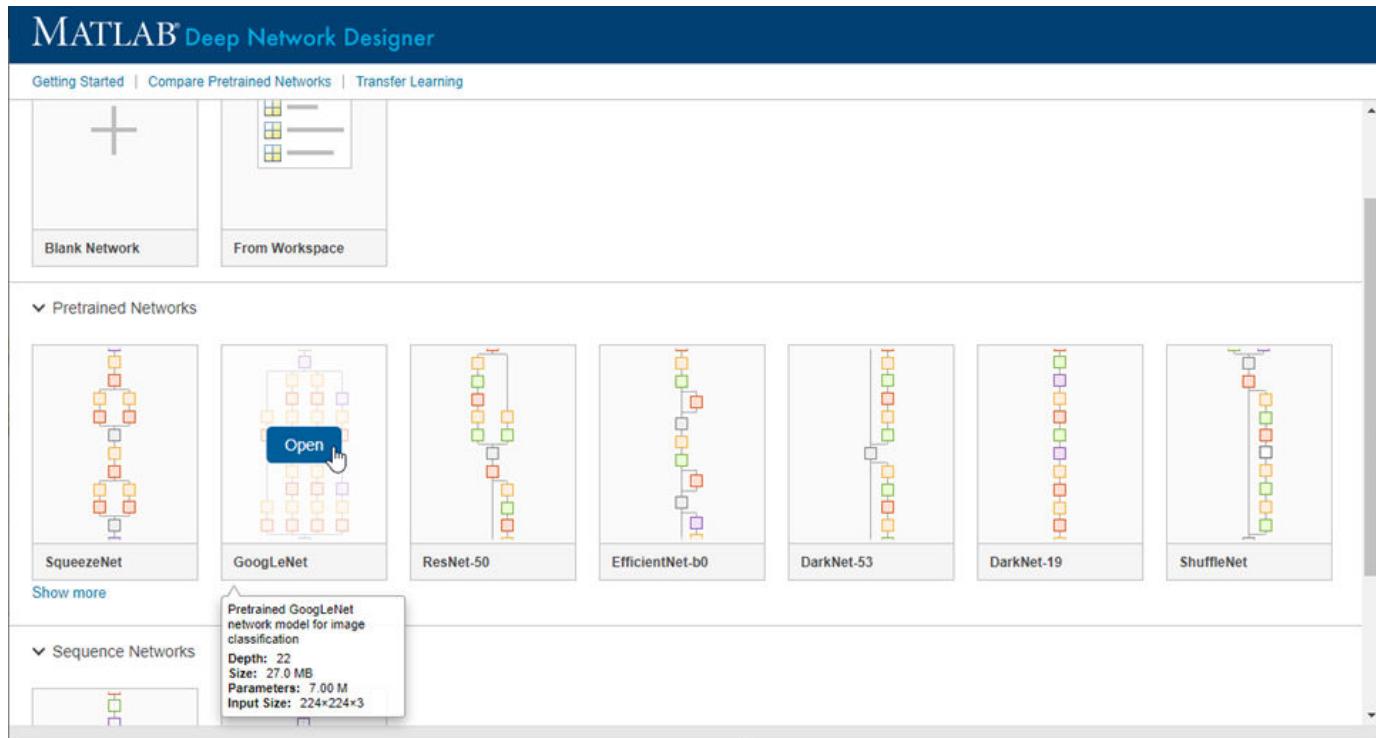
**提示** 从预训练网络开始并通过迁移学习来对它进行微调，通常比从头开始训练新网络要快得多，也容易得多。有关如何使用预训练网络执行迁移学习的示例，请参阅“使用深度网络设计器进行迁移学习”（第 2-2 页）。

## 打开 App 和导入网络

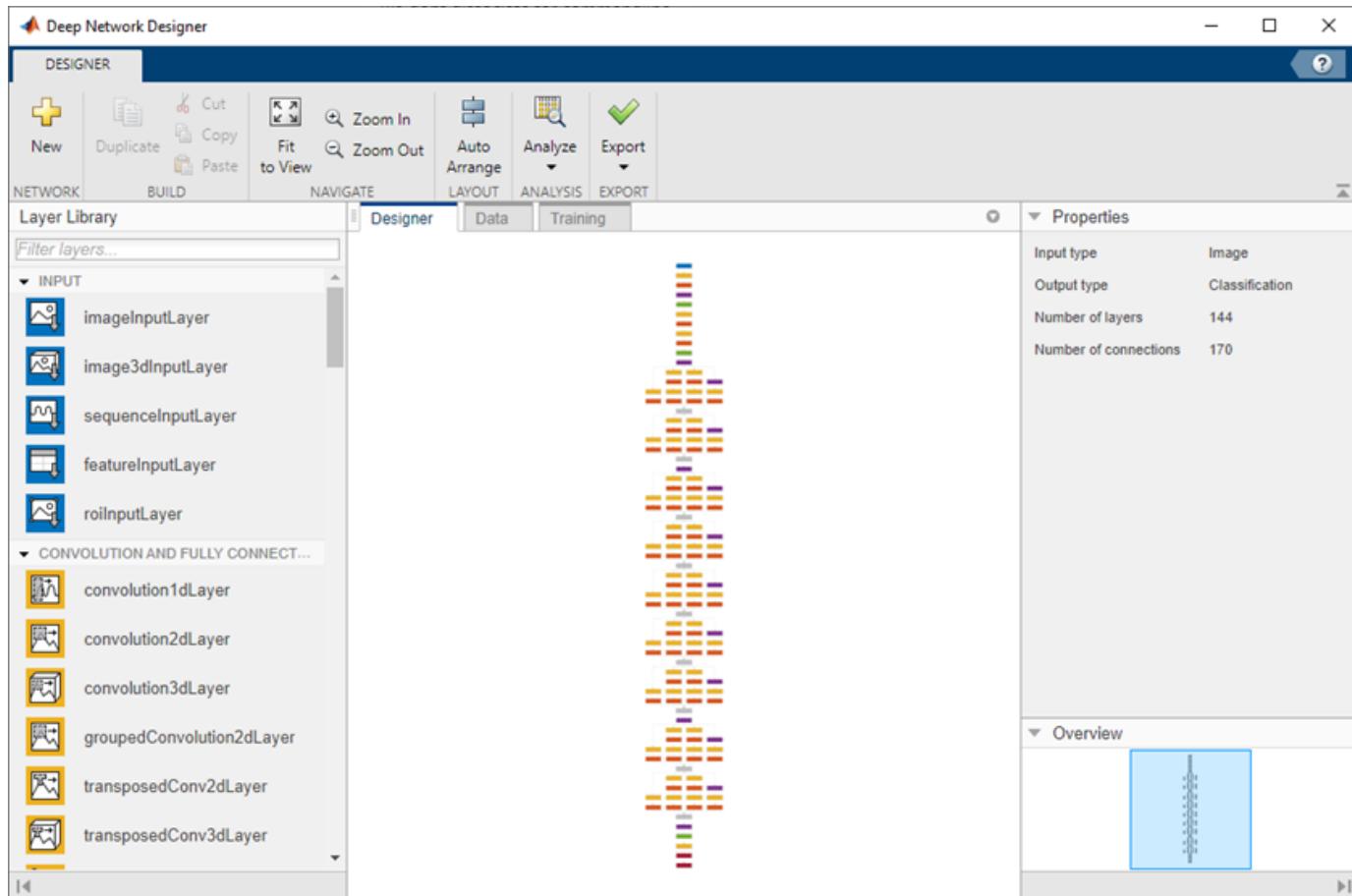
要打开深度网络设计器，请在 **App** 选项卡上的**机器学习和深度学习**下，点击该 App 的图标。您也可以从命令行打开该 App：

**deepNetworkDesigner**

如果您要修改或复制一个现有的预训练网络，您可以从深度网络设计器首页选择它。在首页中，您还可以选择未经训练的序列网络、从工作区加载网络或从头开始构建网络。



选择网络后，深度网络设计器会打开该网络并显示一个缩小视图。

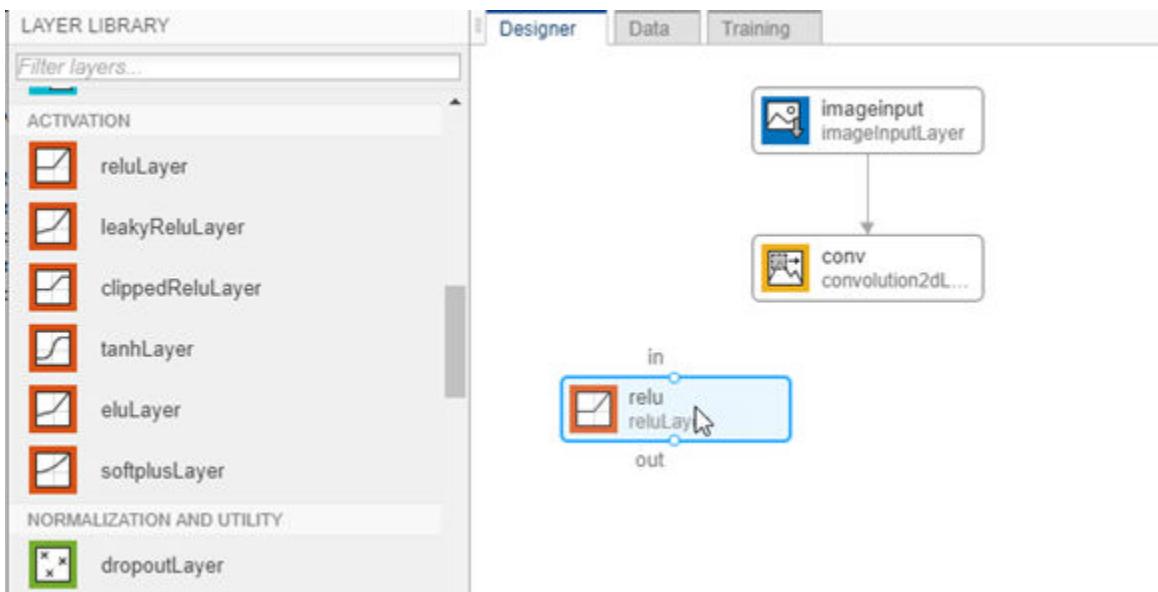


在深度网络设计器的**设计器**窗格中，您可以构造、编辑和分析您的网络。在该 App 中，您可以使用任何内置层来构建网络。您也可以通过在命令行创建自定义层，然后将其导入 App 来使用它。有关在深度网络设计器中使用自定义输出层构造网络的示例，请参阅“Import Custom Layer into Deep Network Designer”。有关可用层的列表和自定义层的示例，请参阅“深度学习层列表”（第 1-18 页）。

通过点击**新建**，从**设计器**选项卡返回到首页。

### 创建和编辑网络

通过从**网络层库**中拖出模块并连接它们来组建一个网络。

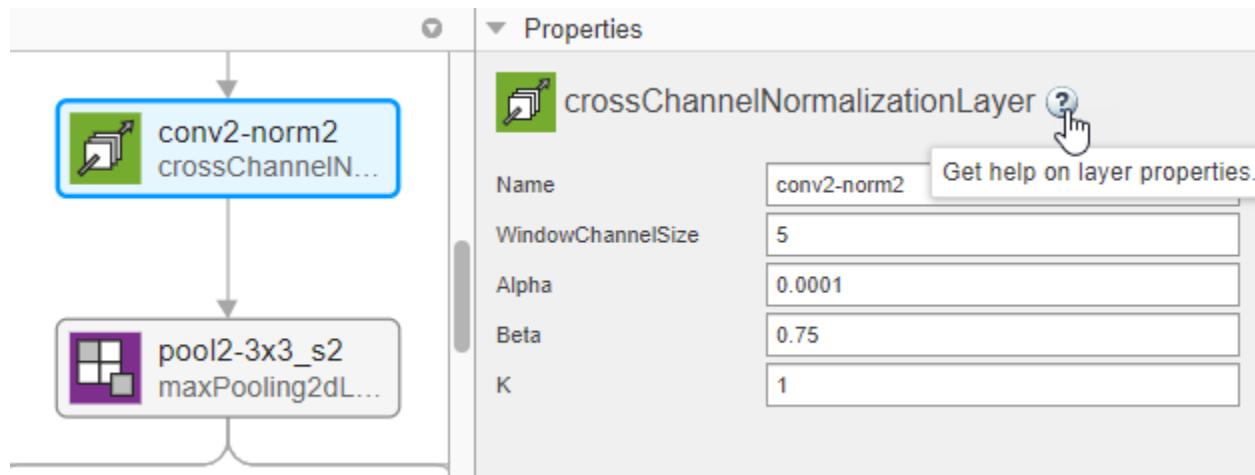


You can also add layers from the workspace to the network in the **Designer** pane.

- 1 Click **New**.
- 2 Pause on **From Workspace** and click **Import**.
- 3 Choose the layers or network to import and click **OK**.
- 4 Click **Add** to add the layers or network to the **Designer** pane.
- 5 Connect the new layers.

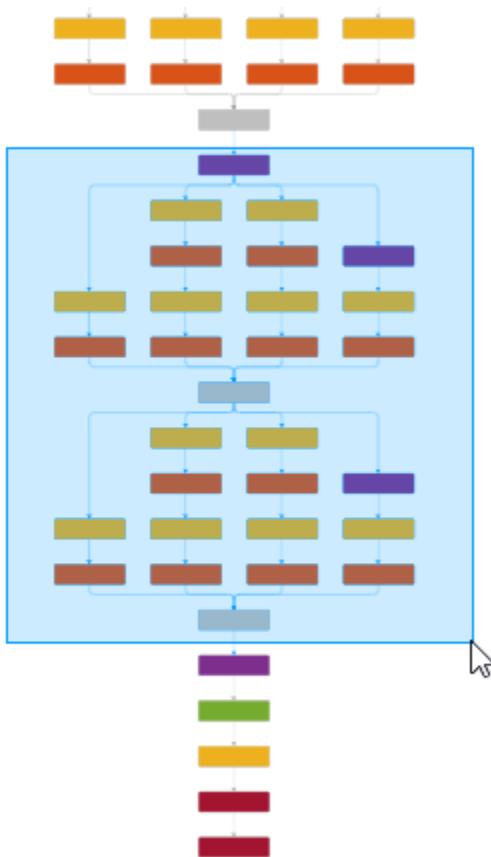
You can also combine pretrained networks by clicking **New** and selecting them from the start page.

要查看和编辑层属性，请选择一个层。有关层属性的信息，请点击层名称旁边的帮助图标。



有关所有层属性的信息，请点击“深度学习层列表”（第 1-18 页）页上的表中的层名称。有关选择合适网络架构的提示，请参阅“Deep Learning Tips and Tricks”。

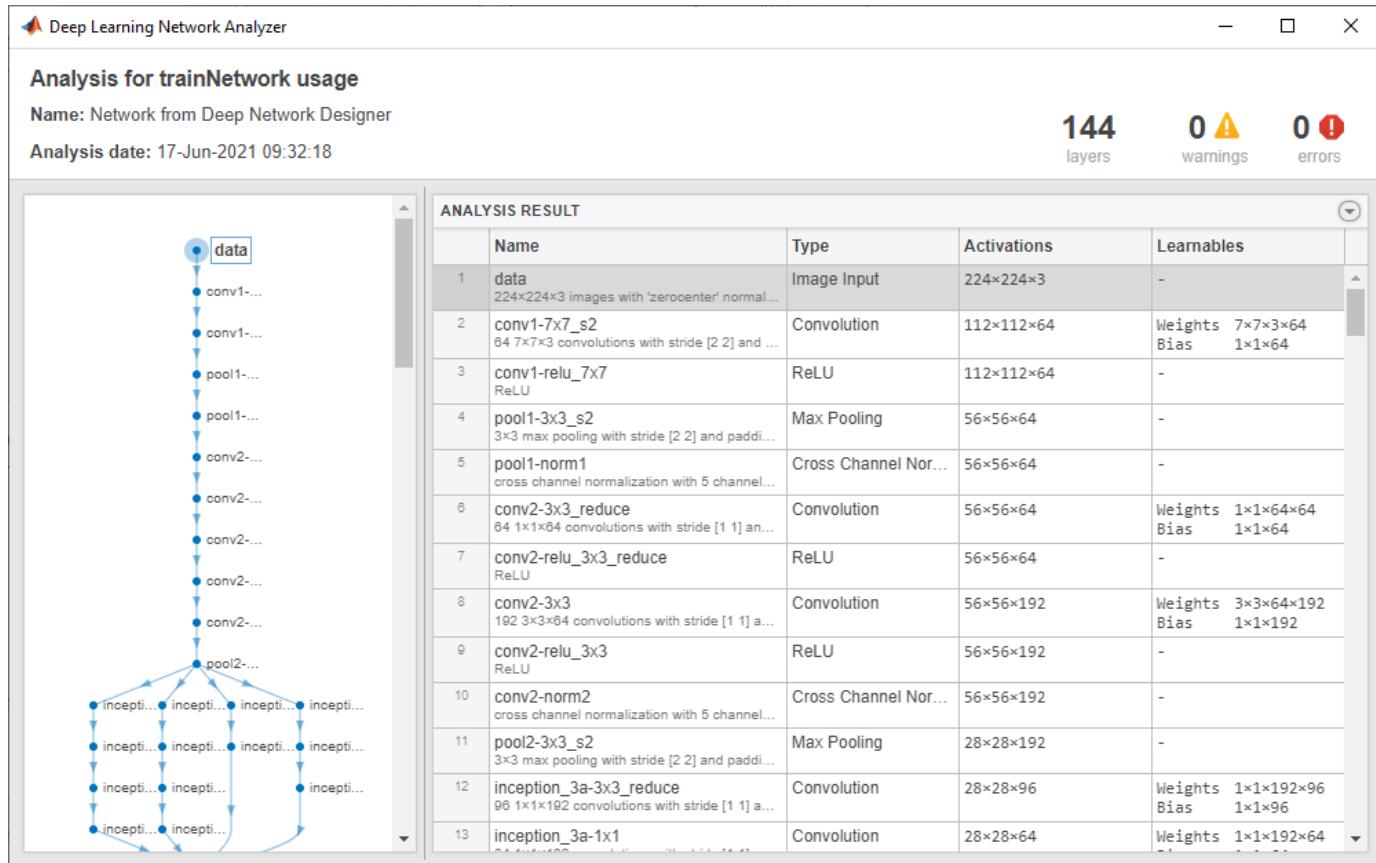
您可以一次处理多个层块。选择多个层，然后复制和粘贴或删除。例如，您可以使用层块来创建卷积、批量归一化和 ReLU 层组的多个副本。您可以预训练网络的末尾添加层来增加网络深度。您也可以编辑预训练网络来简化它。例如，您可以通过从 GoogLeNet 网络中删除层单元（如初始模块）来创建一个更简单的网络。



对于经过训练的网络，复制层会同时复制权重和偏置。

### 检查网络

要检查网络并进一步详细检查层，请在**设计器**选项卡上，点击**分析**。调查问题并检查层属性，以解决网络中的大小不匹配问题。返回到深度网络设计器以编辑层，然后通过再次点击**分析**检查结果。如果 Deep Learning Network Analyzer 报告零错误，则表示编辑过的网络已准备就绪，可以开始训练。



## 训练网络

您可以使用深度网络设计器导入数据并训练网络。在**数据**选项卡上，选择您要用于训练网络的数据。选择**导入数据 > 导入图像数据**，从工作区导入一个 `ImageDatastore` 对象，或导入一个包含图像子文件夹（按类划分）的文件夹。选择**导入数据 > 导入数据存储**，从工作区导入一个内置或自定义数据存储。有关详细信息，请参阅“Import Data into Deep Network Designer”。

导入数据后，通过选择**训练**选项卡并点击**训练**，对网络进行训练。深度网络设计器会复制您在**设计器**窗格中构造的网络，然后使用导入的数据训练该网络。如果您要更好地控制训练，请点击**训练选项**来修改训练选项。有关导入数据和训练在深度网络设计器中构造的网络的详细信息，请参阅“使用深度网络设计器进行迁移学习”（第 2-2 页）。

**提示** 深度网络设计器可以基于图像数据或数据存储对象训练网络。有关如何导出网络并针对序列分类问题对其进行训练的示例，请参阅“使用深度网络设计器创建简单的序列分类网络”（第 2-21 页）。

## 导出网络

要将未经训练的网络导出到工作区进行训练，请在**设计器**选项卡上，点击**导出**。深度网络设计器将网络导出到包含已编辑网络层的一个新变量。

有关显示如何设置训练选项和评估经过训练的网络准确度的命令行示例，请参阅“创建简单的深度学习网络以用于分类”（第 3-39 页）和“训练残差网络进行图像分类”（第 3-13 页）。

要将经过训练的网络导出到工作区，请在训练选项卡上，点击导出。导出的网络具有由深度网络设计器训练的权重的层。

### 生成 MATLAB 代码

使用深度网络设计器，您可以生成 MATLAB 代码，以重新创建在该 App 中执行的网络构造和训练。

有关如何生成用于重新创建网络架构的 MATLAB 代码的示例，请参阅“Generate MATLAB Code to Recreate Network Layers”。

有关如何生成用于重新创建网络架构和网络训练的 MATLAB 代码的示例，请参阅“Generate MATLAB Code to Train Network”。

**另请参阅**  
深度网络设计器

### 相关示例

- “Import Data into Deep Network Designer”
- “使用深度网络设计器创建简单的序列分类网络”（第 2-21 页）
- “深度学习层列表”（第 1-18 页）
- “使用深度网络设计器进行迁移学习”（第 2-2 页）
- “Generate MATLAB Code from Deep Network Designer”
- “Deep Learning Tips and Tricks”

# 使用深度网络设计器创建简单的序列分类网络

此示例说明如何使用深度网络设计器创建简单的长短期记忆 (LSTM) 分类网络。

要训练深度神经网络以对序列数据进行分类，可以使用 LSTM 网络。LSTM 网络是一种循环神经网络 (RNN)，可学习序列数据的时间步之间的长期依存关系。

该示例演示如何：

- 加载序列数据。
- 构造网络架构。
- 指定训练选项。
- 训练网络。
- 预测新数据的标签并计算分类准确度。

## 加载数据

按照 [1] (第 2-0 页) 和 [2] (第 2-0 页) 中的说明加载日语元音数据集。预测变量是包含不同长度序列的元胞数组，特征维度为 12。标签是由标签 1、2、...、9 组成的分类向量。

```
[XTrain,YTrain] = japaneseVowelsTrainData;
[XValidation,YValidation] = japaneseVowelsTestData;
```

查看前几个训练序列的大小。序列是具有 12 行（每个特征一行）和不同列数（每个时间步一列）的矩阵。

**XTrain(1:5)**

```
ans=5×1 cell array
{12×20 double}
{12×26 double}
{12×22 double}
{12×20 double}
{12×21 double}
```

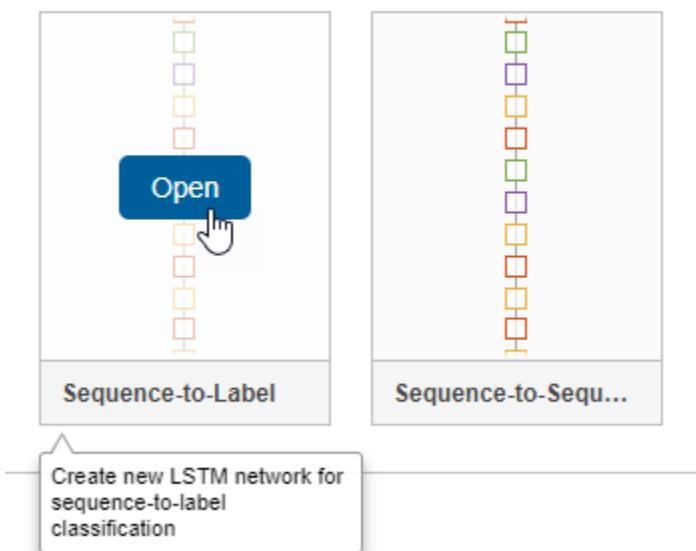
## 定义网络架构

打开深度网络设计器。

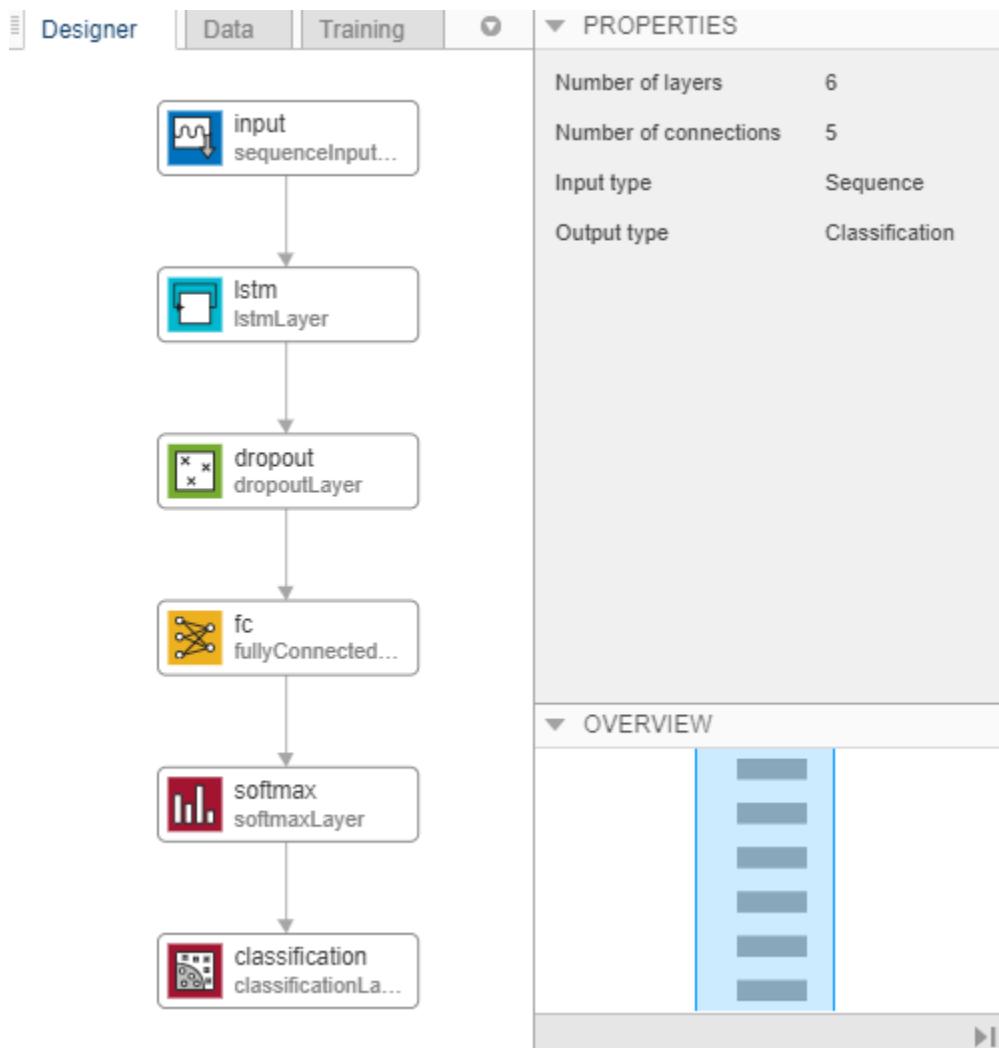
**deepNetworkDesigner**

在**序列到标签**上暂停，然后点击**打开**。这会打开一个适合序列分类问题的预置网络。

### ▼ Sequence Networks

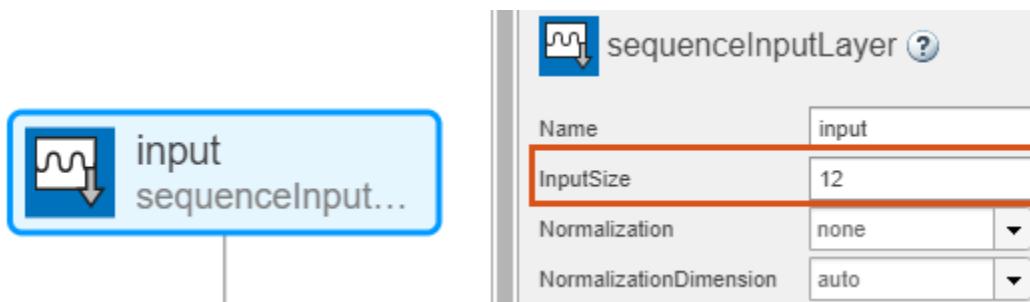


深度网络设计器显示该预置网络。

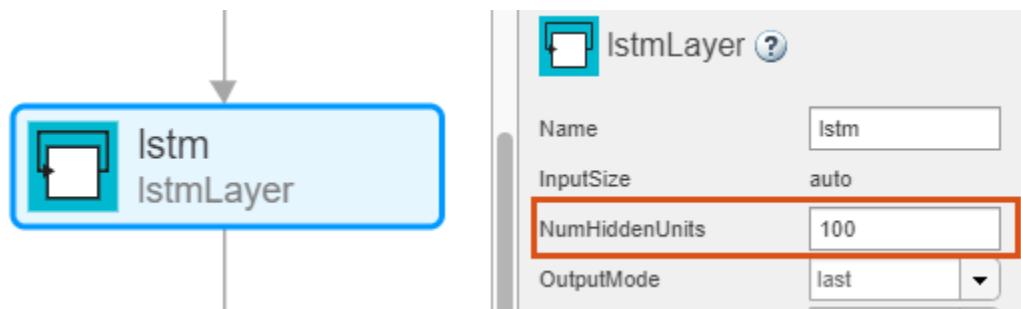


您可以轻松地将此序列网络用于日语元音字母数据集。

选择 **sequencelInputLayer**，检查并确认 **InputSize** 设置为 12，与特征维度匹配。



选择 **LSTM** 并将 **NumHiddenUnits** 设置为 100。



选择 **fullyConnectedLayer**, 检查并确认 **OutputSize** 设置为 9, 即类的数目。



### 检查网络架构

要检查网络并查看层的详细信息, 请[点击分析](#)。

Deep Learning Network Analyzer

Network from Deep Network Designer

Analysis date: 11-May-2020 10:36:43

6 layers    0 warnings    0 errors

	Name	Type	Activations	Learnables
1	input Sequence input with 12 dimensions	Sequence Input	12	-
2	lstm LSTM with 100 hidden units	LSTM	100	InputWeights 400x12 RecurrentWeights 400x100 Bias 400x1
3	dropout 50% dropout	Dropout	100	-
4	fc 9 fully connected layer	Fully Connected	9	Weights 9x100 Bias 9x1
5	softmax softmax	Softmax	9	-
6	classification crossentropyex	Classification Output	-	-

### 导出网络架构

要将网络架构导出到工作区, 请在[设计器](#)选项卡上, 点击[导出](#)。深度网络设计器将网络保存为变量 `layers_1`。

您还可以通过选择导出 > 生成代码来生成用于构造网络架构的代码。

## 训练网络

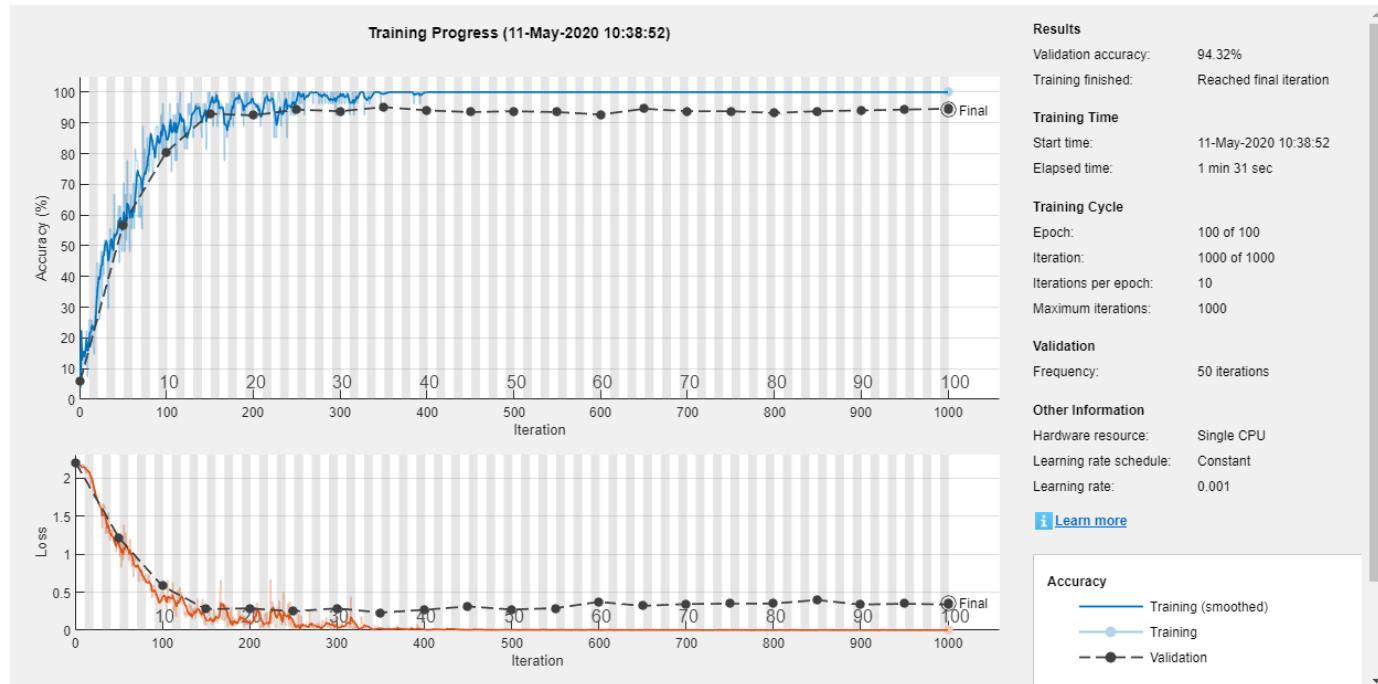
指定训练选项并训练网络。

由于小批量数据存储较小且序列较短，因此更适合在 CPU 上训练。将 'ExecutionEnvironment' 设置为 'cpu'。要在 GPU (如果可用) 上进行训练，请将 'ExecutionEnvironment' 设置为 'auto' (默认值)。

```
miniBatchSize = 27;
options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'MaxEpochs',100, ...
    'MiniBatchSize',miniBatchSize, ...
    'ValidationData',{XValidation,YValidation}, ...
    'GradientThreshold',2, ...
    'Shuffle','every-epoch', ...
    'Verbose',false, ...
    'Plots','training-progress');
```

训练网络。

```
net = trainNetwork(XTrain,YTrain,layers_1,options);
```



## 测试网络

对测试数据进行分类，并计算分类准确度。指定与训练相同的小批量大小。

```
YPred = classify(net,XValidation,'MiniBatchSize',miniBatchSize);
acc = mean(YPred == YValidation)
```

```
acc = 0.9432
```

在接下来的步骤中，您可以尝试通过使用双向 LSTM (BiLSTM) 层或创建更深的网络来提高准确度。有关详细信息，请参阅“长短期记忆网络”（第 1-38 页）。

有关说明如何使用卷积网络对序列数据进行分类的示例，请参阅“使用深度学习进行语音命令识别”（第 4-16 页）。

### 参考资料

[1] Kudo, Mineichi, Jun Toyama, and Masaru Shimbo. "Multidimensional Curve Classification Using Passing-through Regions." *Pattern Recognition Letters* 20, no. 11–13 (November 1999):1103–11. [https://doi.org/10.1016/S0167-8655\(99\)00077-X](https://doi.org/10.1016/S0167-8655(99)00077-X).

[2] Kudo, Mineichi, Jun Toyama, and Masaru Shimbo. Japanese Vowels Data Set. Distributed by UCI Machine Learning Repository. <https://archive.ics.uci.edu/ml/datasets/Japanese+Vowels>

### 另请参阅

深度网络设计器

### 相关示例

- “深度学习层列表”（第 1-18 页）
- “使用深度网络设计器进行迁移学习”（第 2-2 页）
- “Generate MATLAB Code from Deep Network Designer”
- “Deep Learning Tips and Tricks”

# 图像深度学习

---

- “使用深度学习对网络摄像头图像进行分类” (第 3-2 页)
- “训练深度学习网络以对新图像进行分类” (第 3-6 页)
- “训练残差网络进行图像分类” (第 3-13 页)
- “使用 GoogLeNet 对图像进行分类” (第 3-22 页)
- “使用预训练网络提取图像特征” (第 3-27 页)
- “使用 AlexNet 进行迁移学习” (第 3-32 页)
- “创建简单的深度学习网络以用于分类” (第 3-39 页)
- “针对回归训练卷积神经网络” (第 3-44 页)
- “将分类网络转换为回归网络” (第 3-52 页)
- “训练生成对抗网络 (GAN)” (第 3-57 页)
- “训练变分自编码器 (VAE) 以生成图像” (第 3-68 页)

## 使用深度学习对网络摄像头图像进行分类

此示例说明如何使用预训练的深度卷积神经网络 GoogLeNet 实时对来自网络摄像头的图像进行分类。

使用 MATLAB®、普通的网络摄像头和深度神经网络来识别周围环境中的对象。此示例使用 GoogLeNet，它是预训练的深度卷积神经网络（CNN 或 ConvNet），已基于超过一百万个图像进行训练，可以将图像分为 1000 个对象类别（例如键盘、咖啡杯、铅笔和多种动物）。您可以下载 GoogLeNet 并使用 MATLAB 实时连续处理相机图像。

GoogLeNet 已基于大量图像学习了丰富的特征表示。它以图像作为输入，然后输出图像中对象的标签以及每个对象类别的概率。您可以用周围的物品进行试验，以了解 GoogLeNet 对图像进行分类的准确程度。要了解有关网络对象分类的详细信息，可以实时显示排名前五的类的分数，而不是只显示最终的类决策。

### 加载相机和预训练网络

连接到相机并加载预训练的 GoogLeNet 网络。您可以在此步骤使用任何预训练网络。该示例需要 MATLAB Support Package for USB Webcams，以及 Deep Learning Toolbox™ Model for GoogLeNet Network。如果没有安装所需的支持包，软件会提供下载链接。

```
camera = webcam;
net = googlenet;
```

如果要再次运行该示例，请先运行命令 `clear camera`，其中 `camera` 是与网络摄像头的连接。否则将出现错误，因为您不能创建与同一网络摄像头的另一连接。

### 对相机快照进行分类

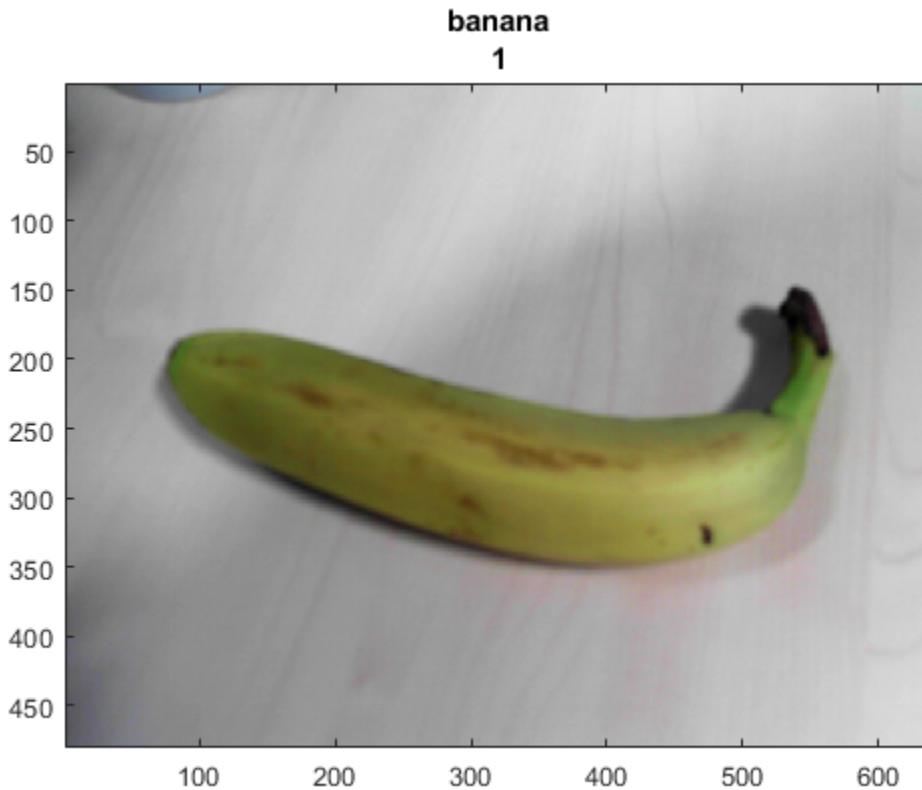
要对图像进行分类，必须将其大小调整为网络的输入大小。获取网络的图像输入层的 `InputSize` 属性的前两个元素。图像输入层是网络的第一层。

```
inputSize = net.Layers(1).InputSize(1:2)
```

```
inputSize =
224 224
```

显示来自相机的图像以及预测的标签及其概率。在调用 `classify` 之前，必须将图像大小调整为网络的输入大小。

```
figure
im = snapshot(camera);
image(im)
im = imresize(im,inputSize);
[label,score] = classify(net,im);
title({char(label),num2str(max(score),2)});
```



### 连续对相机图像进行分类

要连续对相机图像进行分类，请将前面的步骤放入一个循环。在图窗打开时运行该循环。要停止实时预测，只需关闭图窗。在每次迭代结束时使用 **drawnow** 更新图窗。

```

h = figure;

while ishandle(h)
    im = snapshot(camera);
    image(im)
    im = imresize(im,inputSize);
    [label,score] = classify(net,im);
    title({char(label), num2str(max(score),2)});
    drawnow
end

```

### 显示排名靠前的预测值

预测出的类可能快速更改。因此，将排名靠前的预测值显示在一起会有所帮助。您可以通过绘制预测分数靠前的类来显示排名前五的预测值及其概率。

对相机的快照进行分类。显示来自相机的图像以及预测的标签及其概率。使用 **classify** 函数的 **score** 输出显示排名前五的预测值的概率直方图。

创建图窗窗口。首先，调整窗口大小以使宽度增加一倍，并创建两个子图。

```

h = figure;
h.Position(3) = 2*h.Position(3);

```

```
ax1 = subplot(1,2,1);
ax2 = subplot(1,2,2);
```

在左侧子图中，将图像和分类显示在一起。

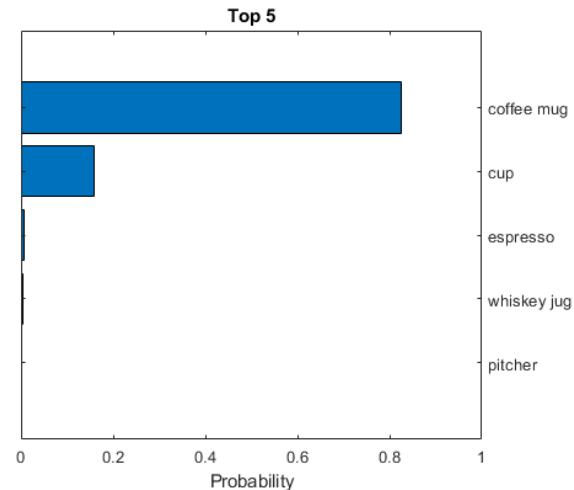
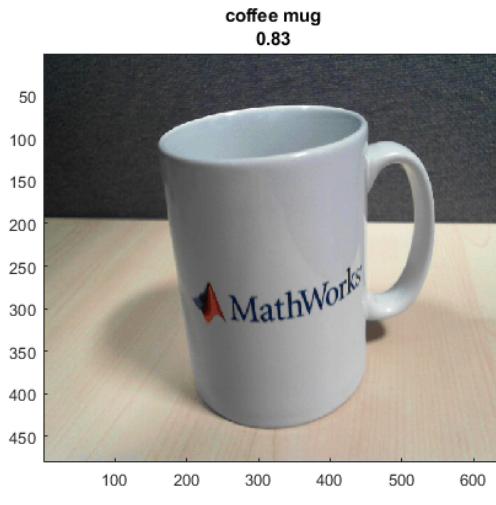
```
im = snapshot(camera);
image(ax1,im)
im = imresize(im,inputSize);
[label,score] = classify(net,im);
title(ax1,{char(label),num2str(max(score),2)});
```

通过选择分数最高的类来选择排名前五的预测值。

```
[~,idx] = sort(score,'descend');
idx = idx(5:-1:1);
classes = net.Layers(end).Classes;
classNamesTop = string(classes(idx));
scoreTop = score(idx);
```

将排名前五的预测值显示为直方图。

```
barh(ax2,scoreTop)
xlim(ax2,[0 1])
title(ax2,'Top 5')
xlabel(ax2,'Probability')
yticklabels(ax2,classNamesTop)
ax2.YAxisLocation = 'right';
```



### 连续分类图像并显示排名靠前的预测值

要连续对来自相机的图像进行分类并显示排名靠前的预测值，请将前面的步骤放入一个循环。在图窗打开时运行该循环。要停止实时预测，只需关闭图窗。在每次迭代结束时使用 `drawnow` 更新图窗。

创建图窗窗口。首先调整窗口大小以使宽度增加一倍，并创建两个子图。要防止坐标区大小改变，请将 **PositionConstraint** 属性设置为 '`innerposition`'。

```
h = figure;
h.Position(3) = 2*h.Position(3);
ax1 = subplot(1,2,1);
```

```
ax2 = subplot(1,2,2);
ax2.PositionConstraint = 'innerposition';
```

连续显示并分类图像，同时显示排名前五的预测值的直方图。

```
while ishandle(h)
    % Display and classify the image
    im = snapshot(camera);
    image(ax1,im)
    im = imresize(im,inputSize);
    [label,score] = classify(net,im);
    title(ax1,[char(label),num2str(max(score),2)]);

    % Select the top five predictions
    [~,idx] = sort(score,'descend');
    idx = idx(5:-1:1);
    scoreTop = score(idx);
    classNamesTop = string(classes(idx));

    % Plot the histogram
    barh(ax2,scoreTop)
    title(ax2,'Top 5')
    xlabel(ax2,'Probability')
    xlim(ax2,[0 1])
    yticklabels(ax2,classNamesTop)
    ax2.YAxisLocation = 'right';

    drawnow
end
```

## 另请参阅

[googlenet](#) | [vgg19](#) | [classify](#)

## 相关示例

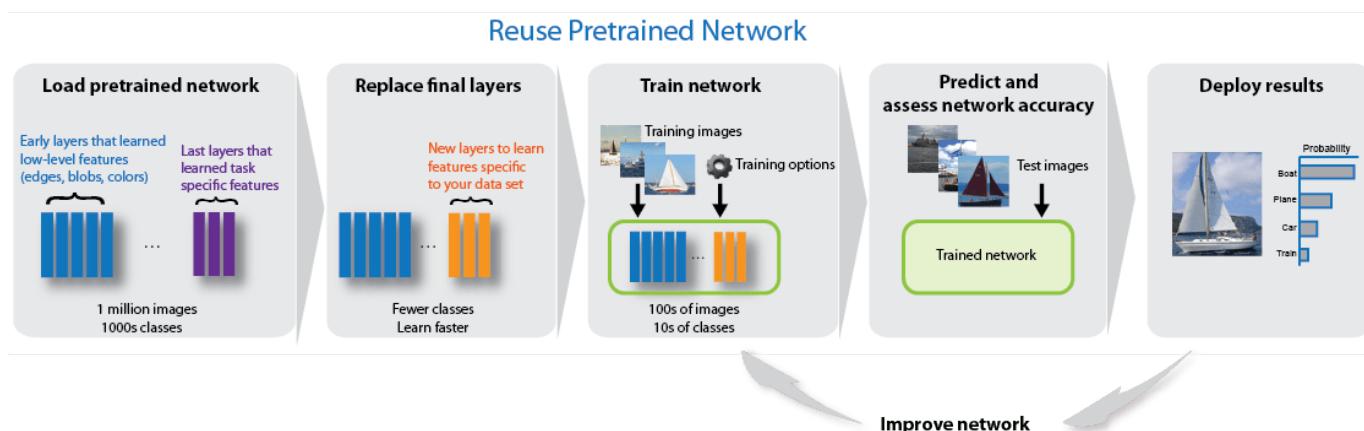
- “Transfer Learning Using Pretrained Network”
- “预训练的深度神经网络”（第 1-8 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）

## 训练深度学习网络以对新图像进行分类

此示例说明如何使用迁移学习来重新训练卷积神经网络以对新图像集进行分类。

预训练的图像分类网络已经对超过一百万个图像进行了训练，可以将图像分为 1000 个对象类别，例如键盘、咖啡杯、铅笔和多种动物。这些网络已基于大量图像学习了丰富的特征表示。网络以图像作为输入，然后输出图像中对象的标签以及每个对象类别的概率。

深度学习应用中常常用到迁移学习。您可以采用预训练的网络，基于它学习新任务。与使用随机初始化的权重从头训练网络相比，通过迁移学习来微调网络要更快更简单。您可以使用较少数量的训练图像快速地将已学习的特征迁移到新任务。



### 加载数据

解压缩新图像并加载这些图像作为图像数据存储。这个非常小的数据集只包含 75 个图像。将数据划分为训练数据集和验证数据集。将 70% 的图像用于训练，30% 的图像用于验证。

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7);
```

### 加载预训练网络

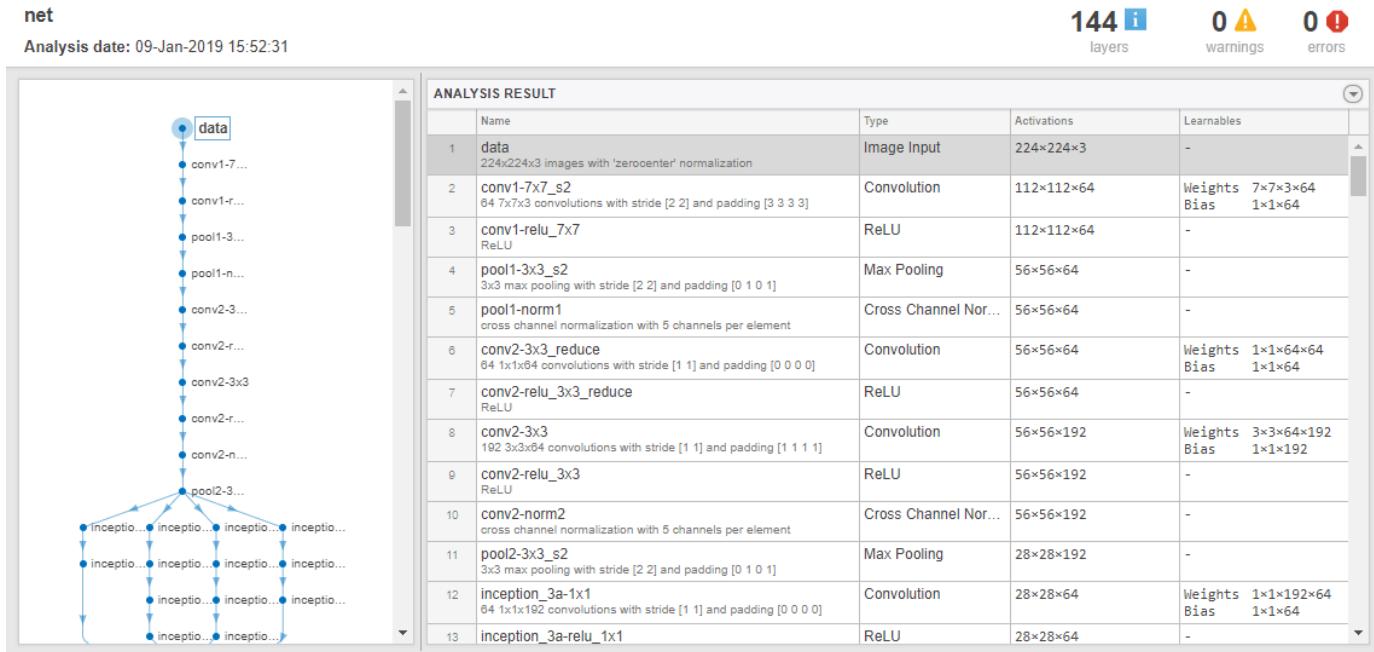
加载预训练的 GoogLeNet 网络。如果未安装 Deep Learning Toolbox™ Model for GoogLeNet Network 支持包，则软件会提供下载链接。

要尝试不同的预训练网络，请在 MATLAB® 中打开此示例并选择其他网络。例如，您可以尝试 `squeezeNet`，这是一个比 `googlenet` 还要快的网络。您可以使用其他预训练网络运行此示例。有关所有可用网络的列表，请参阅“加载预训练网络”（第 1-9 页）。

```
net = googlenet;
```

使用 `analyzeNetwork` 可以交互可视方式呈现网络架构以及有关网络层的详细信息。

```
analyzeNetwork(net)
```



网络的 `Layers` 属性的第一个元素是图像输入层。对于 GoogLeNet 网络，该层需要大小为  $224 \times 224 \times 3$  的输入图像，其中 3 是颜色通道数。其他网络可能需要不同大小的输入图像。例如，Xception 网络需要大小为  $299 \times 299 \times 3$  的图像。

`net.Layers(1)`

```
ans =
ImageInputLayer with properties:
```

```
    Name: 'data'
    InputSize: [224 224 3]
```

```
Hyperparameters
DataAugmentation: 'none'
Normalization: 'zerocenter'
NormalizationDimension: 'auto'
Mean: [224x224x3 single]
```

```
inputSize = net.Layers(1).InputSize;
```

### 替换最终层

网络的卷积层会提取最后一个可学习层和最终分类层用来对输入图像进行分类的图像特征。GoogLeNet 中的 '`loss3-classifier`' 和 '`output`' 这两个层包含有关如何将网络提取的特征合并为类概率、损失值和预测标签的信息。要对预训练网络进行重新训练以对新图像进行分类，请将这两个层替换为适合新数据集的新层。

从经过训练的网络中提取层次图。如果网络是 `SeriesNetwork` 对象（如 AlexNet、VGG-16 或 VGG-19），请将 `net.Layers` 中的层列表转换为层次图。

```
if isa(net,'SeriesNetwork')
lgraph = layerGraph(net.Layers);
```

```

else
    lgraph = layerGraph(net);
end

```

查找要替换的两个层的名称。您可以手动执行此操作，也可以使用支持函数 `findLayersToReplace` 自动查找这两个层。

```
[learnableLayer,classLayer] = findLayersToReplace(lgraph);
[learnableLayer,classLayer]
```

```
ans =
1×2 Layer array with layers:
```

```

1 'loss3-classifier' Fully Connected    1000 fully connected layer
2 'output'          Classification Output crossentropyex with 'tench' and 999 other classes

```

在大多数网络中，具有可学习权重的最后一层是全连接层。将此全连接层替换为新的全连接层，其中输出数量等于新数据集中类的数量（在此示例中为 5）。而在某些网络（如 SqueezeNet）中，最后一个可学习层是一个  $1 \times 1$  卷积层。在这种情况下，请将该卷积层替换为新的卷积层，其中滤波器的数量等于类的数量。要使新层中的学习速度快于迁移的层，请增大该层的学习率因子。

```

numClasses = numel(categories(imdsTrain.Labels));

if isa(learnableLayer,'nnet.cnn.layer.FullyConnectedLayer')
    newLearnableLayer = fullyConnectedLayer(numClasses, ...
        'Name','new_fc', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);

elseif isa(learnableLayer,'nnet.cnn.layer.Convolution2DLayer')
    newLearnableLayer = convolution2dLayer(1,numClasses, ...
        'Name','new_conv', ...
        'WeightLearnRateFactor',10, ...
        'BiasLearnRateFactor',10);
end

lgraph = replaceLayer(lgraph,learnableLayer.Name,newLearnableLayer);

```

分类层指定网络的输出类。将分类层替换为没有类标签的新分类层。`trainNetwork` 会在训练时自动设置层的输出类。

```

newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,classLayer.Name,newClassLayer);

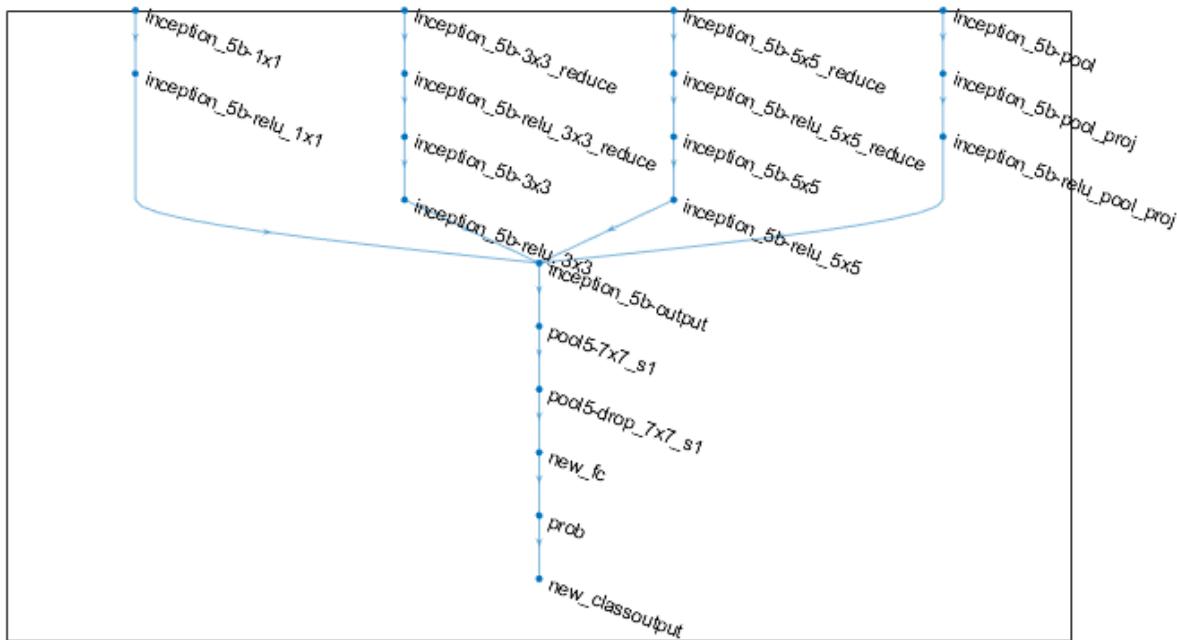
```

要检查新层是否正确连接，请绘制新的层次图并放大网络的最后几层。

```

figure('Units','normalized','Position',[0.3 0.3 0.4 0.4]);
plot(lgraph)
ylim([0,10])

```



## 冻结初始层

现在，网络已准备好针对新的图像集进行重新训练。您也可以选择将较浅网络层的学习率设置为零，来“冻结”这些层的权重。在训练过程中，`trainNetwork` 不会更新已冻结层的参数。由于不需要计算已冻结层的梯度，因此冻结多个初始层的权重可以显著加快网络训练速度。如果新数据集很小，冻结较浅的网络层还可以防止这些层过拟合新数据集。

提取层次图的层和连接，并选择要冻结的层。在 GoogLeNet 中，前 10 个层构成了网络的初始“主干”。使用支持函数 `freezeWeights` 将前 10 个层的学习率设置为零。使用支持函数 `createLgraphUsingConnections` 以原始顺序重新连接所有层。新的层次图包含相同的层，但较浅层的学习率设置为零。

```

layers = lgraph.Layers;
connections = lgraph.Connections;

layers(1:10) = freezeWeights(layers(1:10));
lgraph = createLgraphUsingConnections(layers,connections);

```

## 训练网络

网络要求输入图像的大小为  $224 \times 224 \times 3$ ，但图像数据存储中的图像具有不同大小。使用增强的图像数据存储可自动调整训练图像的大小。指定要对训练图像执行的附加增强操作：沿垂直轴随机翻转训练图像，以及在水平和垂直方向上随机平移训练图像最多 30 个像素并将训练图像缩放最多 10%。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```

pixelRange = [-30 30];
scaleRange = [0.9 1.1];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection', true, ...

```

```
'RandXTranslation',pixelRange, ...
'RandYTranslation',pixelRange, ...
'RandXScale',scaleRange, ...
'RandYScale',scaleRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
'DataAugmentation',imageAugmenter);
```

要在不执行进一步数据增强的情况下自动调整验证图像的大小，请使用增强的图像数据存储，而不指定任何其他预处理操作。

```
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

指定训练选项。将 `InitialLearnRate` 设置为较小的值以减慢尚未冻结的迁移层中的学习速度。在上一步中，您增大了最后一个可学习层的学习率因子，以加快新的最终层中的学习速度。这种学习率设置组合会加快新层中的学习速度，减慢中间层中的学习速度，停止较浅的冻结层中的学习。

指定要训练的轮数。执行迁移学习时，所需的训练轮数相对较少。一轮训练是对整个训练数据集的一个完整训练周期。指定小批量大小和验证数据。每轮计算一次验证准确度。

```
miniBatchSize = 10;
valFrequency = floor(numel(augimdsTrain.Files)/miniBatchSize);
options = trainingOptions('sgdm', ...
'MiniBatchSize',miniBatchSize, ...
'MaxEpochs',6, ...
'InitialLearnRate',3e-4, ...
'Shuffle','every-epoch', ...
'ValidationData',augimdsValidation, ...
'ValidationFrequency',valFrequency, ...
'Verbose',false, ...
'Plots','training-progress');
```

使用训练数据训练网络。默认情况下，`trainNetwork` 使用 GPU（如果有）。这需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。否则，`trainNetwork` 将使用 CPU。您还可以使用 `trainingOptions` 的 `'ExecutionEnvironment'` 名称-值对组参数指定执行环境。由于数据集很小，因此训练很快。

```
net = trainNetwork(augimdsTrain,lgraph,options);
```

### 对验证图像进行分类

使用经过微调的网络对验证图像进行分类，并计算分类准确度。

```
[YPred,probs] = classify(net,augimdsValidation);
accuracy = mean(YPred == imdsValidation.Labels)

accuracy = 0.9000
```

显示四个示例验证图像及预测的标签，以及具有这些标签的图像的预测概率。

```
idx = randperm(numel(imdsValidation.Files),4);
figure
for i = 1:4
    subplot(2,2,i)
    I = readimage(imdsValidation,idx(i));
    imshow(I)
    label = YPred(idx(i));
```

```

title(string(label) + ", " + num2str(100*max(probs(idx(i),:)),3) + "%");
end

```

**MathWorks Playing Cards, 100%****MathWorks Screwdriver, 100%****MathWorks Cap, 70.1%****MathWorks Cube, 100%**

## 参考

[1] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.

[2] *BVLC GoogLeNet Model*. [https://github.com/BVLC/caffe/tree/master/models/bvlc\\_googlenet](https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet)

## 另请参阅

`vgg16` | `vgg19` | `alexnet` | `importCaffeNetwork` | `importCaffeLayers` | `trainNetwork` |  
`layerGraph` | `DAGNetwork` | `plot` | `googlenet` | `analyzeNetwork`

## 相关示例

- “将分类网络转换为回归网络” (第 3-52 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “预训练的深度神经网络” (第 1-8 页)
- “Transfer Learning Using Pretrained Network”

- “训练残差网络进行图像分类” (第 3-13 页)

# 训练残差网络进行图像分类

此示例说明如何创建包含残差连接的深度学习神经网络，并针对 CIFAR-10 数据对其进行训练。残差连接是卷积神经网络架构中的常见元素。使用残差连接可以改善网络中的梯度流，从而可以训练更深的网络。

对于许多应用来说，使用由一个简单的层序列组成的网络就已足够。但是，某些应用要求网络具有更复杂的层次图结构，其中的层可接收来自多个层的输入，也可以输出到多个层。这些类型的网络通常称为有向无环图 (DAG) 网络。残差网络就是一种 DAG 网络，其中的残差（或快捷）连接会绕过主网络层。残差连接让参数梯度可以更轻松地从输出层传播到较浅的网络层，从而能够训练更深的网络。增加网络深度可在执行更困难的任务时获得更高的准确度。

要创建和训练具有层次图结构的网络，请按照以下步骤操作。

- 使用 `layerGraph` 创建一个 `LayerGraph` 对象。层次图指定网络架构。您可以创建一个空的层次图，然后向其中添加层。还可以直接从一组网络层创建一个层次图。这种情况下，`layerGraph` 会依次连接这些层。
- 使用 `addLayers` 向层次图中添加层，使用 `removeLayers` 从层次图中删除层。
- 使用 `connectLayers` 在不同层之间建立层连接，使用 `disconnectLayers` 断开层连接。
- 使用 `plot` 绘制网络架构。
- 使用 `trainNetwork` 训练网络。经过训练的网络是一个 `DAGNetwork` 对象。
- 使用 `classify` 和 `predict` 对新数据执行分类和预测。

您还可以加载预训练网络进行图像分类。有关详细信息，请参阅“[预训练的深度神经网络](#)”（第 1-8 页）。

## 准备数据

下载 CIFAR-10 数据集 [1]。该数据集包含 60,000 个图像。每个图像为  $32 \times 32$  大小，并且具有三个颜色通道 (RGB)。数据集的大小为 175 MB。根据您的 Internet 连接，下载过程可能需要一些时间。

```
datadir = tempdir;
downloadCIFARData(datadir);
```

将 CIFAR-10 训练和测试图像作为四维数组加载。训练集包含 50,000 个图像，测试集包含 10,000 个图像。使用 CIFAR-10 测试图像进行网络验证。

```
[XTrain,YTrain,XValidation,YValidation] = loadCIFARData(datadir);
```

您可以使用以下代码显示训练图像的随机样本。

```
figure;
idx = randperm(size(XTrain,4),20);
im = imtile(XTrain(:,:,idx),'ThumbnailSize',[96,96]);
imshow(im)
```

创建一个 `augmentedImageDatastore` 对象以用于网络训练。在训练过程中，数据存储会沿垂直轴随机翻转训练图像，并在水平方向和垂直方向上将图像随机平移最多四个像素。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
imageSize = [32 32 3];
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
```

```
'RandYTranslation',pixelRange);
augimdsTrain = augmentedImageDatastore(imageSize,XTrain,YTrain, ...
    'DataAugmentation',imageAugmenter, ...
    'OutputSizeMode','randcrop');
```

### 定义网络架构

残差网络架构由以下组件构成：

- 主分支 - 顺序连接的卷积层、批量归一化层和 ReLU 层。
- 残差连接 - 绕过主分支的卷积单元。残差连接和卷积单元的输出按元素相加。当激活区域的大小变化时，残差连接也必须包含  $1 \times 1$  卷积层。残差连接让参数梯度可以更轻松地从输出层流到较浅的网络层，从而能够训练更深的网络。

### 创建主分支

首先创建网络的主分支。主分支包含五部分。

- 初始部分 - 包含图像输入层和带激活函数的初始卷积层。
- 三个卷积层阶段 - 分别具有不同的特征大小 ( $32 \times 32$ 、 $16 \times 16$  和  $8 \times 8$ )。每个阶段包含 N 个卷积单元。在示例的这一部分中，N = 2。每个卷积单元包含两个带激活函数的  $3 \times 3$  卷积层。`netWidth` 参数是网络宽度，定义为网络第一卷积层阶段中的滤波器数目。第二阶段和第三阶段中的前几个卷积单元会将空间维度下采样二分之一。为了使整个网络中每个卷积层所需的计算量大致相同，每次执行空间下采样时，都将滤波器的数量增加一倍。
- 最后部分 - 包含全局平均池化层、全连接层、softmax 层和分类层。

使用 `convolutionalUnit(numF,stride,tag)` 创建一个卷积单元。`numF` 是每一层中卷积滤波器的数量，`stride` 是该单元第一个卷积层的步幅，`tag` 是添加在层名称前面的字符数组。`convolutionalUnit` 函数在示例末尾定义。

为所有层指定唯一名称。卷积单元中的层的名称以 '`SjUk`' 开头，其中 `j` 是阶段索引，`k` 是该阶段内卷积单元的索引。例如，'`S2U1`' 表示第 2 阶段第 1 单元。

```
netWidth = 16;
layers = [
    imageInputLayer([32 32 3],'Name','input')
    convolution2dLayer(3,netWidth,'Padding','same','Name','convInp')
    batchNormalizationLayer('Name','BNInp')
    reluLayer('Name','reluInp')

    convolutionalUnit(netWidth,1,'S1U1')
    additionLayer(2,'Name','add11')
    reluLayer('Name','relu11')
    convolutionalUnit(netWidth,1,'S1U2')
    additionLayer(2,'Name','add12')
    reluLayer('Name','relu12')

    convolutionalUnit(2*netWidth,2,'S2U1')
    additionLayer(2,'Name','add21')
    reluLayer('Name','relu21')
    convolutionalUnit(2*netWidth,1,'S2U2')
    additionLayer(2,'Name','add22')
    reluLayer('Name','relu22')

    convolutionalUnit(4*netWidth,2,'S3U1')
    additionLayer(2,'Name','add31')
```

```

reluLayer('Name','relu31')
convolutionalUnit(4*netWidth,1,'S3U2')
additionLayer(2,'Name','add32')
reluLayer('Name','relu32')

averagePooling2dLayer(8,'Name','globalPool')
fullyConnectedLayer(10,'Name','fcFinal')
softmaxLayer('Name','softmax')
classificationLayer('Name','classoutput')
];

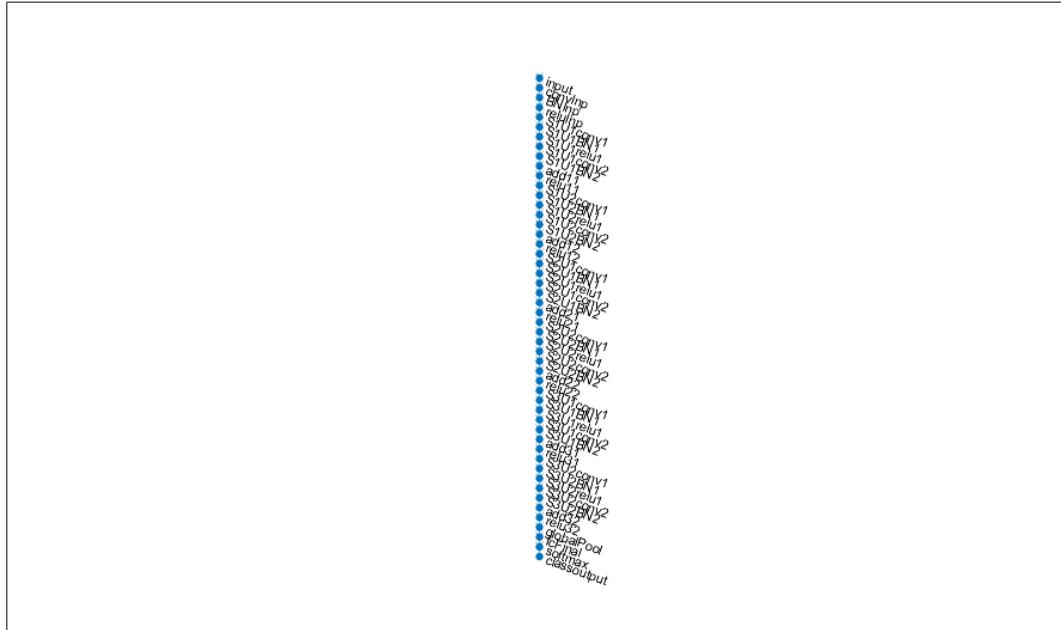
```

根据层数组创建一个层次图。 layerGraph 按顺序连接 layers 中的所有层。绘制层次图。

```

lgraph = layerGraph(layers);
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
plot(lgraph);

```



## 创建残差连接

在卷积单元周围添加残差连接。大多数残差连接不执行任何操作，只是简单地按元素与卷积单元的输出相加。

创建从 'reluInp' 到 'add11' 层的残差连接。由于您在创建相加层时将其输入数指定为 2，因此该层有两个输入，名为 'in1' 和 'in2'。第一个卷积单元的最终层已连接到 'in1' 输入。因此，相加层将第一个卷积单元的输出和 'reluInp' 层相加。

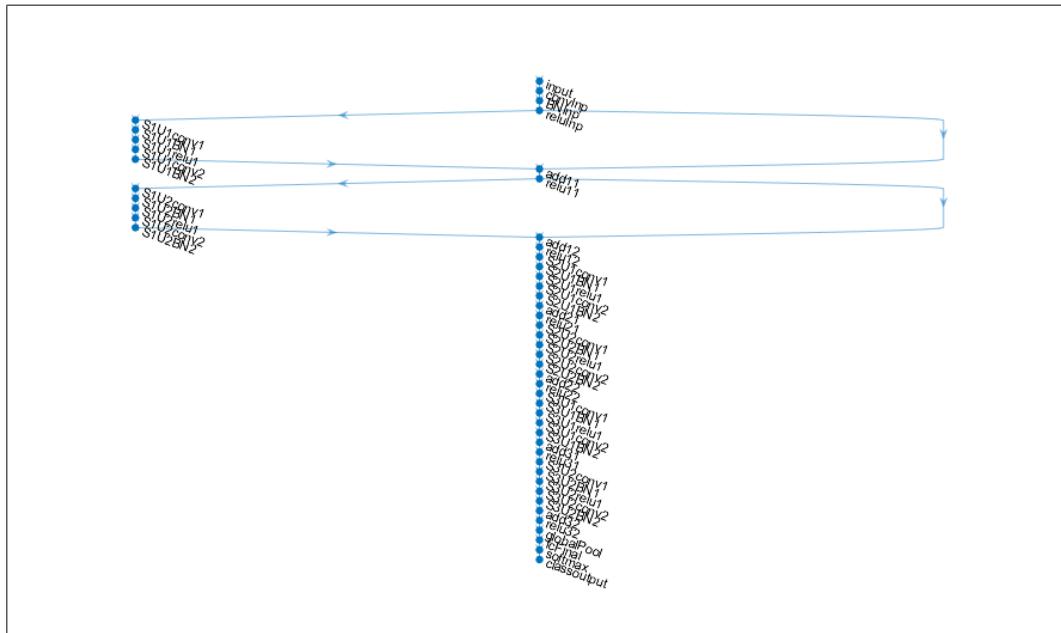
同样，将 'relu11' 层连接到 'add12' 层的第二个输入。通过绘制层次图，确认已正确连接各个层。

```

lgraph = connectLayers(lgraph,'reluInp','add11/in2');
lgraph = connectLayers(lgraph,'relu11','add12/in2');

```

```
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
plot(lgraph);
```



当卷积单元中层激活区域的大小发生变化时（即，当它们在空间维度下采样而在通道维度上采样时），残差连接中激活区域的大小也必须随之变化。通过使用  $1 \times 1$  卷积层及其批量归一化层，更改残差连接中激活区域的大小。

```
skip1 = [
    convolution2dLayer(1,2*netWidth,'Stride',2,'Name','skipConv1')
    batchNormalizationLayer('Name','skipBN1')];
lgraph = addLayers(lgraph,skip1);
lgraph = connectLayers(lgraph,'relu12','skipConv1');
lgraph = connectLayers(lgraph,'skipBN1','add21/in2');
```

在网络的第二阶段添加恒等连接。

```
lgraph = connectLayers(lgraph,'relu21','add22/in2');
```

通过另一个  $1 \times 1$  卷积层及其批量归一化层，更改第二阶段和第三阶段之间的残差连接中激活区域的大小。

```
skip2 = [
    convolution2dLayer(1,4*netWidth,'Stride',2,'Name','skipConv2')
    batchNormalizationLayer('Name','skipBN2')];
lgraph = addLayers(lgraph,skip2);
lgraph = connectLayers(lgraph,'relu22','skipConv2');
lgraph = connectLayers(lgraph,'skipBN2','add31/in2');
```

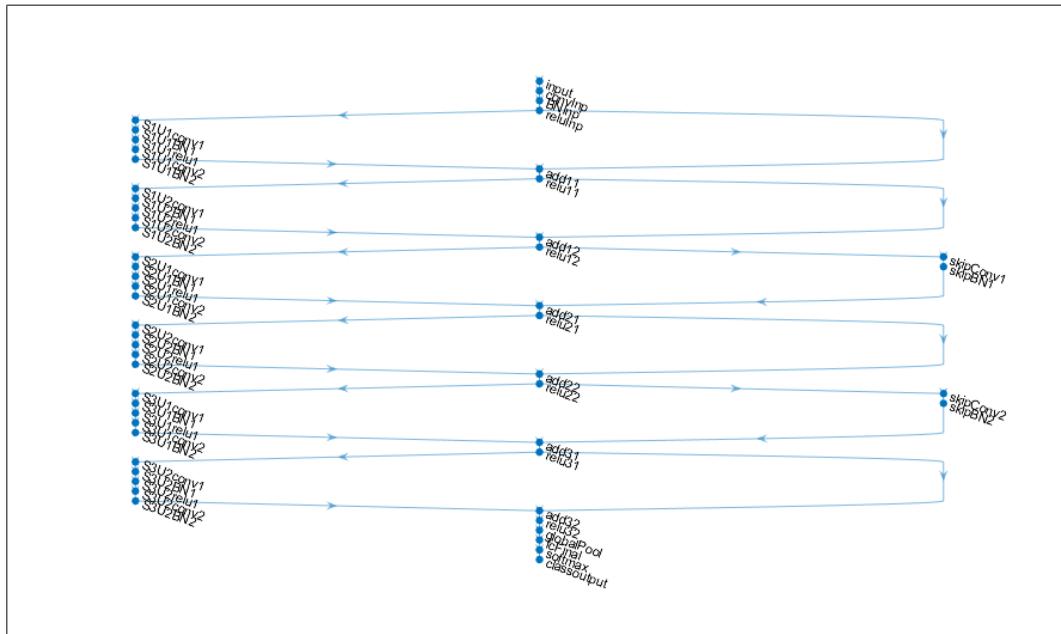
添加最后一个恒等连接，并绘制最终的层次图。

```

lgraph = connectLayers(lgraph,'relu31','add32/in2');

figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
plot(lgraph)

```



## 创建更深的网络

要为任意深度和宽度的 CIFAR-10 数据创建具有残差连接的层次图，请使用支持函数 `residualCIFARlgraph`。

`lgraph = residualCIFARlgraph(netWidth,numUnits,unitType)` 为 CIFAR-10 数据创建具有残差连接的层次图。

- `netWidth` 是网络宽度，定义为网络的前几个  $3 \times 3$  卷积层中的滤波器数量。
- `numUnits` 是网络主分支中的卷积单元数。因为网络由三个阶段组成，其中每个阶段的卷积单元数量都相同，所以 `numUnits` 必须是 3 的整数倍。
- `unitType` 是卷积单元的类型，指定为 "standard" 或 "bottleneck"。一个标准卷积单元由两个  $3 \times 3$  卷积层组成。一个瓶颈卷积单元由三个卷积层组成：一个在通道维度进行下采样的  $1 \times 1$  层，一个  $3 \times 3$  卷积层，以及一个在通道维度进行上采样的  $1 \times 1$  层。因此，瓶颈卷积单元的卷积层数比标准单元多 50%，而其空间  $3 \times 3$  卷积层数却是标准单元的一半。这两种单元类型的计算复杂度相似，但使用瓶颈单元时，残差连接中传播的特征总数要多四倍。网络的总深度定义为顺序卷积层和全连接层的层数之和。对于由标准单元构成的网络，总深度为  $2 * numUnits + 2$ ，对于由瓶颈单元构成的网络，总深度为  $3 * numUnits + 2$ 。

创建一个包含九个标准卷积单元（每阶段三个单元）且宽度为 16 的残差网络。网络总深度为  $2 * 9 + 2 = 20$ 。

```

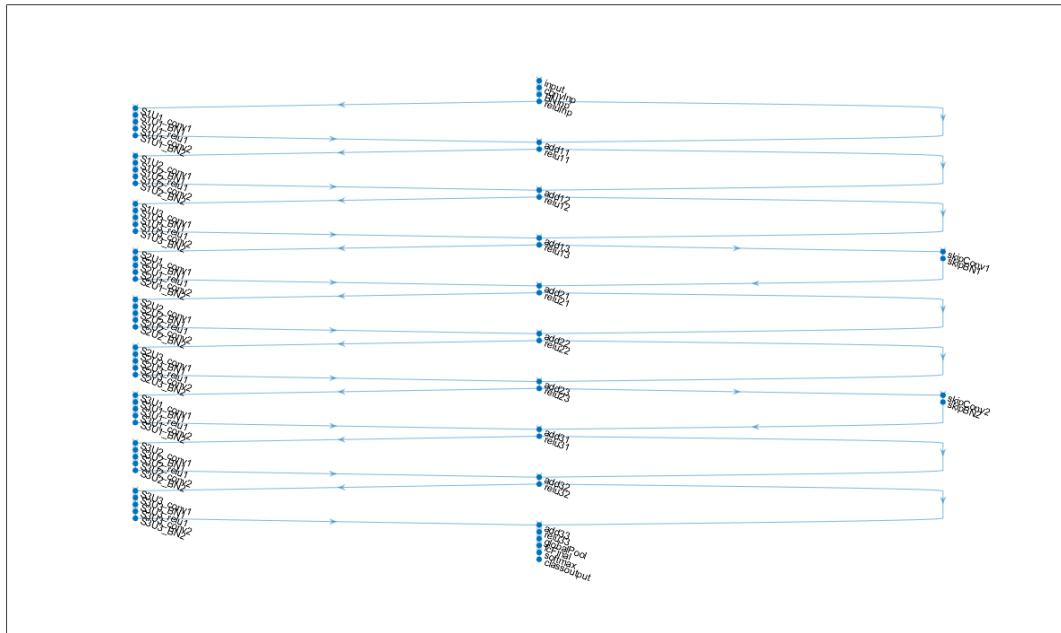
numUnits = 9;
netWidth = 16;

```

```

lgraph = residualCIFARlgraph(netWidth,numUnits,"standard");
figure('Units','normalized','Position',[0.1 0.1 0.8 0.8]);
plot(lgraph)

```



## 训练网络

指定训练选项。对网络进行 80 轮训练。选择与小批量大小成正比的学习率，并在 60 轮训练后将学习率降低十分之一。每轮训练后都使用验证数据验证一次网络。

```

miniBatchSize = 128;
learnRate = 0.1*miniBatchSize/128;
valFrequency = floor(size(XTrain,4)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'InitialLearnRate',learnRate, ...
    'MaxEpochs',80, ...
    'MiniBatchSize',miniBatchSize, ...
    'VerboseFrequency',valFrequency, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose',false, ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',valFrequency, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',60);

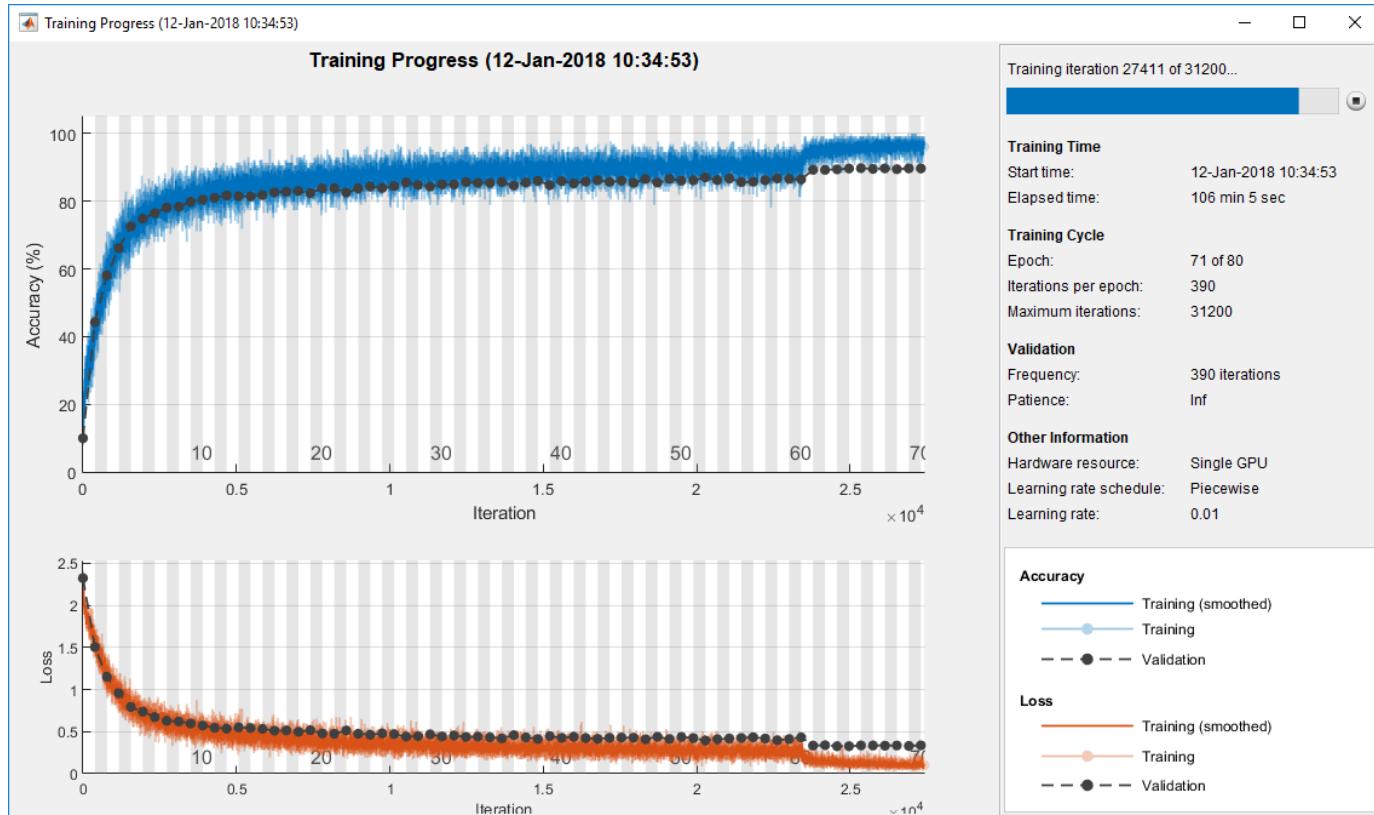
```

要使用 `trainNetwork` 训练网络，请将 `doTraining` 标志设置为 `true`。否则，请加载预训练的网络。在一个较好的 GPU 上训练网络大约需要两小时。如果您没有 GPU，则训练需要更长时间。

```

doTraining = false;
if doTraining
    trainedNet = trainNetwork(augimdsTrain,lgraph,options);
else
    load('CIFARNet-20-16.mat','trainedNet');
end

```



## 评估经过训练的网络

基于训练集（无数据增强）和验证集计算网络的最终准确度。

```

[YValPred,probs] = classify(trainedNet,XValidation);
validationError = mean(YValPred ~= YValidation);
YTrainPred = classify(trainedNet,XTrain);
trainError = mean(YTrainPred ~= YTrain);
disp("Training error:" + trainError*100 + "%")

```

Training error: 2.862%

```
disp("Validation error:" + validationError*100 + "%")
```

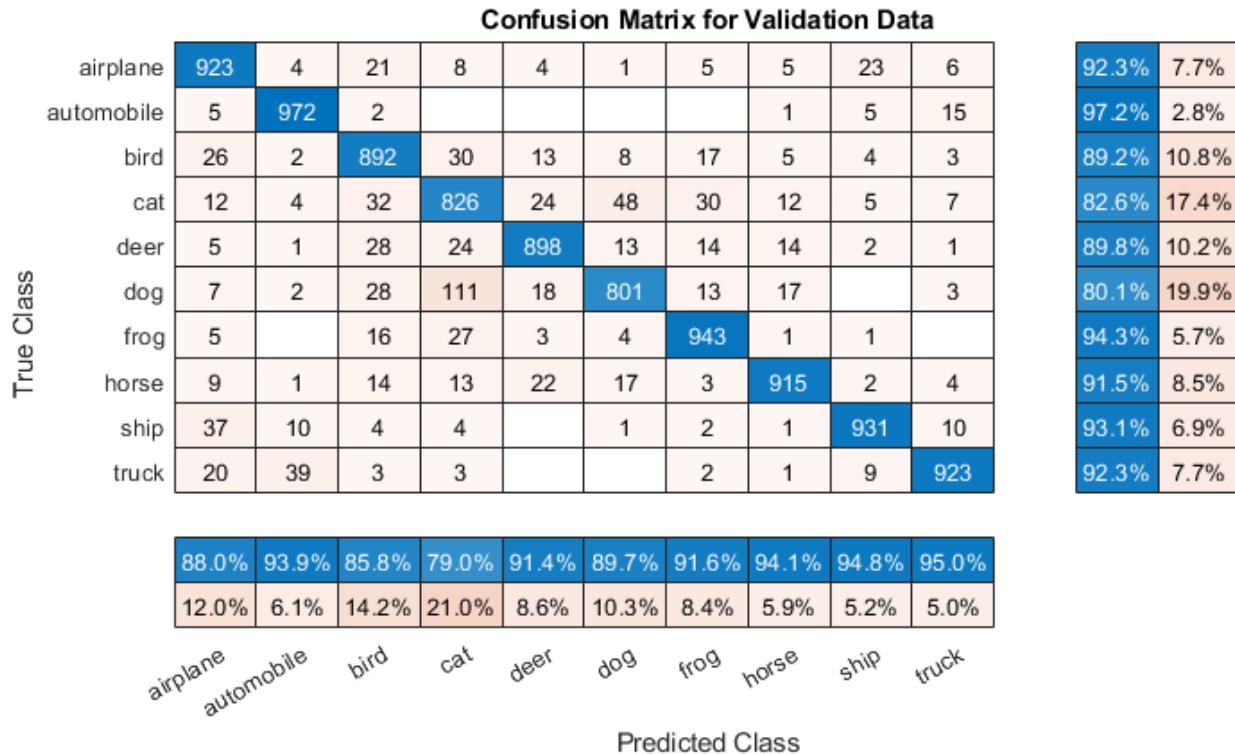
Validation error: 9.76%

绘制混淆矩阵。使用列汇总和行汇总显示每个类的准确率和召回率。网络最常将猫与狗混淆。

```

figure('Units','normalized','Position',[0.2 0.2 0.4 0.4]);
cm = confusionchart(YValidation,YValPred);
cm.Title = 'Confusion Matrix for Validation Data';
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';

```



您可以使用以下代码显示包含九个测试图像的随机样本，以及它们的预测类和这些类的概率。

```
figure
idx = randperm(size(XValidation,4),9);
for i = 1:numel(idx)
    subplot(3,3,i)
    imshow(XValidation(:,:,idx(i)));
    prob = num2str(100*max(probs(idx(i),:)),3);
    predClass = char(YValPred(idx(i)));
    title([predClass,',',prob,'%'])
end
```

**convolutionalUnit(numF,stride,tag)** 创建一个层数组，其中包含两个卷积层以及对应的批量归一化层和 ReLU 层。 **numF** 是卷积滤波器的数量，**stride** 是第一个卷积层的步幅，**tag** 是添加在所有层名称前面的标记。

```
function layers = convolutionalUnit(numF,stride,tag)
layers = [
    convolution2dLayer(3,numF,'Padding','same','Stride',stride,'Name',[tag,'conv1'])
    batchNormalizationLayer('Name',[tag,'BN1'])
    reluLayer('Name',[tag,'relu1'])
    convolution2dLayer(3,numF,'Padding','same','Name',[tag,'conv2'])
    batchNormalizationLayer('Name',[tag,'BN2'])];
end
```

## 参考

- [1] Krizhevsky, Alex. "Learning multiple layers of features from tiny images." (2009). <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>

- 
- [2] He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

## 另请参阅

`trainNetwork` | `trainingOptions` | `layerGraph` | `analyzeNetwork`

## 相关示例

- “使用贝叶斯优化进行深度学习” (第 5-8 页)
- “设置参数并训练卷积神经网络” (第 1-35 页)
- “预训练的深度神经网络” (第 1-8 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 使用 GoogLeNet 对图像进行分类

此示例说明如何使用预训练的深度卷积神经网络 GoogLeNet 对图像进行分类。

GoogLeNet 已经对超过一百万个图像进行了训练，可以将图像分为 1000 个对象类别（例如键盘、咖啡杯、铅笔和多种动物）。该网络已基于大量图像学习了丰富的特征表示。网络以图像作为输入，然后输出图像中对象的标签以及每个对象类别的概率。

### 加载预训练网络

加载预训练的 GoogLeNet 网络。此步骤需要 Deep Learning Toolbox™ Model for GoogLeNet Network 支持包。如果没有安装所需的支持包，软件会提供下载链接。

您还可以选择加载不同的预训练网络进行图像分类。要尝试不同的预训练网络，请在 MATLAB® 中打开此示例并选择其他网络。例如，您可以尝试 `squeezeNet`，这是一个比 `googlenet` 还要快的网络。您可以使用户其他预训练网络运行此示例。有关所有可用网络的列表，请参阅“加载预训练网络”（第 1-9 页）。

```
net = googlenet;
```

要分类的图像的大小必须与网络的输入大小相同。对于 GoogLeNet，网络的 `Layers` 属性的第一个元素是图像输入层。网络输入大小是图像输入层的 `InputSize` 属性。

```
inputSize = net.Layers(1).InputSize
```

```
inputSize = 1×3
```

```
224 224 3
```

`Layers` 属性的最后一个元素是分类输出层。该层的 `ClassNames` 属性包含网络学习的类的名称。查看总共 1000 个类名称中的 10 个随机类名称。

```
classNames = net.Layers(end).ClassNames;
numClasses = numel(classNames);
disp(classNames(randperm(numClasses,10)))
```

```
'papillon'  
'eggnog'  
'jackfruit'  
'castle'  
'sleeping bag'  
'redshank'  
'Band Aid'  
'wok'  
'seat belt'  
'orange'
```

### 读取图像并调整图像大小

读取并显示要分类的图像。

```
I = imread('peppers.png');
figure
imshow(I)
```



显示图像的大小。图像为 384×512 像素，并且具有三个颜色通道 (RGB)。

```
size(I)  
ans = 1×3  
384 512 3
```

使用 `imresize` 将图像大小调整为网络的输入大小。调整大小会略微更改图像的纵横比。

```
I = imresize(I,inputSize(1:2));  
figure  
imshow(I)
```



根据您的应用，您可能希望以不同方式调整图像大小。例如，您可以使用 `I(1:inputSize(1),1:inputSize(2),:)` 剪去图像的左上角。如果您有 Image Processing Toolbox™，则可以使用 `imcrop` 函数。

#### 对图像进行分类

使用 `classify` 对图像进行分类并计算类概率。网络正确地将图像分类为甜椒。用于分类的网络训练为针对每个输入图像输出单个标签，即使图像包含多个对象时也是如此。

```
[label,scores] = classify(net,I);
label

label = categorical
    bell pepper
```

显示图像及预测的标签，以及具有该标签的图像的预测概率。

```
figure
imshow(I)
title(string(label) + "," + num2str(100*scores(classNames == label),3) + "%");
```

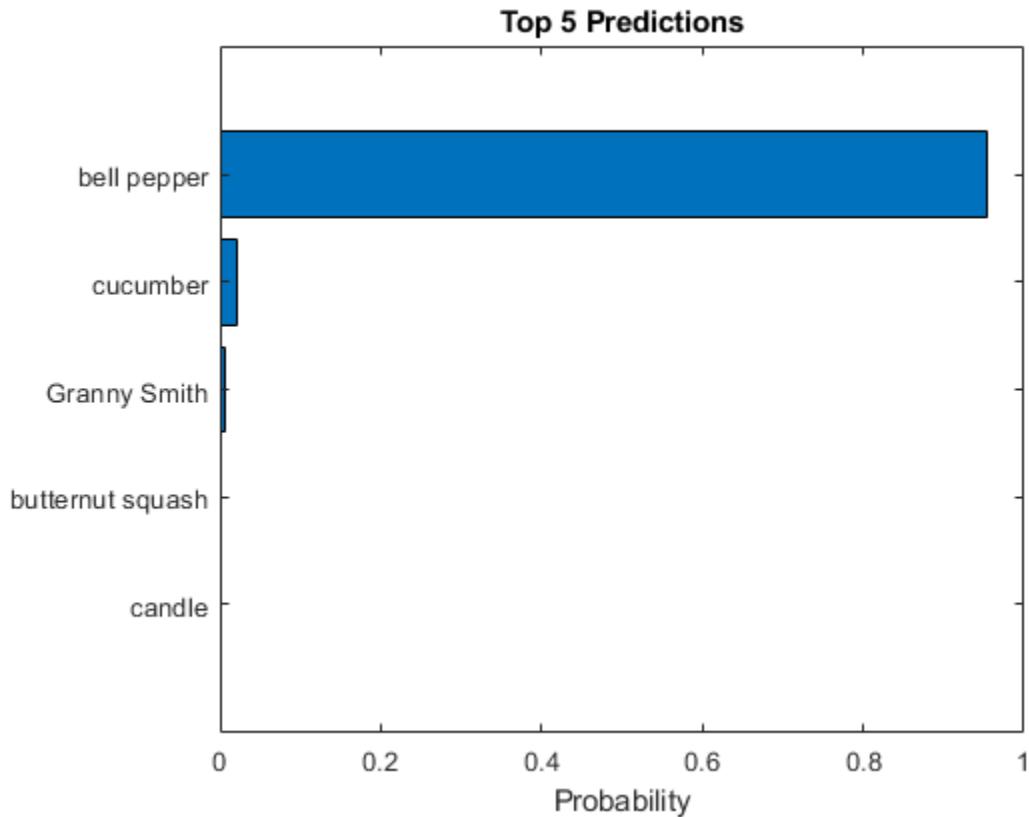


### 显示排名靠前的预测值

显示排名前五的预测标签，并以直方图形式显示它们的相关概率。由于网络将图像分类为如此多的对象类别，并且许多类别是相似的，因此在评估网络时通常会考虑准确度排名前五的几个类别。网络以高概率将图像分类为甜椒。

```
[~,idx] = sort(scores,'descend');
idx = idx(5:-1:1);
classNamesTop = net.Layers(end).ClassNames(idx);
scoresTop = scores(idx);

figure
barh(scoresTop)
xlim([0 1])
title('Top 5 Predictions')
xlabel('Probability')
yticklabels(classNamesTop)
```



## 参考

- [1] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9. 2015.
- [2] *BVLC GoogLeNet Model*. [https://github.com/BVLC/caffe/tree/master/models/bvlc\\_googlenet](https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet)

## 另请参阅

DAGNetwork | googlenet | classify | predict | squeezenet

## 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “预训练的深度神经网络” (第 1-8 页)
- “训练深度学习网络以对新图像进行分类” (第 3-6 页)

# 使用预训练网络提取图像特征

此示例说明如何从预训练的卷积神经网络中提取已学习的图像特征，并使用这些特征来训练图像分类器。特征提取是使用预训练深度网络的表征能力的最简单最快捷的方式。例如，您可以使用 `fitcecoc`(Statistics and Machine Learning Toolbox™) 基于提取的特征来训练支持向量机 (SVM)。由于特征提取只需要遍历一次数据，因此如果没有 GPU 来加速网络训练，则不妨从特征提取开始。

## 加载数据

解压缩示例图像并加载这些图像作为图像数据存储。`imageDatastore` 根据文件夹名称自动标注图像，并将数据存储为 `ImageDatastore` 对象。通过图像数据存储可以存储大图像数据，包括无法放入内存的数据。将数据拆分，其中 70% 用作训练数据，30% 用作测试数据。

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData','IncludeSubfolders',true,'LabelSource','foldernames');
[imdsTrain,imdsTest] = splitEachLabel(imds,0.7,'randomized');
```

在这个非常小的数据集中，现在有 55 个训练图像和 20 个验证图像。显示一些示例图像。

```
numTrainImages = numel(imdsTrain.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(imdsTrain,idx(i));
    imshow(I)
end
```



### 加载预训练网络

加载预训练的 ResNet-18 网络。如果未安装 Deep Learning Toolbox Model for ResNet-18 Network 支持包，则软件会提供下载链接。ResNet-18 已基于超过一百万个图像进行训练，可以将图像分为 1000 个对象类别（例如键盘、鼠标、铅笔和多种动物）。因此，该模型已基于大量图像学习了丰富的特征表示。

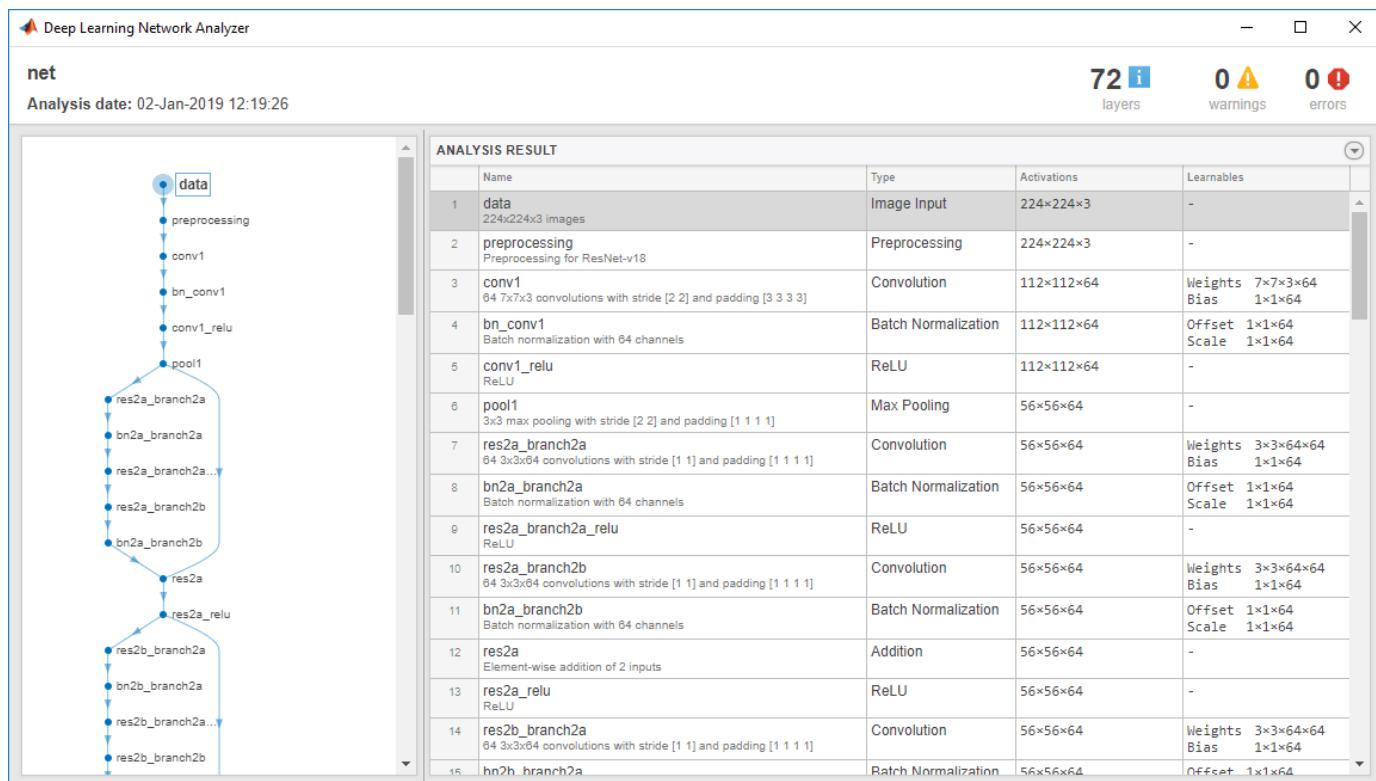
```
net = resnet18

net =
DAGNetwork with properties:

    Layers: [71x1 nnet.cnn.layer.Layer]
    Connections: [78x2 table]
    InputNames: {'data'}
    OutputNames: {'ClassificationLayer_predictions'}
```

分析网络架构。第一层（图像输入层）需要大小为  $224 \times 224 \times 3$  的输入图像，其中 3 是颜色通道数。

```
inputSize = net.Layers(1).InputSize;
analyzeNetwork(net)
```



### 提取图像特征

网络要求输入图像的大小为  $224 \times 224 \times 3$ ，但图像数据存储中的图像具有不同大小。要在将训练图像和测试图像输入到网络之前自动调整它们的大小，请创建增强的图像数据存储，指定所需的图像大小，并将这些数据存储用作 activations 的输入参数。

```
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain);
augimdsTest = augmentedImageDatastore(inputSize(1:2),imdsTest);
```

网络构造输入图像的分层表示。更深层包含更高级别的特征，这些特征使用较浅层的较低级别特征构建。要获得训练图像和测试图像的特征表示，请对网络末尾的全局池化层 'pool5'，使用 `activations`。全局池化层汇集所有空间位置的输入特征，总共提供 512 个特征。

```
layer = 'pool5';
featuresTrain = activations(net,augimdsTrain,layer,'OutputAs','rows');
featuresTest = activations(net,augimdsTest,layer,'OutputAs','rows');
```

**whos featuresTrain**

Name	Size	Bytes	Class	Attributes
featuresTrain	55x512	112640	single	

从训练数据和测试数据中提取类标签。

```
YTrain = imdsTrain.Labels;
YTest = imdsTest.Labels;
```

### 拟合图像分类器

使用从训练图像中提取的特征作为预测变量，并使用 `fitcecoc` (Statistics and Machine Learning Toolbox) 拟合多类支持向量机 (SVM)。

```
classifier = fitcecoc(featuresTrain,YTrain);
```

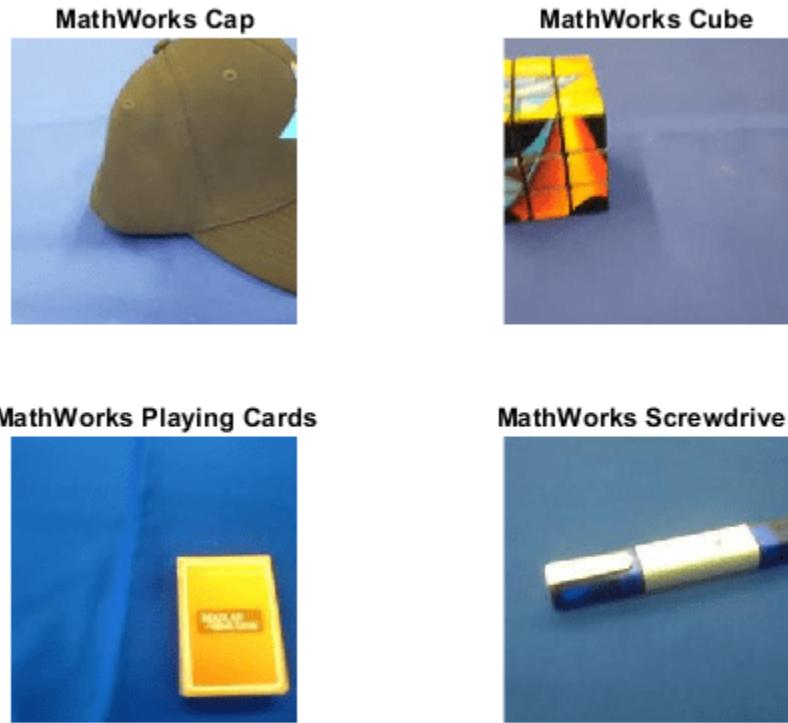
### 对测试图像进行分类

使用经过训练的 SVM 模型和从测试图像中提取的特征对测试图像进行分类。

```
YPred = predict(classifier,featuresTest);
```

显示四个示例测试图像及预测的标签。

```
idx = [1 5 10 15];
figure
for i = 1:numel(idx)
    subplot(2,2,i)
    I = readimage(imdsTest,idx(i));
    label = YPred(idx(i));
    imshow(I)
    title(char(label))
end
```



计算针对测试集的分类准确度。准确度是网络预测正确的标签的比例。

```
accuracy = mean(YPred == YTest)
```

```
accuracy = 1
```

### 基于较浅特征训练分类器

您还可以从网络的较浅层提取特征，并基于这些特征训练分类器。较浅的层通常具有较高的空间分辨率和较大的激活总数，提取的特征也较少、较浅。从 'res3b\_relu' 层中提取特征。这是输出 128 个特征的最顶层，激活的空间大小为  $28 \times 28$ 。

```
layer = 'res3b_relu';
featuresTrain = activations(net,augimdsTrain,layer);
featuresTest = activations(net,augimdsTest,layer);
whos featuresTrain
```

Name	Size	Bytes	Class	Attributes
featuresTrain	$28 \times 28 \times 128 \times 55$	22077440	single	

此示例第一部分中使用的提取特征是从全局池化层的所有空间位置汇集而来的。要在从较浅层中提取特征时获得相同的结果，请手动对所有空间位置的激活区域求平均。要获得  $N \times C$  形式的特征，其中  $N$  是观测值数目， $C$  是特征数量，请删除单一维度并转置。

```
featuresTrain = squeeze(mean(featuresTrain,[1 2]))';
featuresTest = squeeze(mean(featuresTest,[1 2]))';
whos featuresTrain
```

```
Name          Size        Bytes Class Attributes
featuresTrain  55x128      28160  single
```

基于较浅特征训练 SVM 分类器。计算测试准确度。

```
classifier = fitcecoc(featuresTrain,YTrain);
YPred = predict(classifier,featuresTest);
accuracy = mean(YPred == YTest)

accuracy = 0.9500
```

两个经过训练的 SVM 都具有高准确度。如果使用特征提取时的准确度不够高，则尝试迁移学习。有关示例，请参阅“训练深度学习网络以对新图像进行分类”（第 3-6 页）。有关预训练网络的列表和比较，请参阅“预训练的深度神经网络”（第 1-8 页）。

## 另请参阅

[fitcecoc](#) | [resnet50](#)

## 相关示例

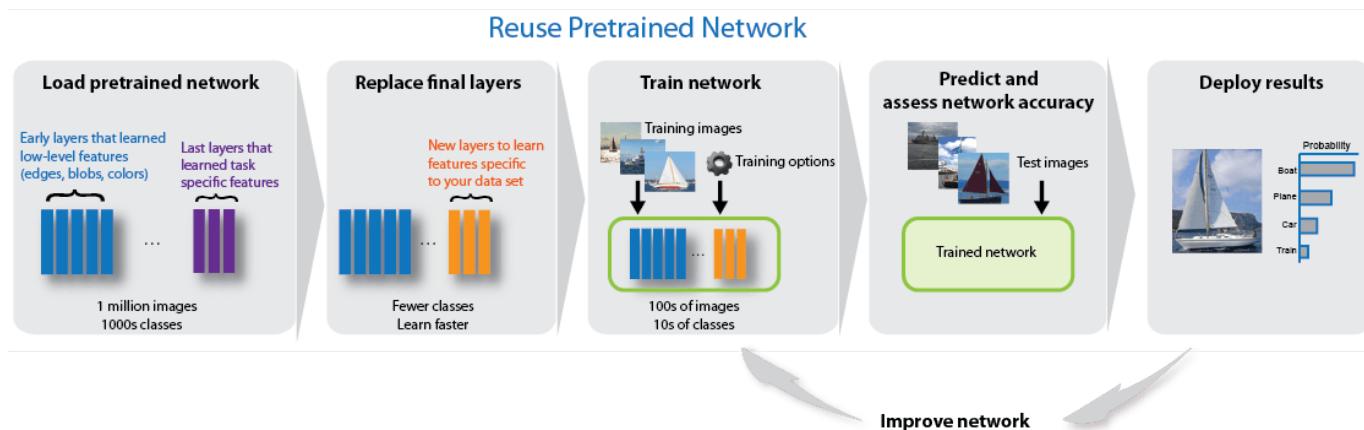
- “训练深度学习网络以对新图像进行分类”（第 3-6 页）
- “预训练的深度神经网络”（第 1-8 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）

## 使用 AlexNet 进行迁移学习

此示例说明如何微调预训练的 AlexNet 卷积神经网络以对新的图像集合执行分类。

AlexNet 已基于超过一百万个图像进行训练，可以将图像分为 1000 个对象类别（例如键盘、咖啡杯、铅笔和多种动物）。该网络已基于大量图像学习了丰富的特征表示。网络以图像作为输入，然后输出图像中对象的标签以及每个对象类别的概率。

深度学习应用中常常用到迁移学习。您可以采用预训练的网络，基于它学习新任务。与使用随机初始化的权重从头训练网络相比，通过迁移学习微调网络要更快更简单。您可以使用较少数量的训练图像快速地将已学习的特征迁移到新任务。



### 加载数据

解压缩新图像并加载这些图像作为图像数据存储。 `imageDatastore` 根据文件夹名称自动标注图像，并将数据存储为 `Image datastore` 对象。通过图像数据存储可以存储大图像数据，包括无法放入内存的数据，并在卷积神经网络的训练过程中高效分批读取图像。

```
unzip('MerchData.zip');
imds = imageDatastore('MerchData',...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

将数据划分为训练数据集和验证数据集。将 70% 的图像用于训练，30% 的图像用于验证。`splitEachLabel` 将 `images` 数据存储拆分为两个新的数据存储。

```
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');
```

这个非常小的数据集现在包含 55 个训练图像和 20 个验证图像。显示一些示例图像。

```
numTrainImages = numel(imdsTrain.Labels);
idx = randperm(numTrainImages,16);
figure
for i = 1:16
    subplot(4,4,i)
    I = readimage(imdsTrain,idx(i));
    imshow(I)
end
```



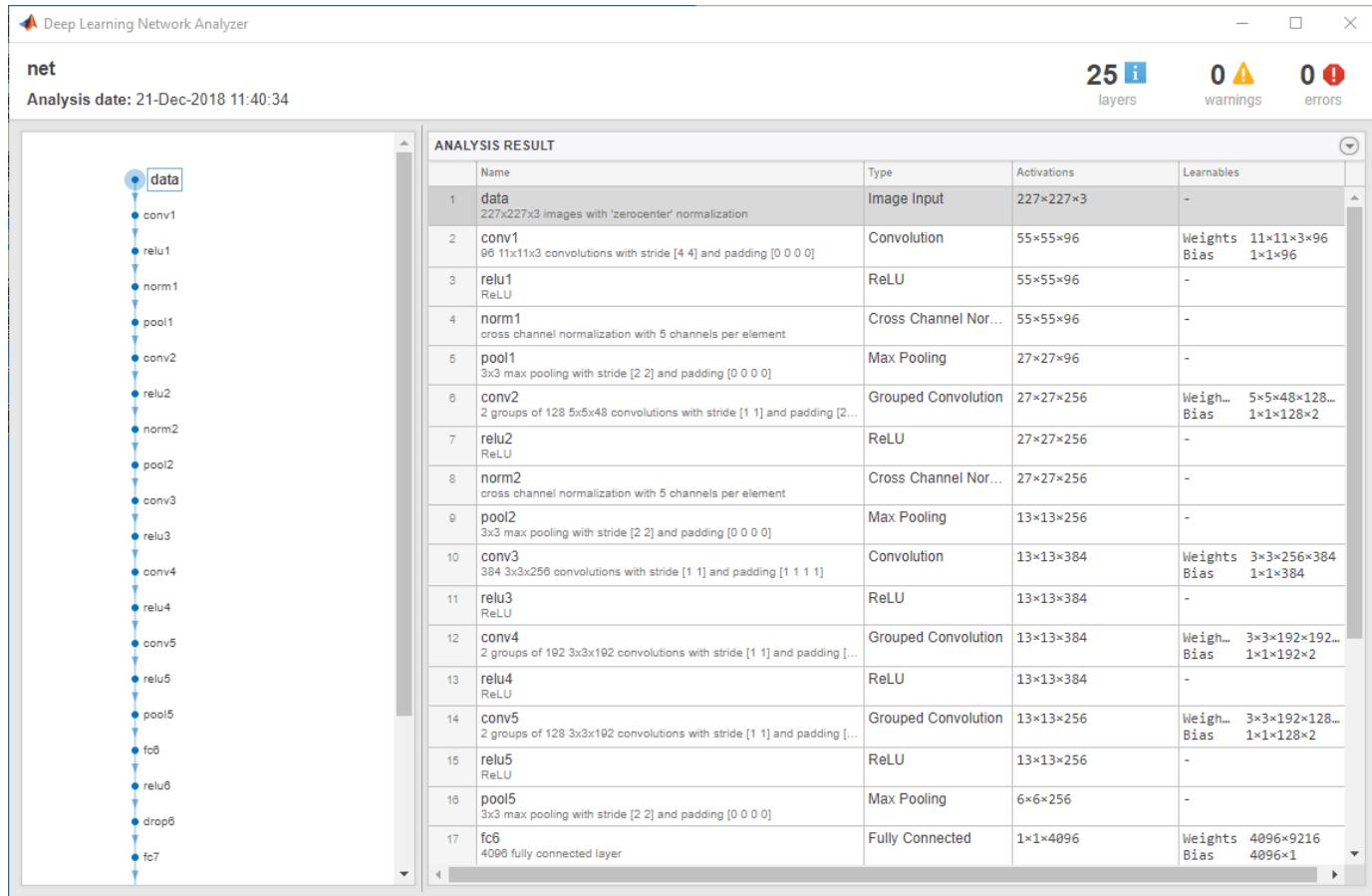
### 加载预训练网络

加载预训练的 AlexNet 神经网络。如果未安装 Deep Learning Toolbox™ Model for AlexNet Network，则软件会提供下载链接。AlexNet 已基于超过一百万个图像进行训练，可以将图像分为 1000 个对象类别（例如键盘、鼠标、铅笔和多种动物）。因此，该模型已基于大量图像学习了丰富的特征表示。

```
net = alexnet;
```

使用 **analyzeNetwork** 可以交互可视方式呈现网络架构以及有关网络层的详细信息。

```
analyzeNetwork(net)
```



第一层（图像输入层）需要大小为  $227 \times 227 \times 3$  的输入图像，其中 3 是颜色通道数。

```
inputSize = net.Layers(1).InputSize
```

```
inputSize = 1×3
```

```
227 227 3
```

### 替换最终层

预训练网络 **net** 的最后三层针对 1000 个类进行配置。必须针对新分类问题微调这三个层。从预训练网络中提取除最后三层之外的所有层。

```
layersTransfer = net.Layers(1:end-3);
```

通过将最后三层替换为全连接层、softmax 层和分类输出层，将层迁移到新分类任务。根据新数据指定新的全连接层的选项。将全连接层设置为大小与新数据中的类数相同。要使新层中的学习速度快于迁移的层，请增大全连接层的 **WeightLearnRateFactor** 和 **BiasLearnRateFactor** 值。

```
numClasses = numel(categories(imdsTrain.Labels))
```

```
numClasses = 5
```

```
layers = [  
    layersTransfer
```

```
fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)
softmaxLayer
classificationLayer];
```

## 训练网络

网络要求输入图像的大小为  $227 \times 227 \times 3$ ，但图像数据存储中的图像具有不同大小。使用增强的图像数据存储可自动调整训练图像的大小。指定要对训练图像额外执行的增强操作：沿垂直轴随机翻转训练图像，以及在水平和垂直方向上随机平移训练图像最多 30 个像素。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
pixelRange = [-30 30];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...
    'DataAugmentation',imageAugmenter);
```

要在不执行进一步数据增强的情况下自动调整验证图像的大小，请使用增强的图像数据存储，而不指定任何其他预处理操作。

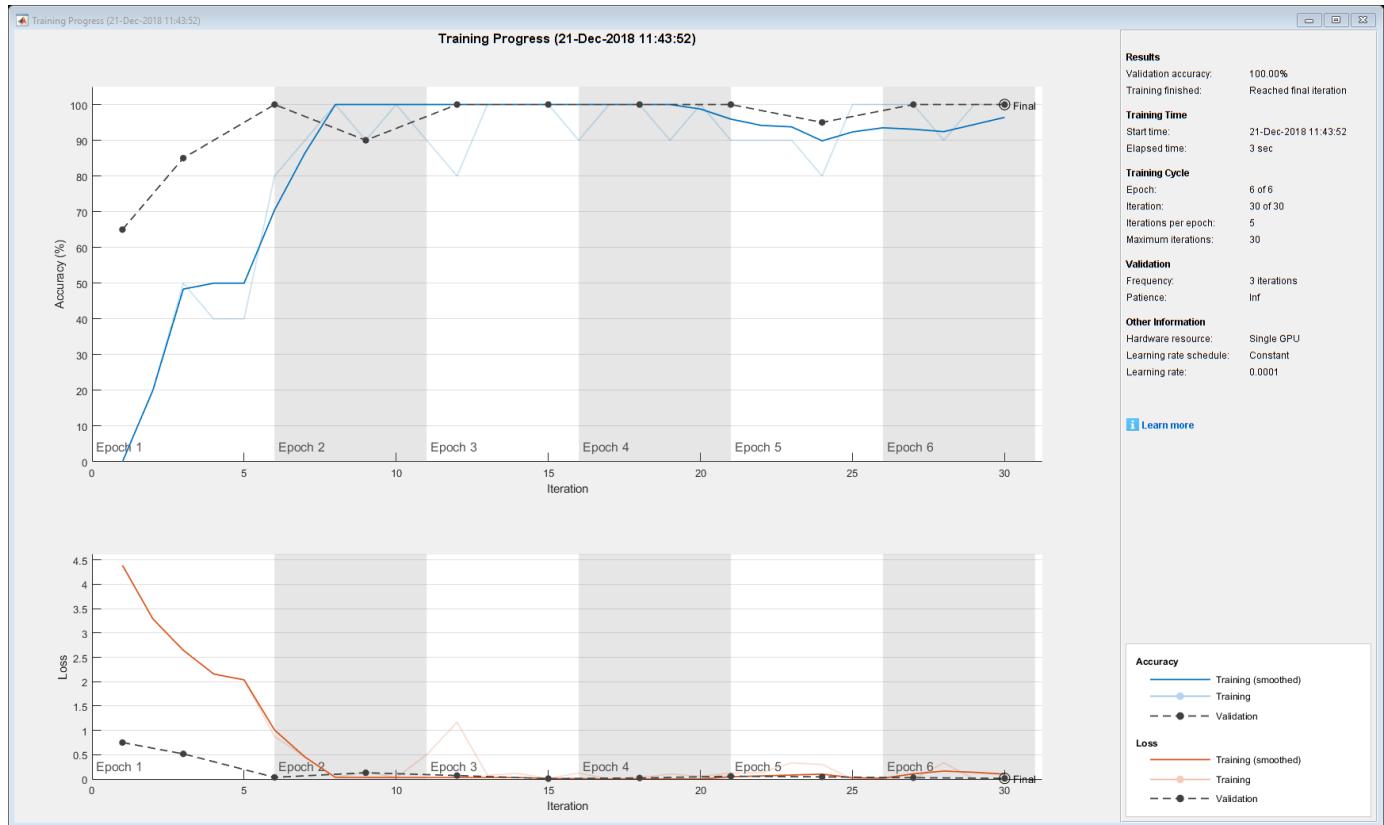
```
augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);
```

指定训练选项。对于迁移学习，请保留预训练网络的较浅层中的特征（迁移的层权重）。要减慢迁移的层中的学习速度，请将初始学习速率设置为较小的值。在上一步中，您增大了全连接层的学习率因子，以加快新的最终层中的学习速度。这种学习率设置组合只会加快新层中的学习速度，对于其他层则会减慢学习速度。执行迁移学习时，所需的训练轮数相对较少。一轮训练是对整个训练数据集的一个完整训练周期。指定小批量大小和验证数据。软件在训练过程中每 **ValidationFrequency** 次迭代验证一次网络。

```
options = trainingOptions('sgdm', ...
    'MiniBatchSize',10, ...
    'MaxEpochs',6, ...
    'InitialLearnRate',1e-4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',augimdsValidation, ...
    'ValidationFrequency',3, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

训练由迁移层和新层组成的网络。默认情况下，**trainNetwork** 使用 GPU（如果有），否则使用 CPU。在 GPU 上训练需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。您还可以使用 **trainingOptions** 的**'ExecutionEnvironment'** 名称-值对组参数指定执行环境。

```
netTransfer = trainNetwork(augimdsTrain,layers,options);
```



### 对验证图像进行分类

使用经过微调的网络对验证图像进行分类。

```
[YPred,scores] = classify(netTransfer,augimdsValidation);
```

显示四个示例验证图像及预测的标签。

```
idx = randperm(numel(imdsValidation.Files),4);
figure
for i = 1:4
    subplot(2,2,i)
    I = readimage(imdsValidation,idx(i));
    imshow(I)
    label = YPred(idx(i));
    title(string(label));
end
```

**MathWorks Playing Cards****MathWorks Screwdriver****MathWorks Cap****MathWorks Screwdriver**

计算针对验证集的分类准确度。准确度是网络预测正确的标签的比例。

```
YValidation = imdsValidation.Labels;
accuracy = mean(YPred == YValidation)

accuracy = 1
```

有关提高分类准确度的提示，请参阅“Deep Learning Tips and Tricks”。

## 参考

- [1] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." *Advances in neural information processing systems*. 2012.
- [2] *BVLC AlexNet Model*. [https://github.com/BVLC/caffe/tree/master/models/bvlc\\_alexnet](https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet)

## 另请参阅

**trainNetwork | trainingOptions | alexnet | analyzeNetwork**

## 相关示例

- “了解卷积神经网络”（第 1-16 页）
- “设置参数并训练卷积神经网络”（第 1-35 页）

- “使用预训练网络提取图像特征”（第 3-27 页）
- “预训练的深度神经网络”（第 1-8 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）

# 创建简单的深度学习网络以用于分类

此示例说明如何创建和训练简单的卷积神经网络来进行深度学习分类。卷积神经网络是深度学习的基本工具，尤其适用于图像识别。

该示例演示如何：

- 加载和浏览图像数据。
- 定义网络架构。
- 指定训练选项。
- 训练网络。
- 预测新数据的标签并计算分类准确度。

有关如何以交互方式创建和训练简单图像分类网络的示例，请参阅“使用深度网络设计器创建简单的图像分类网络”。

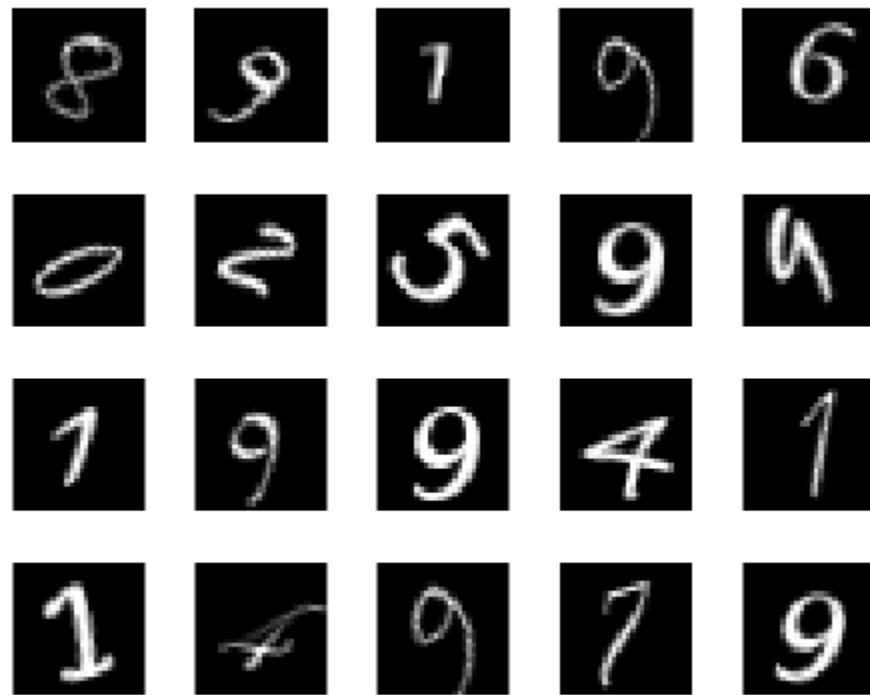
## 加载和浏览图像数据

加载数字样本数据作为图像数据存储。`imageDatastore` 根据文件夹名称自动标注图像，并将数据存储为 `ImageDatastore` 对象。通过图像数据存储可以存储大图像数据，包括无法放入内存的数据，并在卷积神经网络的训练过程中高效分批读取图像。

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos',...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true,'LabelSource','foldernames');
```

显示数据存储中的部分图像。

```
figure;
perm = randperm(10000,20);
for i = 1:20
    subplot(4,5,i);
    imshow(imds.Files{perm(i)});
```



计算每个类别中的图像数量。`labelCount` 是一个表，其中列出了标签，以及每个标签对应的图像数量。数据存储包含数字 0-9 的总共 10000 个图像，每个数字对应 1000 个图像。您可以在网络的最后一个全连接层中指定类数作为 `OutputSize` 参数。

```
labelCount = countEachLabel(imds)
```

```
labelCount=10×2 table
  Label Count
```

Label	Count
0	1000
1	1000
2	1000
3	1000
4	1000
5	1000
6	1000
7	1000
8	1000
9	1000

您必须在网络的输入层中指定图像的大小。检查 `digitData` 中第一个图像的大小。每个图像的大小均为  $28 \times 28 \times 1$  像素。

```
img = readimage(imds,1);
size(img)
```

```
ans = 1×2
```

```
28 28
```

## 指定训练集和验证集

将数据划分为训练数据集和验证数据集，以使训练集中的每个类别包含 750 个图像，并且验证集包含对应每个标签的其余图像。`splitEachLabel` 将数据存储 `digitData` 拆分为两个新的数据存储 `trainDigitData` 和 `valDigitData`。

```
numTrainFiles = 750;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

## 定义网络架构

定义卷积神经网络架构。

```
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

**图像输入层** `imageInputLayer` 用于指定图像大小，在本例中为  $28 \times 28 \times 1$ 。这些数字对应于高度、宽度和通道大小。数字数据由灰度图像组成，因此通道大小（颜色通道）为 1。对于彩色图像，通道大小为 3，对应于 RGB 值。您不需要打乱数据，因为 `trainNetwork` 默认会在训练开始时打乱数据。`trainNetwork` 还可以在训练过程中的每轮训练开始时自动打乱数据。

**卷积层** 在卷积层中，第一个参数是 `filterSize`，它是训练函数在沿图像扫描时使用的滤波器的高度和宽度。在此示例中，数字 3 表示滤波器大小为  $3 \times 3$ 。您可以为滤波器的高度和宽度指定不同大小。第二个参数是滤波器数量 `numFilters`，它是连接到同一输入区域的神经元数量。此参数决定了特征图的数量。使用 `'Padding'` 名称-值对组对输入特征图进行填充。对于默认步幅为 1 的卷积层，`'same'` 填充可确保空间输出大小与输入大小相同。您也可以使用 `convolution2dLayer` 名称-值对组参数定义该层的步幅和学习率。

**批量归一化层** 批量归一化层对网络中的激活值和梯度传播进行归一化，使网络训练成为更简单的优化问题。在卷积层和非线性部分（例如 ReLU 层）之间使用批量归一化层，来加速网络训练并降低对网络初始化的敏感度。使用 `batchNormalizationLayer` 创建批量归一化层。

**ReLU 层**批量归一化层后接一个非线性激活函数。最常见的激活函数是修正线性单元 (ReLU)。使用 `reluLayer` 创建 ReLU 层。

**最大池化层**卷积层（带激活函数）有时会后跟下采样操作，以减小特征图的空间大小并删除冗余空间信息。通过下采样可以增加更深卷积层中的滤波器数量，而不会增加每层所需的计算量。下采样的一种方法是使用最大池化，可使用 `maxPooling2dLayer` 创建。最大池化层返回由第一个参数 `poolSize` 指定的矩形输入区域的最大值。在此示例中，该矩形区域的大小是 [2,2]。'Stride' 名称-值对组参数指定训练函数在沿输入扫描时所采用的步长。

**全连接层**卷积层和下采样层后跟一个或多个全连接层。顾名思义，全连接层中的神经元将连接到前一层中的所有神经元。该层将先前层在图像中学习的所有特征组合在一起，以识别较大的模式。最后一个全连接层将特征组合在一起对图像进行分类。因此，最后一个全连接层中的 `OutputSize` 参数等于目标数据中的类数。在此示例中，输出大小为 10，对应于 10 个类。使用 `fullyConnectedLayer` 创建全连接层。

**softmax 层** softmax 激活函数对全连接层的输出进行归一化。softmax 层的输出由总和为 1 的多个正数组成，这些数字随后可被分类层用作分类概率。使用 `softmaxLayer` 函数在最后一个全连接层后创建一个 softmax 层。

**分类层**最终层是分类层。该层使用 softmax 激活函数针对每个输入返回的概率，将输入分配到其中一个互斥类并计算损失。要创建分类层，请使用 `classificationLayer`。

### 指定训练选项

定义网络结构体后，指定训练选项。使用具有动量的随机梯度下降 (SGDM) 训练网络，初始学习率为 0.01。将最大训练轮数设置为 4。一轮训练是对整个训练数据集的一个完整训练周期。通过指定验证数据和验证频率，监控训练过程中的网络准确度。每轮训练都会打乱数据。软件基于训练数据训练网络，并在训练过程中按固定时间间隔计算基于验证数据的准确度。验证数据不用于更新网络权重。打开训练进度图，关闭命令行窗口输出。

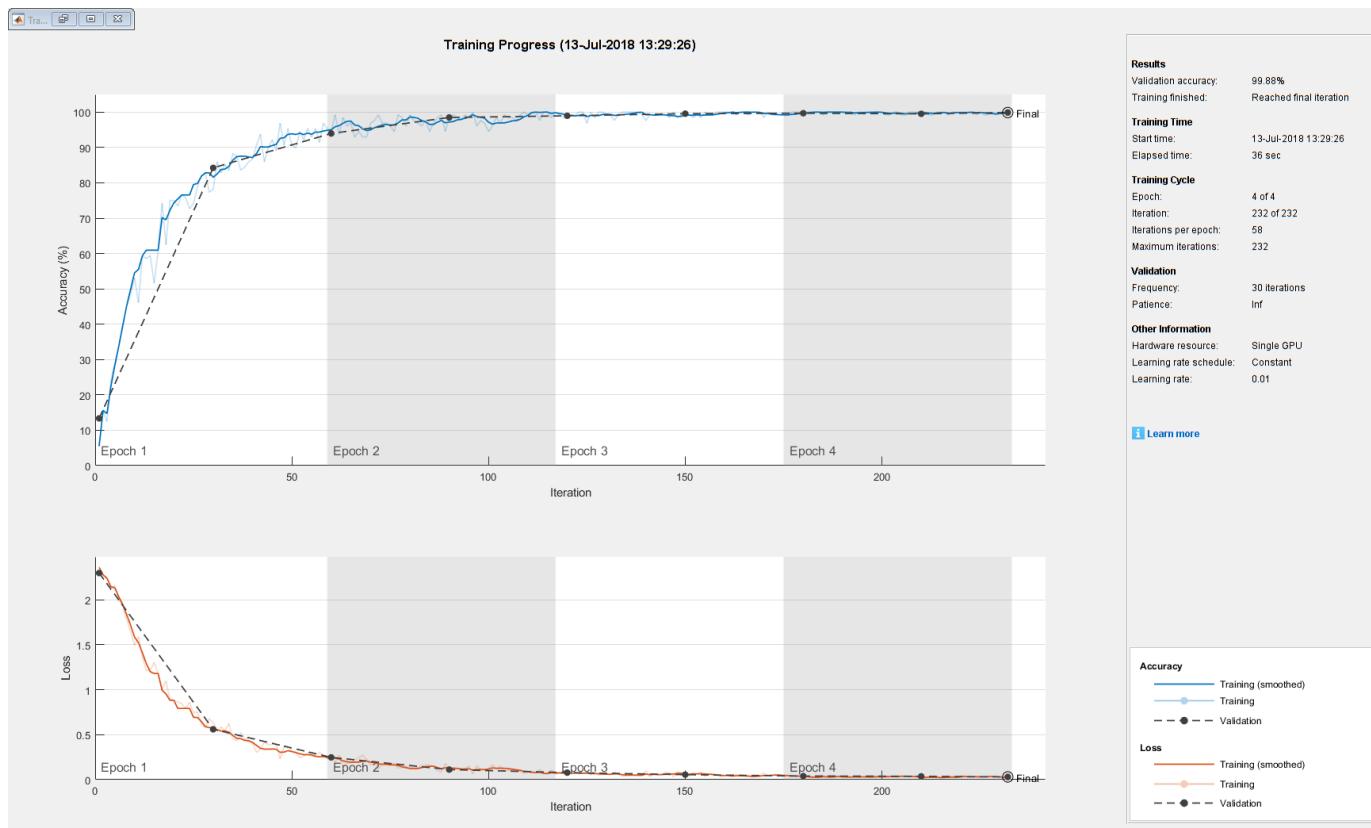
```
options = trainingOptions('sgdm',...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',4, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

### 使用训练数据训练网络

使用 `layers` 定义的架构、训练数据和训练选项训练网络。默认情况下，`trainNetwork` 使用 GPU（如果有），否则使用 CPU。在 GPU 上训练需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release” (Parallel Computing Toolbox)。您还可以使用 `trainingOptions` 的 'ExecutionEnvironment' 名称-值对组参数指定执行环境。

训练进度图显示了小批量损失和准确度以及验证损失和准确度。有关训练进度图的详细信息，请参阅“监控深度学习训练进度”（第 5-22 页）。损失是交叉熵损失。准确度是网络分类正确的图像的百分比。

```
net = trainNetwork(imdsTrain,layers,options);
```



### 对验证图像进行分类并计算准确度

使用经过训练的网络预测验证数据的标签，并计算最终验证准确度。准确度是网络预测正确的标签的比例。在本例中，超过 99% 的预测标签与验证集的真实标签相匹配。

```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;

accuracy = sum(YPred == YValidation)/numel(YValidation)

accuracy = 0.9988
```

### 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [analyzeNetwork](#) | [深度网络设计器](#)

### 相关示例

- “了解卷积神经网络”（第 1-16 页）
- “指定卷积神经网络的层”（第 1-25 页）
- “设置参数并训练卷积神经网络”（第 1-35 页）
- “预训练的深度神经网络”（第 1-8 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）
- 深度学习入门之旅

## 针对回归训练卷积神经网络

此示例说明如何使用卷积神经网络拟合回归模型来预测手写数字的旋转角度。

卷积神经网络 (CNN 或 ConvNet) 是深度学习的基本工具，尤其适用于分析图像数据。例如，您可以用 CNN 对图像进行分类。要预测连续数据（例如角度和距离），可以在网络末尾包含回归层。

该示例构造一个卷积神经网络架构，训练网络，并使用经过训练的网络预测手写数字的旋转角度。这些预测对于光学字符识别很有用。

此外，您可以选择使用 `imrotate` (Image Processing Toolbox™) 旋转图像，并可选择使用 `boxplot` (Statistics and Machine Learning Toolbox™) 创建残差箱线图。

### 加载数据

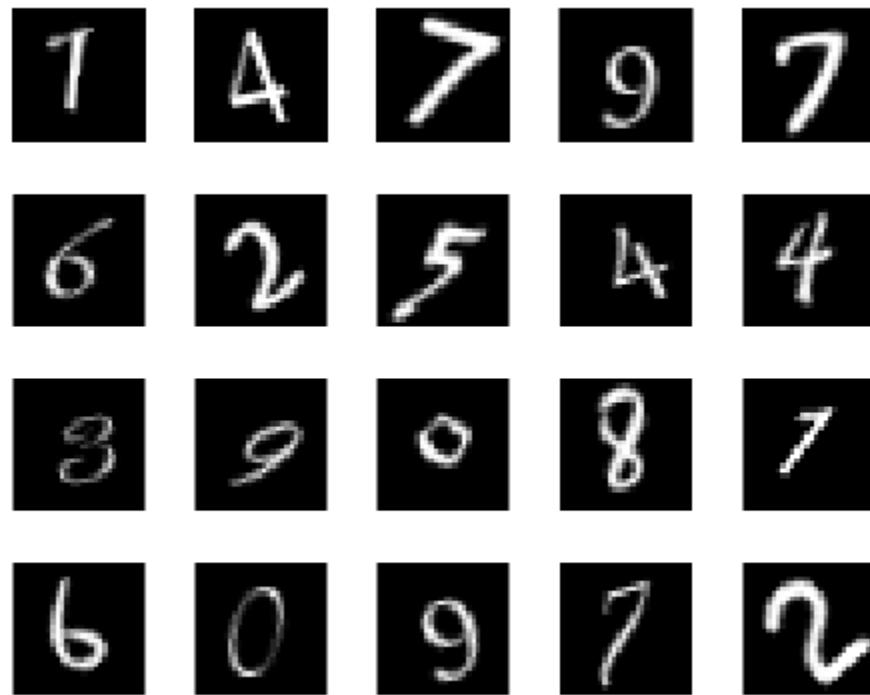
数据集包含手写数字的合成图像以及每个图像的旋转角度（以度为单位）。

使用 `digitTrain4DArrayData` 和 `digitTest4DArrayData` 以四维数组的形式加载训练图像和验证图像。输出 `YTrain` 和 `YValidation` 是以度为单位的旋转角度。训练数据集和验证数据集各包含 5000 个图像。

```
[XTrain,~,YTrain] = digitTrain4DArrayData;
[XValidation,~,YValidation] = digitTest4DArrayData;
```

使用 `imshow` 显示 20 个随机训练图像。

```
numTrainImages = numel(YTrain);
figure
idx = randperm(numTrainImages,20);
for i = 1:numel(idx)
    subplot(4,5,i)
    imshow(XTrain(:,:,:,:idx(i)))
end
```



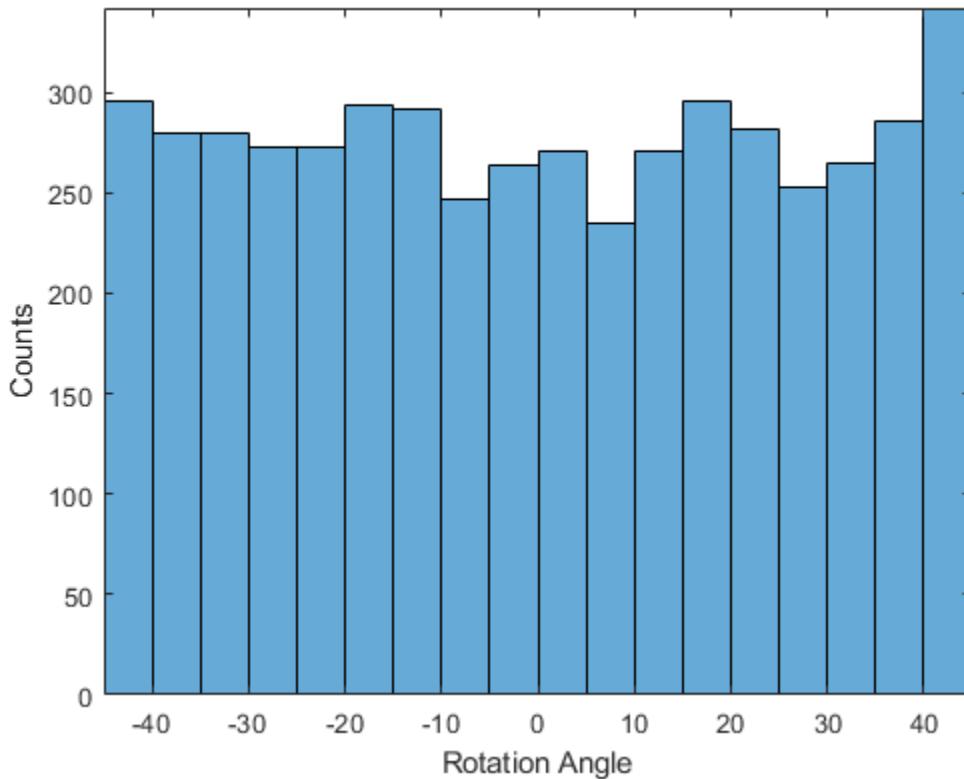
### 检查数据归一化

在训练神经网络时，最好确保数据在网络的所有阶段均归一化。对于使用梯度下降的网络训练，归一化有助于训练的稳定和加速。如果您的数据比例不佳，则损失可能会变为 NaN，并且网络参数在训练过程中可能发生偏离。归一化数据的常用方法包括重新缩放数据，使其范围变为 [0,1]，或使其均值为 0 且标准差为 1。您可以归一化以下数据：

- 输入数据。在将预测变量输入到网络之前对其进行归一化。在此示例中，输入图像已归一化到范围 [0,1]。
- 层输出。您可以使用批量归一化层来归一化每个卷积层和全连接层的输出。
- 响应。如果使用批量归一化层来归一化网络末尾的层输出，则网络的预测值在训练开始时就被归一化。如果响应的比例与这些预测值完全不同，则网络训练可能无法收敛。如果您的响应比例不佳，则尝试对其进行归一化，并查看网络训练是否有所改善。如果在训练之前将响应归一化，则必须变换经过训练网络的预测值，以获得原始响应的预测值。

绘制响应的分布。响应（以度为单位的旋转角度）大致均匀地分布在 -45 和 45 之间，效果很好，无需归一化。在分类问题中，输出是类概率，始终需要归一化。

```
figure
histogram(YTrain)
axis tight
ylabel('Counts')
xlabel('Rotation Angle')
```



通常，数据不必完全归一化。但是，如果在此示例中训练网络来预测  $100 * \text{YTrain}$  或  $\text{YTrain} + 500$  而不是  $\text{YTrain}$ ，则损失将变为 NaN，并且网络参数在训练开始时会发生偏离。即使预测  $aY + b$  的网络与预测  $Y$  的网络之间的唯一差异是对最终全连接层的权重和偏置的简单重新缩放，也会出现这些结果。

如果输入或响应的分布非常不均匀或偏斜，您还可以在训练网络之前对数据执行非线性变换（例如，取其对数）。

### 创建网络层

要求解回归问题，请创建网络层并在网络末尾包含一个回归层。

第一层定义输入数据的大小和类型。输入图像的大小为  $28 \times 28 \times 1$ 。创建与训练图像大小相同的图像输入层。

网络的中间层定义网络的核心架构，大多数计算和学习都在此处进行。

最终层定义输出数据的大小和类型。对于回归问题，全连接层必须位于网络末尾的回归层之前。创建一个大小为 1 的全连接输出层以及一个回归层。

在 Layer 数组中将所有层组合在一起。

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer
    averagePooling2dLayer(2,'Stride',2)
    fullyConnectedLayer(1)
    regressionLayer]
```

```

convolution2dLayer(3,16,'Padding','same')
batchNormalizationLayer
reluLayer
averagePooling2dLayer(2,'Stride',2)
convolution2dLayer(3,32,'Padding','same')
batchNormalizationLayer
reluLayer
convolution2dLayer(3,32,'Padding','same')
batchNormalizationLayer
reluLayer
dropoutLayer(0.2)
fullyConnectedLayer(1)
regressionLayer];

```

## 训练网络

创建网络训练选项。进行 30 轮训练。将初始学习率设置为 0.001，并在 20 轮训练后降低学习率。通过指定验证数据和验证频率，监控训练过程中的网络准确度。软件基于训练数据训练网络，并在训练过程中按固定时间间隔计算基于验证数据的准确度。验证数据不用于更新网络权重。打开训练进度图，关闭命令行窗口输出。

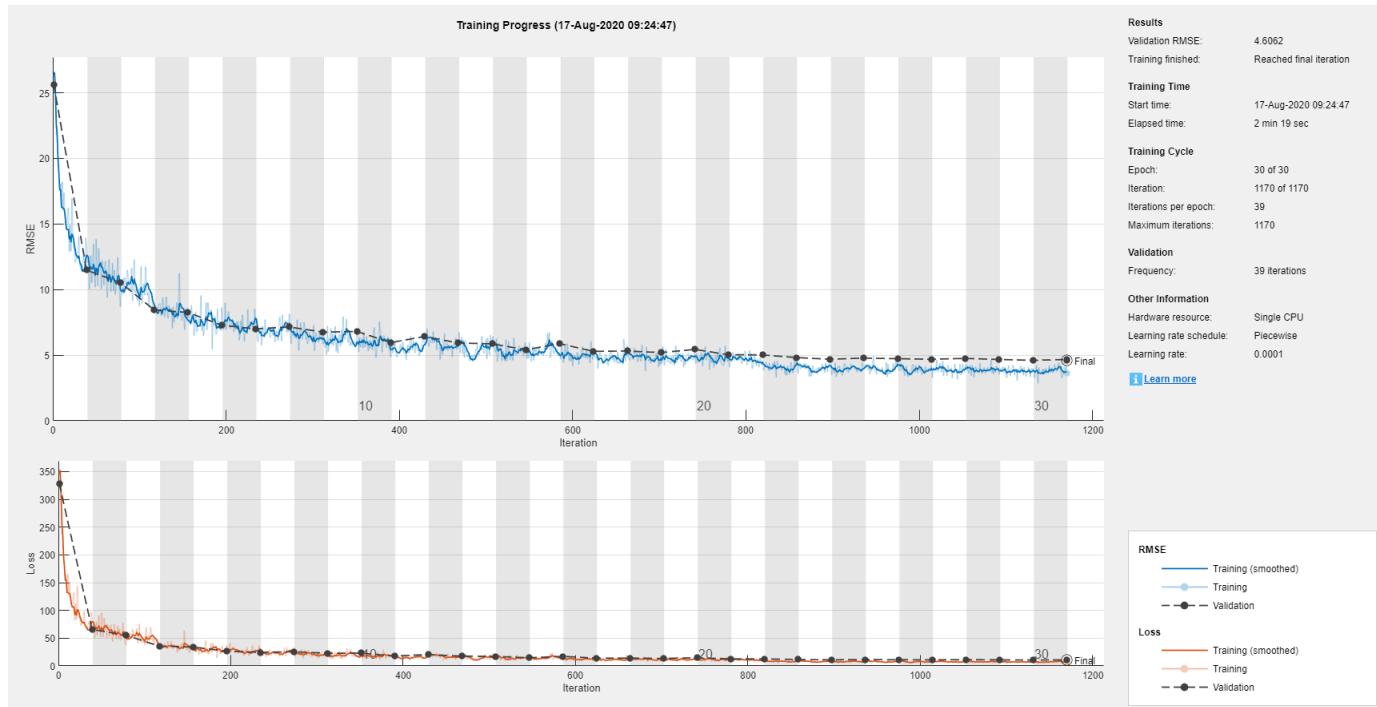
```

miniBatchSize = 128;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ...
    'MaxEpochs',30, ...
    'InitialLearnRate',1e-3, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',20, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',validationFrequency, ...
    'Plots','training-progress', ...
    'Verbose',false);

```

使用 `trainNetwork` 创建网络。如果存在兼容的 GPU，此命令会使用 GPU。使用 GPU 需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。否则，`trainNetwork` 将使用 CPU。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



检查 net 的 Layers 属性中包含的网络架构的详细信息。

`net.Layers`

```
ans =
18×1 Layer array with layers:
1 'imageinput'      Image Input      28×28×1 images with 'zerocenter' normalization
2 'conv_1'          Convolution     8 3×3×1 convolutions with stride [1 1] and padding 'same'
3 'batchnorm_1'    Batch Normalization  Batch normalization with 8 channels
4 'relu_1'          ReLU           ReLU
5 'avgpool2d_1'    Average Pooling   2×2 average pooling with stride [2 2] and padding [0 0 0 0]
6 'conv_2'          Convolution     16 3×3×8 convolutions with stride [1 1] and padding 'same'
7 'batchnorm_2'    Batch Normalization  Batch normalization with 16 channels
8 'relu_2'          ReLU           ReLU
9 'avgpool2d_2'    Average Pooling   2×2 average pooling with stride [2 2] and padding [0 0 0 0]
10 'conv_3'         Convolution     32 3×3×16 convolutions with stride [1 1] and padding 'same'
11 'batchnorm_3'    Batch Normalization  Batch normalization with 32 channels
12 'relu_3'          ReLU           ReLU
13 'conv_4'          Convolution     32 3×3×32 convolutions with stride [1 1] and padding 'same'
14 'batchnorm_4'    Batch Normalization  Batch normalization with 32 channels
15 'relu_4'          ReLU           ReLU
16 'dropout'        Dropout         20% dropout
17 'fc'             Fully Connected   1 fully connected layer
18 'regressionoutput' Regression Output mean-squared-error with response 'Response'
```

## 测试网络

基于验证数据评估准确度来测试网络性能。

使用 `predict` 预测验证图像的旋转角度。

```
YPredicted = predict(net,XValidation);
```

## 评估性能

通过计算以下值来评估模型性能：

- 1 在可接受误差界限内的预测值的百分比
- 2 预测旋转角度和实际旋转角度的均方根误差 (RMSE)

计算预测旋转角度和实际旋转角度之间的预测误差。

```
predictionError = YValidation - YPredicted;
```

计算在实际角度的可接受误差界限内的预测值的数量。将阈值设置为 10 度。计算此阈值范围内的预测值的百分比。

```
thr = 10;
numCorrect = sum(abs(predictionError) < thr);
numValidationImages = numel(YValidation);
```

```
accuracy = numCorrect/numValidationImages
```

```
accuracy = 0.9690
```

使用均方根误差 (RMSE) 来衡量预测旋转角度和实际旋转角度之间的差异。

```
squares = predictionError.^2;
rmse = sqrt(mean(squares))
```

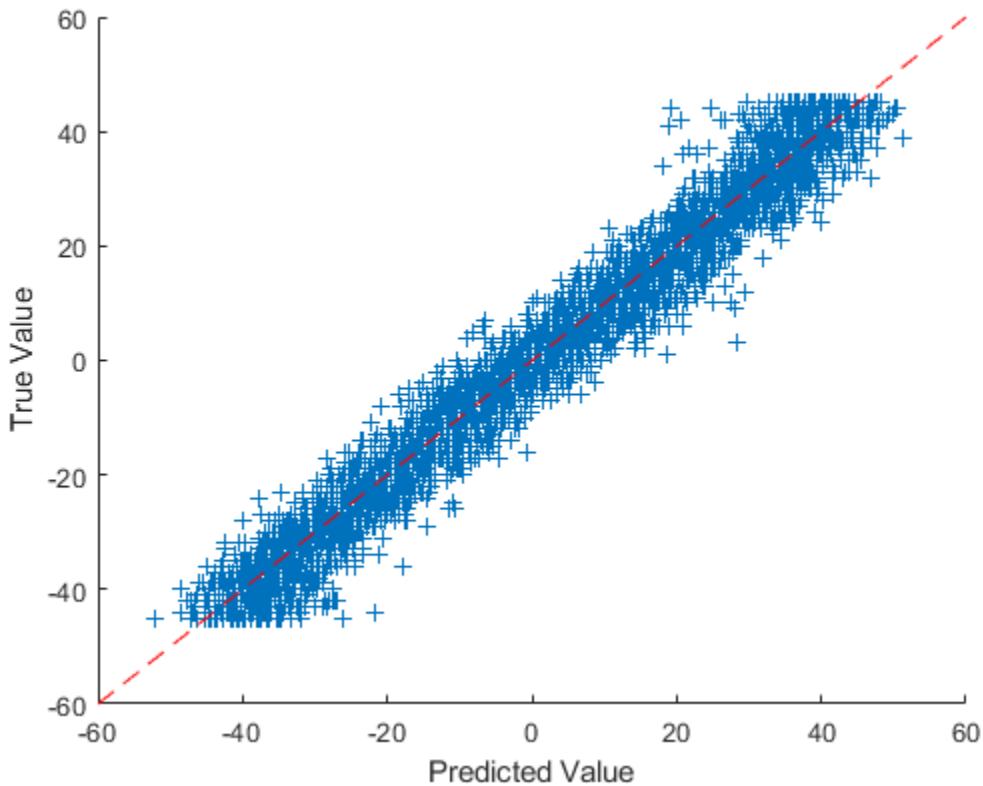
```
rmse = single
4.6062
```

## 可视化预测

在散点图中可视化预测。绘制预测值对真实值的图。

```
figure
scatter(YPredicted,YValidation,'+')
xlabel("Predicted Value")
ylabel("True Value")

hold on
plot([-60 60], [-60 60],'r--')
```



### 校正数字旋转

您可以使用 Image Processing Toolbox 中的函数来摆正数字并将它们显示在一起。使用 `imrotate` (Image Processing Toolbox) 根据预测的旋转角度旋转 49 个样本数字。

```
idx = randperm(numValidationImages,49);
for i = 1:numel(idx)
    image = XValidation(:,:,idx(i));
    predictedAngle = YPredicted(idx(i));
    imagesRotated(:,:,:,:i) = imrotate(image,predictedAngle,'bicubic','crop');
end
```

显示原始数字以及校正旋转后的数字。您可以使用 `montage` (Image Processing Toolbox) 将数字显示在同一个图像上。

```
figure
subplot(1,2,1)
montage(XValidation(:,:,:,:idx))
title('Original')

subplot(1,2,2)
montage(imagesRotated)
title('Corrected')
```

Original	Corrected
4 0 3 3 1 1 3	4 0 1 3 1 1 3
4 1 1 8 3 4 1	4 1 1 8 3 4 1
1 6 2 6 1 3 2	7 6 2 6 1 3 2
8 7 1 1 0 6 2	8 7 7 7 0 6 2
5 5 2 0 0 3 2	5 5 2 0 0 3 2
2 4 0 3 4 2 0	2 4 0 3 4 2 0
0 3 3 3 8 8 8	0 3 3 3 8 8 8

## 另请参阅

[regressionLayer](#) | [classificationLayer](#)

## 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “将分类网络转换为回归网络” (第 3-52 页)

## 将分类网络转换为回归网络

此示例说明如何将经过训练的分类网络转换为回归网络。

预训练的图像分类网络已经对超过一百万个图像进行了训练，可以将图像分为 1000 个对象类别，例如键盘、咖啡杯、铅笔和多种动物。这些网络已基于大量图像学习了丰富的特征表示。网络以图像作为输入，然后输出图像中对象的标签以及每个对象类别的概率。

深度学习应用中常常用到迁移学习。您可以采用预训练的网络，基于它学习新任务。此示例说明如何加载预训练的分类网络，以及如何重新训练该网络以用于回归任务。

在此示例中，我们会加载一个预训练的用于分类的卷积神经网络架构，然后替换用于分类的层并重新训练网络，以预测手写数字的旋转角度。您还可以选择使用 `imrotate` (Image Processing Toolbox™)，利用预测值来校正图像旋转。

### 加载预训练网络

从支持文件 `digitsNet.mat` 中加载预训练网络。此文件包含对手写数字进行分类的分类网络。

```
load digitsNet
layers = net.Layers

layers =
15×1 Layer array with layers:

 1 'imageinput'  Image Input      28×28×1 images with 'zerocenter' normalization
 2 'conv_1'       Convolution     8 3×3×1 convolutions with stride [1 1] and padding 'same'
 3 'batchnorm_1'  Batch Normalization  Batch normalization with 8 channels
 4 'relu_1'       ReLU           ReLU
 5 'maxpool_1'   Max Pooling    2×2 max pooling with stride [2 2] and padding [0 0 0 0]
 6 'conv_2'       Convolution     16 3×3×8 convolutions with stride [1 1] and padding 'same'
 7 'batchnorm_2'  Batch Normalization  Batch normalization with 16 channels
 8 'relu_2'       ReLU           ReLU
 9 'maxpool_2'   Max Pooling    2×2 max pooling with stride [2 2] and padding [0 0 0 0]
10 'conv_3'       Convolution     32 3×3×16 convolutions with stride [1 1] and padding 'same'
11 'batchnorm_3'  Batch Normalization  Batch normalization with 32 channels
12 'relu_3'       ReLU           ReLU
13 'fc'          Fully Connected  10 fully connected layer
14 'softmax'     Softmax         softmax
15 'classoutput' Classification Output  crossentropyex with '0' and 9 other classes
```

### 加载数据

数据集包含手写数字的合成图像以及每个图像的旋转角度（以度为单位）。

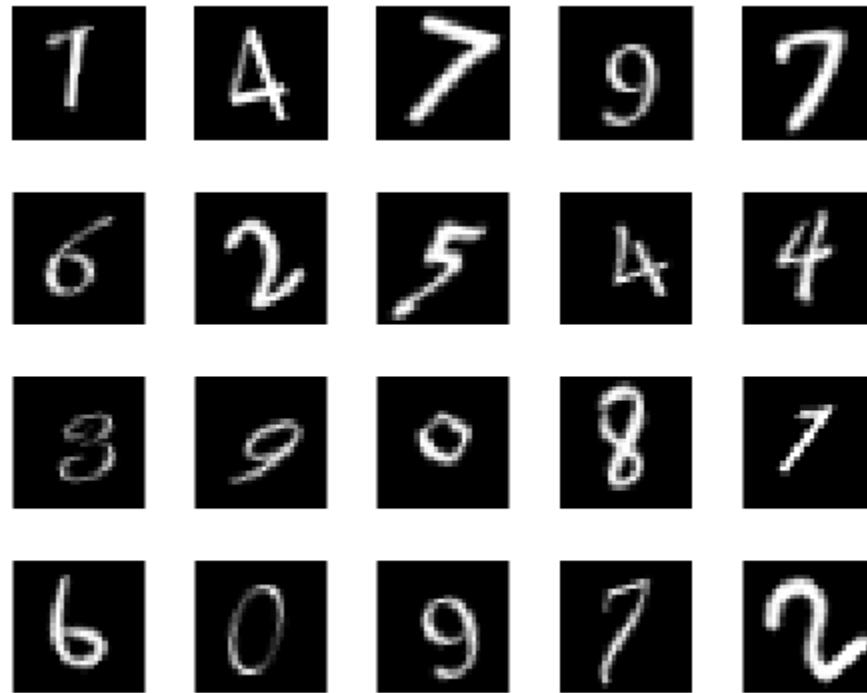
使用 `digitTrain4DArrayData` 和 `digitTest4DArrayData` 以四维数组的形式加载训练图像和验证图像。输出 `YTrain` 和 `YValidation` 是以度为单位的旋转角度。训练数据集和验证数据集各包含 5000 个图像。

```
[XTrain,~,YTrain] = digitTrain4DArrayData;
[XValidation,~,YValidation] = digitTest4DArrayData;
```

使用 `imshow` 显示 20 个随机训练图像。

```
numTrainImages = numel(YTrain);
figure
idx = randperm(numTrainImages,20);
```

```
for i = 1:numel(idx)
    subplot(4,5,i)
    imshow(XTrain(:,:,idx(i)))
end
```



## 替换最终层

网络的卷积层会提取最后一个可学习层和最终分类层用来对输入图像进行分类的图像特征。**digitsNet** 中的 'fc' 和 'classoutput' 这两个层包含有关如何将网络提取的特征合并成类概率、损失值和预测标签的信息。要重新训练一个预训练网络以用于回归任务，需要将这两个层替换为适用于该任务的新层。

将最终全连接层（softmax 层）和分类输出层替换为大小为 1（响应数）的全连接层和回归层。

```
numResponses = 1;
layers = [
    layers(1:12)
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

## 冻结初始层

现在，网络已准备好可以基于新数据进行重新训练。您也可以选择将较浅网络层的学习率设置为零，来“冻结”这些层的权重。在训练过程中，**trainNetwork** 不会更新已冻结层的参数。由于不需要计算已冻结层的梯度，因此冻结多个初始层的权重可以显著加快网络训练速度。如果新数据集很小，冻结较浅的网络层还可以防止这些层过拟合新数据集。

使用辅助函数 **freezeWeights** 将前 12 个层的学习率设置为零。

```
layers(1:12) = freezeWeights(layers(1:12));
```

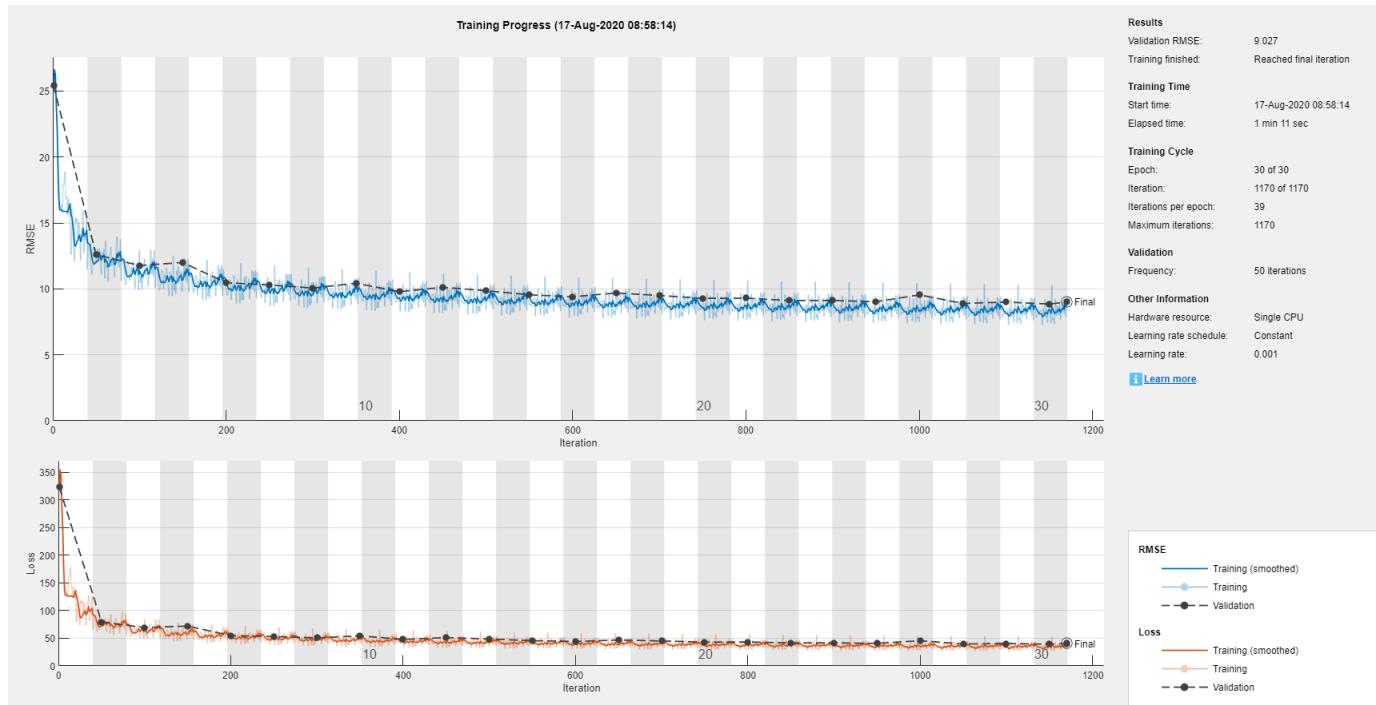
### 训练网络

创建网络训练选项。将初始学习率设置为 0.001。通过指定验证数据，监控训练过程中的网络准确度。打开训练进度图，关闭命令行窗口输出。

```
options = trainingOptions('sgdm',...
    'InitialLearnRate',0.001,...
    'ValidationData',{XValidation,YValidation},...
    'Plots','training-progress',...
    'Verbose',false);
```

使用 `trainNetwork` 创建网络。如果存在兼容的 GPU，此命令会使用 GPU。使用 GPU 需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。否则，`trainNetwork` 将使用 CPU。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



### 测试网络

基于验证数据评估准确度来测试网络性能。

使用 `predict` 预测验证图像的旋转角度。

```
YPred = predict(net,XValidation);
```

通过计算以下值来评估模型性能：

- 1 在可接受误差界限内的预测值的百分比
- 2 预测旋转角度和实际旋转角度的均方根误差 (RMSE)

计算预测旋转角度和实际旋转角度之间的预测误差。

```
predictionError = YValidation - YPred;
```

计算在实际角度的可接受误差界限内的预测值的数量。将阈值设置为 10 度。计算此阈值范围内的预测值的百分比。

```
thr = 10;
numCorrect = sum(abs(predictionError) < thr);
numImagesValidation = numel(YValidation);

accuracy = numCorrect/numImagesValidation

accuracy = 0.7532
```

使用均方根误差 (RMSE) 来衡量预测旋转角度和实际旋转角度之间的差异。

```
rmse = sqrt(mean(predictionError.^2))

rmse = single
9.0270
```

### 校正数字旋转

您可以使用 Image Processing Toolbox 中的函数来摆正数字并将它们显示在一起。使用 **imrotate** (Image Processing Toolbox) 根据预测的旋转角度旋转 49 个样本数字。

```
idx = randperm(numImagesValidation,49);
for i = 1:numel(idx)
    I = XValidation(:, :, :, idx(i));
    Y = YPred(idx(i));
    XValidationCorrected(:, :, :, i) = imrotate(I, Y, 'bicubic', 'crop');
end
```

显示原始数字以及校正旋转后的数字。使用 **montage** (Image Processing Toolbox) 将数字一起显示在一个图像上。

```
figure
subplot(1,2,1)
montage(XValidation(:, :, :, idx))
title('Original')

subplot(1,2,2)
montage(XValidationCorrected)
title('Corrected')
```

Original	Corrected
4 6 A 1 2 7 4	4 6 4 1 1 1 4
9 5 9 0 3 9 0	9 6 9 0 3 9 0
Y > 8 6 5 a >	1 7 8 6 5 9 7
6 > 8 9 9 2	6 7 8 9 9 9 2
0 8 1 0 1 5 3	0 8 1 0 7 5 3
5 8 0 4 5 + 2	5 8 6 4 5 1 2
2 2 9 e 8 3 Y	2 2 9 e 8 3 1

## 另请参阅

[regressionLayer](#) | [classificationLayer](#)

## 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)

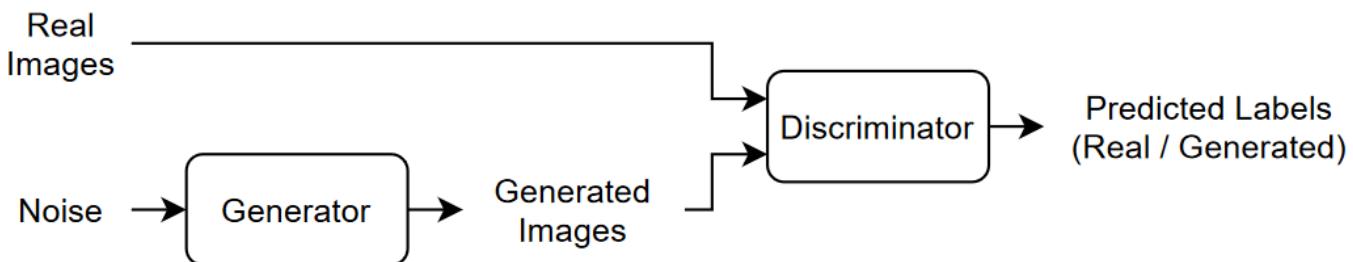
## 训练生成对抗网络 (GAN)

此示例说明如何训练生成对抗网络来生成图像。

生成对抗网络 (GAN) 是一种深度学习网络，它能够生成与真实输入数据具有相似特征的数据。

一个 GAN 由两个一起训练的网络组成：

- 1 生成器 - 给定随机值（潜在输入）向量作为输入，此网络可生成与训练数据具有相同结构的数据。
- 2 判别器 - 给定包含来自训练数据和来自生成器的生成数据的观测值的数据批量，此网络尝试将观测值划分为 "real" 或 "generated"。



要训练 GAN，需要同时训练两个网络以最大化两个网络的性能：

- 训练生成器以生成“欺骗”判别器的数据。
- 训练判别器以区分真实数据和生成的数据。

为了优化生成器的性能，当给定生成的数据时，最大化判别器的损失。也就是说，生成器的目标是生成判别器分类为 "real" 的数据。

为了优化判别器的性能，当给定真实数据和生成的数据批量时，最小化判别器的损失。即判别器的目标是不被生成器“欺骗”。

理想情况下，这些策略会得到能够生成令人信服的真实数据的生成器，以及已学习到训练数据特有的强特征表示的判别器。

### 加载训练数据

下载并提取 Flowers 数据集 [1]。

```

url = 'http://download.tensorflow.org/example_images/flower_photos.tgz';
downloadFolder = tempdir;
filename = fullfile(downloadFolder,'flower_dataset.tgz');

imageFolder = fullfile(downloadFolder,'flower_photos');
if ~exist(imageFolder,'dir')
    disp('Downloading Flowers data set (218 MB...)')
    websave(filename,url);
    untar(filename,downloadFolder)
end
  
```

创建一个包含花卉照片的图像数据库。

```
datasetFolder = fullfile(imageFolder);
```

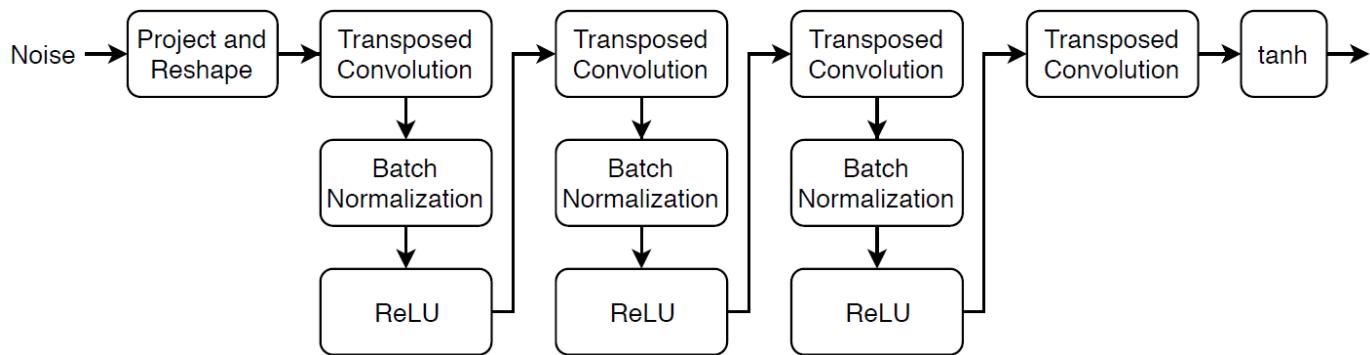
```
imds = imageDatastore(datasetFolder, ...
    'IncludeSubfolders',true);
```

增强数据以包括随机水平翻转，并将图像大小调整为  $64 \times 64$ 。

```
augmenter = imageDataAugmenter('RandXReflection',true);
augimds = augmentedImageDatastore([64 64],imds,'DataAugmentation',augmenter);
```

### 定义生成器网络

定义以下网络架构，它使用大小为 100 的随机向量生成图像。



此网络：

- 使用投影和重构层将大小为 100 的随机向量转换为  $7 \times 7 \times 128$  数组。
- 使用一系列带批量归一化和 ReLU 层的转置卷积层，将生成的数组扩增到  $64 \times 64 \times 3$  数组。

将此网络架构定义为一个层次图，并指定以下网络属性。

- 对于转置卷积层，指定  $5 \times 5$  滤波器，每一层的滤波器数量递减，步幅为 2，并在每条边裁剪输出。
- 对于最终的转置卷积层，指定与生成图像的三个 RGB 通道对应的三个  $5 \times 5$  滤波器，以及前一层的输出大小。
- 在网络末尾，包括一个 tanh 层。

要投影和重构噪声输入，请使用自定义层 `projectAndReshapeLayer`，它以支持文件的形式附加到此示例中。`projectAndReshapeLayer` 对象使用一个全连接操作来扩增输入，并将输出重构为指定的大小。

```
filterSize = 5;
numFilters = 64;
numLatentInputs = 100;

projectionSize = [4 4 512];

layersGenerator = [
    featureInputLayer(numLatentInputs,'Name','in')
    projectAndReshapeLayer(projectionSize,numLatentInputs,'Name','proj');
    transposedConv2dLayer(filterSize,4*numFilters,'Name','tconv1')
    batchNormalizationLayer('Name','bnorm1')
    reluLayer('Name','relu1')
    transposedConv2dLayer(filterSize,2*numFilters,'Stride',2,'Cropping','same','Name','tconv2')
    batchNormalizationLayer('Name','bnorm2')
    reluLayer('Name','relu2')
    transposedConv2dLayer(filterSize,numFilters,'Stride',2,'Cropping','same','Name','tconv3')
```

```

batchNormalizationLayer('Name','bnorm3')
reluLayer('Name','relu3')
transposedConv2dLayer(filterSize,3,'Stride',2,'Cropping','same','Name','tconv4')
tanhLayer('Name','tanh')];

lgraphGenerator = layerGraph(layersGenerator);

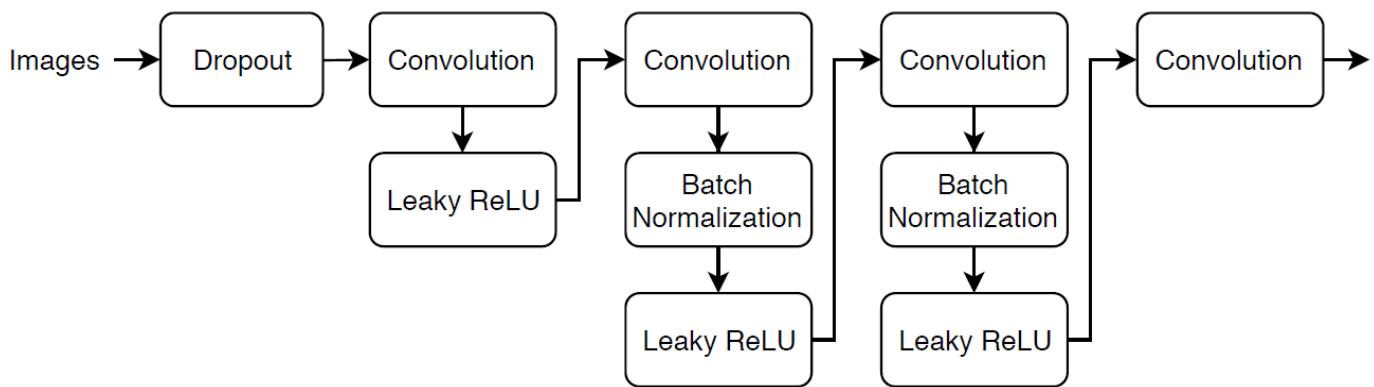
```

要使用自定义训练循环训练网络并支持自动微分，请将层次图转换为 `dlnetwork` 对象。

```
dlnetGenerator = dlnetwork(lgraphGenerator);
```

### 定义判别器网络

定义以下网络，它对真实图像和生成的  $64 \times 64$  图像进行分类。



创建一个网络，该网络接受  $64 \times 64 \times 3$  图像，并使用一系列具有批量归一化和泄漏 ReLU 层的卷积层返回一个标量预测分数。使用丢弃法给输入图像添加噪声。

- 对于丢弃层，指定丢弃概率为 0.5。
- 对于卷积层，指定  $5 \times 5$  滤波器，每一层的滤波器数量递增。同时指定步幅为 2 以及对输出进行填充。
- 对于泄漏 ReLU 层，指定 0.2 的尺度。
- 对于最终层，指定具有一个  $4 \times 4$  滤波器的卷积层。

要输出范围 [0,1] 内的概率，请使用 `modelGradients` 函数中的 `sigmoid` 函数，该函数列在示例的模型梯度函数（第 3-0 页）部分。

```

dropoutProb = 0.5;
numFilters = 64;
scale = 0.2;

inputSize = [64 64 3];
filterSize = 5;

layersDiscriminator = [
    imageInputLayer(inputSize,'Normalization','none','Name','in')
    dropoutLayer(dropoutProb,'Name','dropout')
    convolution2dLayer(filterSize,numFilters,'Stride',2,'Padding','same','Name','conv1')
    leakyReluLayer(scale,'Name','lrelu1')
    convolution2dLayer(filterSize,2*numFilters,'Stride',2,'Padding','same','Name','conv2')
    batchNormalizationLayer('Name','bn2')
    leakyReluLayer(scale,'Name','lrelu2')
    convolution2dLayer(filterSize,4*numFilters,'Stride',2,'Padding','same','Name','conv3')
]

```

```

batchNormalizationLayer('Name','bn3')
leakyReluLayer(scale,'Name','lrelu3')
convolution2dLayer(filterSize,8*numFilters,'Stride',2,'Padding','same','Name','conv4')
batchNormalizationLayer('Name','bn4')
leakyReluLayer(scale,'Name','lrelu4')
convolution2dLayer(4,1,'Name','conv5')];

lgraphDiscriminator = layerGraph(layersDiscriminator);

```

要使用自定义训练循环训练网络并支持自动微分，请将层次图转换为 **dlnetwork** 对象。

```
dlnetDiscriminator = dlnetwork(lgraphDiscriminator);
```

### 定义模型梯度和损失函数

创建在示例的模型梯度函数（第 3-0 页）部分列出的函数 **modelGradients**，该函数接受生成器和判别器网络、小批量输入数据、随机值数组和翻转因子作为输入，并返回损失关于网络中可学习参数的梯度和两个网络的分数。

### 指定训练选项

使用小批量大小 128 进行 500 轮训练。对于较大的数据集，您可能不需要进行这么多轮训练。

```
numEpochs = 500;
miniBatchSize = 128;
```

指定 Adam 优化的选项。对于两个网络，都指定：

- 学习率为 0.0002
- 梯度衰减因子为 0.5
- 梯度平方衰减因子为 0.999

```
learnRate = 0.0002;
gradientDecayFactor = 0.5;
squaredGradientDecayFactor = 0.999;
```

如果判别器过快地学会了如何判别真实图像和生成的图像，则生成器可能无法进行训练。为了更好地平衡判别器和生成器的学习，请通过随机翻转标签向真实数据添加噪声。

指定 0.3 的 **flipFactor** 值以翻转 30% 的真实标签（总标签的 15%）。请注意，这不会减损生成器，因为所有生成的图像仍被正确标注。

```
flipFactor = 0.3;
```

每经过 100 次迭代就显示生成的验证图像。

```
validationFrequency = 100;
```

### 训练模型

使用 **minibatchqueue** 处理和管理小批量图像。对于每个小批量：

- 使用自定义小批量预处理函数 **preprocessMiniBatch**（在此示例末尾定义）在 [-1,1] 范围内重新缩放图像。
- 丢弃观测值少于 128 个的任何不完整小批量。
- 用维度标签 'SSCB'（空间、空间、通道、批量）格式化图像数据。默认情况下，**minibatchqueue** 对象将数据转换为基础类型为 **single** 的 **dlarray** 对象。

- 在 GPU 上（如果有）进行训练。当 `minibatchqueue` 的 'OutputEnvironment' 选项为 "auto" 时，`minibatchqueue` 将每个输出转换为 `gpuArray`（如果 GPU 可用）。使用 GPU 需要 Parallel Computing Toolbox™ 和具有 3.0 或更高计算能力的支持 CUDA® 的 NVIDIA® GPU。

```
augimds.MiniBatchSize = miniBatchSize;
executionEnvironment = "auto";
mbq = minibatchqueue(augimds, ...
    'MiniBatchSize',miniBatchSize, ...
    'PartialMiniBatch','discard', ...
    'MiniBatchFcn',@preprocessMiniBatch, ...
    'MiniBatchFormat','SSCB', ...
    'OutputEnvironment',executionEnvironment);
```

使用自定义训练循环训练模型。在每次迭代中循环使用训练数据并更新网络参数。为了监控训练进度，以保留的随机值数组作为生成器输入来显示一批生成图像，同时显示其相关的分数图。

为 Adam 初始化参数。

```
trailingAvgGenerator = [];
trailingAvgSqGenerator = [];
trailingAvgDiscriminator = [];
trailingAvgSqDiscriminator = [];
```

为了监控训练进度，使用一批保留的随机值固定向量作为输入馈送到生成器，显示一批生成图像并绘制其网络分数图。

创建一个由保留的随机值组成的数组。

```
numValidationImages = 25;
ZValidation = randn(numLatentInputs,numValidationImages,'single');
```

将数据转换为 `dlarray` 对象，并指定维度标签 'CB'（通道、批量）。

```
dlZValidation = dlarray(ZValidation,'CB');
```

对于 GPU 训练，将数据转换为 `gpuArray` 对象。

```
if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
    dlZValidation = gpuArray(dlZValidation);
end
```

初始化训练进度图。创建一个图窗，并将其调整为两倍宽度。

```
f = figure;
f.Position(3) = 2*f.Position(3);
```

为生成的图像和网络分数创建子图。

```
imageAxes = subplot(1,2,1);
scoreAxes = subplot(1,2,2);
```

为分数图初始化动画线条。

```
lineScoreGenerator = animatedline(scoreAxes,'Color',[0 0.447 0.741]);
lineScoreDiscriminator = animatedline(scoreAxes, 'Color', [0.85 0.325 0.098]);
legend('Generator','Discriminator');
```

```

ylim([0 1])
 xlabel("Iteration")
 ylabel("Score")
 grid on

```

训练 GAN。对于每轮训练，对数据存储进行乱序处理，并循环使用小批量数据。

对于每个小批量：

- 使用 `dlfeval` 和 `modelGradients` 函数计算模型梯度。
- 使用 `adamupdate` 函数更新网络参数。
- 绘制两个网络的分数。
- 在每 `validationFrequency` 次迭代后，显示一批基于固定保留生成器输入的生成图像。

训练可能需要一些时间来运行。

```

iteration = 0;
start = tic;

% Loop over epochs.
for epoch = 1:numEpochs

    % Reset and shuffle datastore.
    shuffle(mbq);

    % Loop over mini-batches.
    while hasdata(mbq)
        iteration = iteration + 1;

        % Read mini-batch of data.
        dlX = next(mbq);

        % Generate latent inputs for the generator network. Convert to
        % dlarray and specify the dimension labels 'CB' (channel, batch).
        % If training on a GPU, then convert latent inputs to gpuArray.
        Z = randn(numLatentInputs,miniBatchSize,'single');
        dlZ = dlarray(Z,'CB');

        if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
            dlZ = gpuArray(dlZ);
        end

        % Evaluate the model gradients and the generator state using
        % dlfeval and the modelGradients function listed at the end of the
        % example.
        [gradientsGenerator, gradientsDiscriminator, stateGenerator, scoreGenerator, scoreDiscriminator] = ...
            dlfeval(@modelGradients, dlnetGenerator, dlnetDiscriminator, dlX, dlZ, flipFactor);
        dlnetGenerator.State = stateGenerator;

        % Update the discriminator network parameters.
        [dlnetDiscriminator,trailingAvgDiscriminator,trailingAvgSqDiscriminator] = ...
            adamupdate(dlnetDiscriminator, gradientsDiscriminator, ...
            trailingAvgDiscriminator, trailingAvgSqDiscriminator, iteration, ...
            learnRate, gradientDecayFactor, squaredGradientDecayFactor);

        % Update the generator network parameters.
        [dlnetGenerator,trailingAvgGenerator,trailingAvgSqGenerator] = ...

```

```

adamupdate(dlnetGenerator, gradientsGenerator, ...
trailingAvgGenerator, trailingAvgSqGenerator, iteration, ...
learnRate, gradientDecayFactor, squaredGradientDecayFactor);

% Every validationFrequency iterations, display batch of generated images using the
% held-out generator input.
if mod(iteration,validationFrequency) == 0 || iteration == 1
    % Generate images using the held-out generator input.
    dlXGeneratedValidation = predict(dlnetGenerator,dlZValidation);

    % Tile and rescale the images in the range [0 1].
    I = imtile(extractdata(dlXGeneratedValidation));
    I = rescale(I);

    % Display the images.
    subplot(1,2,1);
    image(imageAxes,I)
    xticklabels([]);
    yticklabels([]);
    title("Generated Images");
end

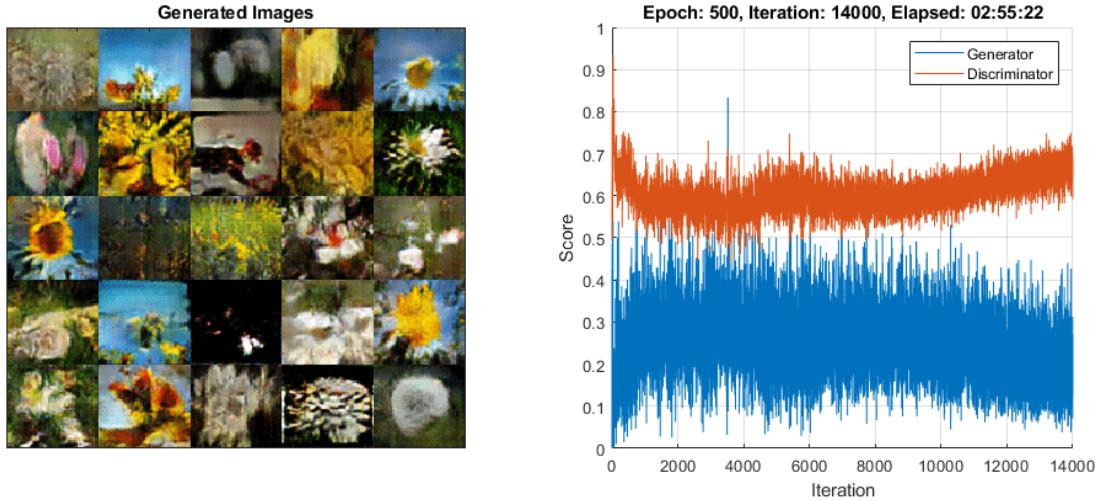
% Update the scores plot.
subplot(1,2,2)
addpoints(lineScoreGenerator,iteration, ...
    double(gather(extractdata(scoreGenerator))));

addpoints(lineScoreDiscriminator,iteration, ...
    double(gather(extractdata(scoreDiscriminator))));

% Update the title with training progress information.
D = duration(0,0,toc(start),'Format','hh:mm:ss');
title(... 
    "Epoch: " + epoch + ", " + ...
    "Iteration: " + iteration + ", " + ...
    "Elapsed: " + string(D))

drawnow
end
end

```



此时，判别器已学会在生成的图像中识别真实图像的强特征表示。顺带，生成器已学会类似的强特征表示，能够生成类似于训练数据的图像。

训练图显示生成器和判别器网络的分数。要了解有关如何解释网络分数的详细信息，请参阅“Monitor GAN Training Progress and Identify Common Failure Modes”。

### 生成新图像

要生成新图像，请使用生成器上的 `predict` 函数和包含一批随机向量的 `dlarray` 对象。要一起显示图像，请使用 `imtile` 函数，并使用 `rescale` 函数重新缩放图像。

创建一个 `dlarray` 对象（其中包含 25 个随机向量），以输入到生成器网络中。

```
numObservations = 25;
ZNew = randn(numLatentInputs,numObservations,'single');
dlZNew = dlarray(ZNew,'CB');
```

要使用 GPU 生成图像，还要将数据转换为 `gpuArray` 对象。

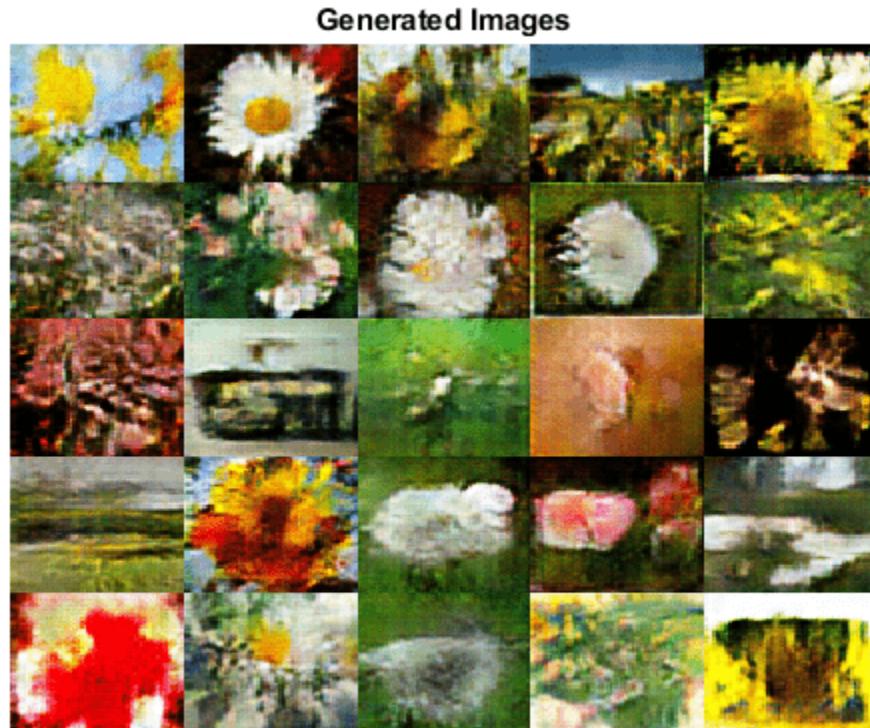
```
if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
    dlZNew = gpuArray(dlZNew);
end
```

使用 `predict` 函数以及生成器和输入数据生成新图像。

```
dlXGeneratedNew = predict(dlnetGenerator,dlZNew);
```

显示图像。

```
I = imtile(extractdata(dlXGeneratedNew));
I = rescale(I);
figure
image(I)
axis off
title("Generated Images")
```



## 模型梯度函数

函数 `modelGradients` 接受生成器和判别器 `dlnetwork` 对象 `dlnetGenerator` 和 `dlnetDiscriminator`、小批量输入数据 `dlX`、随机值数组 `dlZ` 和要翻转的真实标签的百分比 `flipFactor` 作为输入，并返回损失关于网络中可学习参数的梯度、生成器状态和两个网络的分数。由于判别器输出不在 [0,1] 范围中，`modelGradients` 函数应用 `sigmoid` 函数将其转换为概率。

```

function [gradientsGenerator, gradientsDiscriminator, stateGenerator, scoreGenerator, scoreDiscriminator] = ...
    modelGradients(dlnetGenerator, dlnetDiscriminator, dlX, dlZ, flipFactor)

% Calculate the predictions for real data with the discriminator network.
dlYPred = forward(dlnetDiscriminator, dlX);

% Calculate the predictions for generated data with the discriminator network.
[dlXGenerated,stateGenerator] = forward(dlnetGenerator,dlZ);
dlYPredGenerated = forward(dlnetDiscriminator, dlXGenerated);

% Convert the discriminator outputs to probabilities.
probGenerated = sigmoid(dlYPredGenerated);
probReal = sigmoid(dlYPred);

% Calculate the score of the discriminator.
scoreDiscriminator = (mean(probReal) + mean(1-probGenerated)) / 2;

% Calculate the score of the generator.
scoreGenerator = mean(probGenerated);

```

```
% Randomly flip a fraction of the labels of the real images.
numObservations = size(probReal,4);
idx = randperm(numObservations,floor(flipFactor * numObservations));

% Flip the labels.
probReal(:,:,:,:idx) = 1 - probReal(:,:,:,:idx);

% Calculate the GAN loss.
[lossGenerator, lossDiscriminator] = ganLoss(probReal,probGenerated);

% For each network, calculate the gradients with respect to the loss.
gradientsGenerator = dlgradient(lossGenerator, dlnetGenerator.Learnables,'RetainData',true);
gradientsDiscriminator = dlgradient(lossDiscriminator, dlnetDiscriminator.Learnables);

end
```

### GAN 损失函数和分数

生成器的目标是生成判别器分类为 "real" 的数据。为了最大化判别器将生成器生成的图像判别为真实图像的概率，最小化负对数似然函数。

给定判别器的输出  $Y$ :

- $\hat{Y} = \sigma(Y)$  是输入图像属于 "real" 类的概率。
- $1 - \hat{Y}$  是输入图像属于 "generated" 类的概率。

请注意，sigmoid 运算  $\sigma$  发生在 **modelGradients** 函数中。生成器的损失函数由下式给出

$$\text{lossGenerator} = -\text{mean}(\log(\hat{Y}_{\text{Generated}})),$$

其中  $\hat{Y}_{\text{Generated}}$  包含生成图像的判别器输出概率。

判别器的目标是不被生成器“欺骗”。为了最大化判别器成功判别真实图像和生成图像的概率，最小化对应的负对数似然函数之和。

判别器的损失函数由下式给出

$$\text{lossDiscriminator} = -\text{mean}(\log(\hat{Y}_{\text{Real}})) - \text{mean}(\log(1 - \hat{Y}_{\text{Generated}})),$$

其中  $\hat{Y}_{\text{Real}}$  包含真实图像的判别器输出概率。

要按照从 0 到 1 的范围来度量生成器和判别器实现各自目标的程度，可以使用分数的概念。

生成器分数是生成图像判别器输出的概率平均值：

$$\text{scoreGenerator} = \text{mean}(\hat{Y}_{\text{Generated}}).$$

判别器分数是真实图像和生成图像判别器输出的概率平均值：

$$\text{scoreDiscriminator} = \frac{1}{2}\text{mean}(\hat{Y}_{\text{Real}}) + \frac{1}{2}\text{mean}(1 - \hat{Y}_{\text{Generated}}).$$

分数与损失成反比，但实际上包含相同的信息。

```

function [lossGenerator, lossDiscriminator] = ganLoss(probReal,probGenerated)

% Calculate the loss for the discriminator network.
lossDiscriminator = -mean(log(probReal)) - mean(log(1-probGenerated));

% Calculate the loss for the generator network.
lossGenerator = -mean(log(probGenerated));

end

```

### 小批量预处理函数

**preprocessMiniBatch** 函数使用以下步骤预处理数据：

- 1 从传入的元胞数组中提取图像数据，并串联成一个数值数组。
- 2 将图像重新缩放到 [-1,1] 范围内。

```

function X = preprocessMiniBatch(data)

% Concatenate mini-batch
X = cat(4,data{:});

% Rescale the images in the range [-1 1].
X = rescale(X,-1,1,'InputMin',0,'InputMax',255);

end

```

### 参考资料

- 1 The TensorFlow Team. Flowers [http://download.tensorflow.org/example\\_images/flower\\_photos.tgz](http://download.tensorflow.org/example_images/flower_photos.tgz)
- 2 Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." Preprint, submitted November 19, 2015. <http://arxiv.org/abs/1511.06434>.

### 另请参阅

**dlnetwork | forward | predict | dlarray | dlgradient | dlfeval | adamupdate | minibatchqueue**

### 详细信息

- “Train Conditional Generative Adversarial Network (CGAN)”
- “Monitor GAN Training Progress and Identify Common Failure Modes”
- “Train Fast Style Transfer Network”
- “定义自定义训练循环、损失函数和网络” (第 17-33 页)
- “Train Network Using Custom Training Loop”
- “Specify Training Options in Custom Training Loop”
- “深度学习层列表” (第 1-18 页)
- “Deep Learning Tips and Tricks”

## 训练变分自编码器 (VAE) 以生成图像

此示例说明如何在 MATLAB 中创建一个变分自编码器 (VAE) 以生成数字图像。VAE 生成 MNIST 数据集样式的手写数字。

VAE 不同于常规的自编码器，因为它们不使用编码解码过程来重新构造输入。它们在潜在空间上施加一个概率分布，并且学习该分布，使得来自解码器的输出的分布与观测到的数据的分布相匹配。然后，它们从这个分布中采样以生成新数据。

在此示例中，您构造一个 VAE 网络，基于 MNIST 数据集对其进行训练，并生成与该数据集非常相似的新图像。

### 加载数据

从 <http://yann.lecun.com/exdb/mnist/> 下载 MNIST 文件，并将 MNIST 数据集加载到工作区中 [1]。调用此示例附带的 `processImagesMNIST` 和 `processLabelsMNIST` 辅助函数，将文件中的数据加载到 MATLAB 数组中。

由于 VAE 是将重新构造的数字与输入进行比较，而不是与分类标签进行比较，因此您不需要使用 MNIST 数据集中的训练标签。

```
trainImagesFile = 'train-images-idx3-ubyte.gz';
testImagesFile = 't10k-images-idx3-ubyte.gz';
testLabelsFile = 't10k-labels-idx1-ubyte.gz';

XTrain = processImagesMNIST(trainImagesFile);

Read MNIST image data...
Number of images in the dataset: 60000 ...

numTrainImages = size(XTrain,4);
XTest = processImagesMNIST(testImagesFile);

Read MNIST image data...
Number of images in the dataset: 10000 ...

YTest = processLabelsMNIST(testLabelsFile);

Read MNIST label data...
Number of labels in the dataset: 10000 ...
```

### 构造网络

自编码器有两个部分：编码器和解码器。编码器接受一个图像输入并输出一个压缩表示（编码），该压缩表示是大小为 `latentDim` 的向量，在此示例中等于 20。解码器接受该压缩表示，对其进行解码，并重新创建原始图像。

为了使计算在数值上更加稳定，通过让网络从方差的对数中学习，将可能值的范围从  $[0,1]$  增加到  $[-\inf, 0]$ 。定义大小为 `latent_dim` 的两个向量：一个用于均值  $\mu$ ，另一个用于方差的对数  $\log(\sigma^2)$ 。然后使用这两个向量创建采样分布。

使用二维卷积，后跟一个全连接层，将图像从  $28 \times 28 \times 1$  MNIST 图像下采样至潜在空间中的编码。然后，使用转置的二维卷积将  $1 \times 1 \times 20$  编码放大回  $28 \times 28 \times 1$  图像。

```
latentDim = 20;
imageSize = [28 28 1];
```

```

encoderLG = layerGraph([
    imageInputLayer(imageSize,'Name','input_encoder','Normalization','none')
    convolution2dLayer(3, 32, 'Padding','same', 'Stride', 2, 'Name', 'conv1')
    reluLayer('Name','relu1')
    convolution2dLayer(3, 64, 'Padding','same', 'Stride', 2, 'Name', 'conv2')
    reluLayer('Name','relu2')
    fullyConnectedLayer(2 * latentDim, 'Name', 'fc_encoder')
]);

decoderLG = layerGraph([
    imageInputLayer([1 1 latentDim], 'Name', 'i', 'Normalization', 'none')
    transposedConv2dLayer(7, 64, 'Cropping', 'same', 'Stride', 7, 'Name', 'transpose1')
    reluLayer('Name','relu1')
    transposedConv2dLayer(3, 64, 'Cropping', 'same', 'Stride', 2, 'Name', 'transpose2')
    reluLayer('Name','relu2')
    transposedConv2dLayer(3, 32, 'Cropping', 'same', 'Stride', 2, 'Name', 'transpose3')
    reluLayer('Name','relu3')
    transposedConv2dLayer(3, 1, 'Cropping', 'same', 'Name', 'transpose4')
]);

```

要使用自定义训练循环训练两个网络并支持自动微分，请将层次图转换为 **dlnetwork** 对象。

```

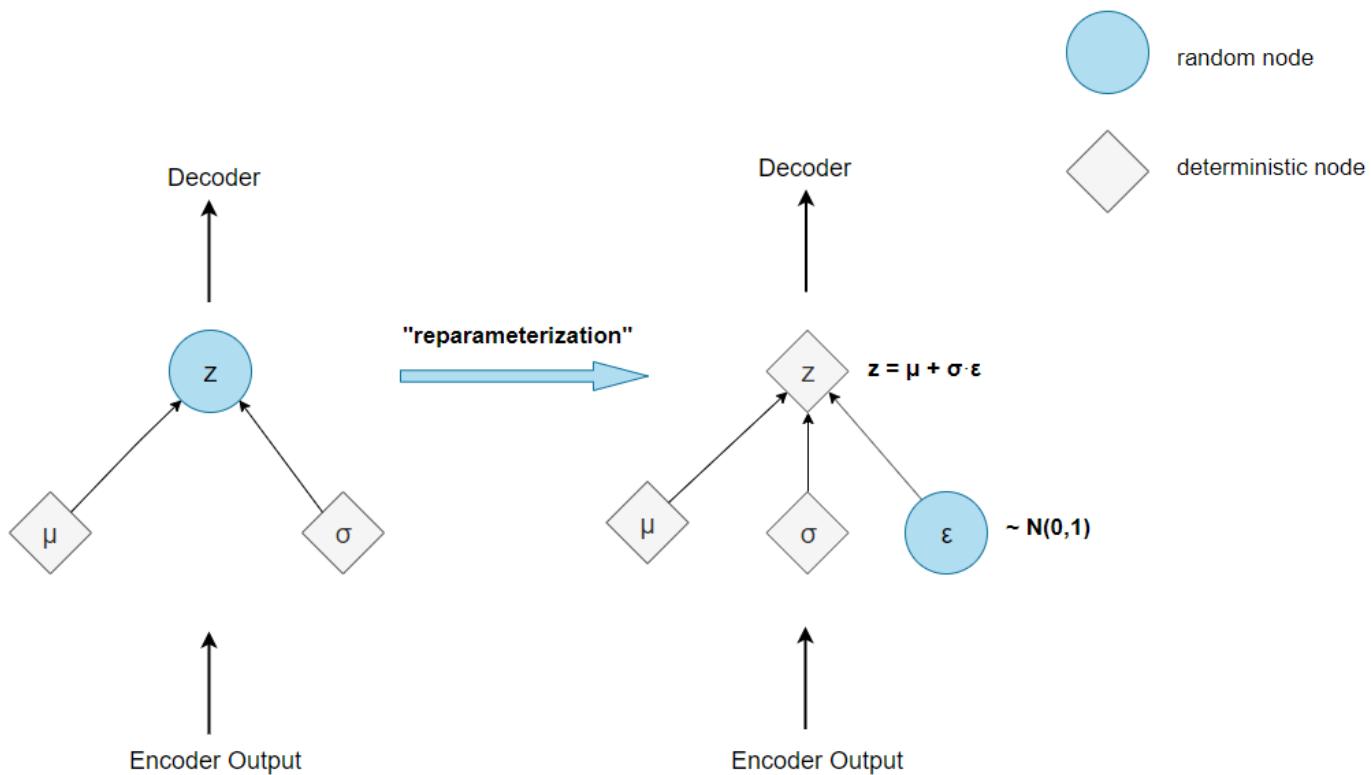
encoderNet = dlnetwork(encoderLG);
decoderNet = dlnetwork(decoderLG);

```

### 定义模型梯度函数

辅助函数 **modelGradients** (第 3-0 页) 接受编码器和解码器 **dlnetwork** 对象以及输入数据 **X** 的一个小批量，并返回损失关于网络中可学习参数的梯度。此辅助函数在此示例的末尾定义。

该函数分两步执行此过程：采样和损失 (第 3-0 页)。采样步骤对均值向量和方差向量进行采样，以创建要传递给解码器网络的最终编码。但是，由于无法通过随机采样操作进行反向传播，您必须使用重参数化方法。此方法将随机采样运算移至辅助变量  $\varepsilon$ ，然后通过按均值  $\mu_i$  进行移位，并按标准差  $\sigma_i$  进行缩放。其想法是从  $N(\mu_i, \sigma_i^2)$  采样与从  $\mu_i + \varepsilon \cdot \sigma_i$  采样相同，其中  $\varepsilon \sim N(0, 1)$ 。下图形象地说明了此想法。



损失步骤将在采样步骤生成的编码传入解码器网络进行处理，确定损失，然后使用损失计算梯度。VAE 中的损失，也称为证据下界 (ELBO) 损失，定义为两个独立损失项之和：

$$\text{ELBO loss} = \text{reconstruction loss} + \text{KL loss}.$$

重建损失通过使用均方误差 (MSE) 来测量解码器输出与原始输入的接近程度：

$$\text{reconstruction loss} = \text{MSE}(\text{decoder output}, \text{original image}).$$

KL 损失，即 Kullback–Leibler 散度，测量两个概率分布之间的差异。在本例中，最小化 KL 损失意味着确保学习的均值和方差尽可能接近目标 (正态) 分布的均值和方差。对于大小为  $n$  的潜在维度，KL 损失的计算公式如下

$$\text{KL loss} = -0.5 \cdot \sum_{i=1}^n (1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2).$$

包含 KL 损失项的实际效果是将由于重建损失而学习到的聚类紧密地聚集在潜在空间中心周围，形成连续的采样空间。

### 指定训练选项

在 GPU 上 (如果有) 训练 (需要 Parallel Computing Toolbox™)。

`executionEnvironment = "auto";`

指定网络的训练选项。在使用 Adam 优化器时，您需要用空数组初始化每个网络的尾部平均梯度和尾部平均梯度平方衰减率。

```
numEpochs = 50;
miniBatchSize = 512;
```

```

lr = 1e-3;
numIterations = floor(numTrainImages/miniBatchSize);
iteration = 0;

avgGradientsEncoder = [];
avgGradientsSquaredEncoder = [];
avgGradientsDecoder = [];
avgGradientsSquaredDecoder = [];

```

## 训练模型

使用自定义训练循环训练模型。

对于一轮训练中的每次迭代：

- 从训练集中获取下一个批量。
- 将小批量转换为 `dlarray` 对象，确保指定维度标签 '`SSCB`' (空间、空间、通道、批量)。
- 对于 GPU 训练，将 `dlarray` 转换为 `gpuArray` 对象。
- 使用 `dlfeval` 和 `modelGradients` 函数计算模型梯度。
- 使用 `adamupdate` 函数更新两个网络的网络可学习参数和平均梯度。

在每轮训练结束时，传入测试集图像，使之通过自编码器，并显示该轮训练的损失和训练时间。

```

for epoch = 1:numEpochs
    tic;
    for i = 1:numIterations
        iteration = iteration + 1;
        idx = (i-1)*miniBatchSize+1:i*miniBatchSize;
        XBatch = XTrain(:,:,idx);
        XBatch = dlarray(single(XBatch), 'SSCB');

        if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment == "gpu"
            XBatch = gpuArray(XBatch);
        end

        [infGrad, genGrad] = dlfeval(...%
            @modelGradients, encoderNet, decoderNet, XBatch);

        [decoderNet.Learnables, avgGradientsDecoder, avgGradientsSquaredDecoder] = ...%
            adamupdate(decoderNet.Learnables, ...%
                genGrad, avgGradientsDecoder, avgGradientsSquaredDecoder, iteration, lr);
        [encoderNet.Learnables, avgGradientsEncoder, avgGradientsSquaredEncoder] = ...%
            adamupdate(encoderNet.Learnables, ...%
                infGrad, avgGradientsEncoder, avgGradientsSquaredEncoder, iteration, lr);
    end
    elapsedTime = toc;

    [z, zMean, zLogvar] = sampling(encoderNet, XTest);
    xPred = sigmoid(forward(decoderNet, z));
    elbo = ELBOloss(XTest, xPred, zMean, zLogvar);
    disp("Epoch : "+epoch+" Test ELBO loss = "+gather(extractdata(elbo))+...
        ". Time taken for epoch = "+elapsedTime +"s")
end

```

Epoch : 1 Test ELBO loss = 28.0145. Time taken for epoch = 28.0573s  
Epoch : 2 Test ELBO loss = 24.8995. Time taken for epoch = 8.797s

```
Epoch : 3 Test ELBO loss = 23.2756. Time taken for epoch = 8.8824s
Epoch : 4 Test ELBO loss = 21.151. Time taken for epoch = 8.5979s
Epoch : 5 Test ELBO loss = 20.5335. Time taken for epoch = 8.8472s
Epoch : 6 Test ELBO loss = 20.232. Time taken for epoch = 8.5068s
Epoch : 7 Test ELBO loss = 19.9988. Time taken for epoch = 8.4356s
Epoch : 8 Test ELBO loss = 19.8955. Time taken for epoch = 8.4015s
Epoch : 9 Test ELBO loss = 19.7991. Time taken for epoch = 8.8089s
Epoch : 10 Test ELBO loss = 19.6773. Time taken for epoch = 8.4269s
Epoch : 11 Test ELBO loss = 19.5181. Time taken for epoch = 8.5771s
Epoch : 12 Test ELBO loss = 19.4532. Time taken for epoch = 8.4227s
Epoch : 13 Test ELBO loss = 19.3771. Time taken for epoch = 8.5807s
Epoch : 14 Test ELBO loss = 19.2893. Time taken for epoch = 8.574s
Epoch : 15 Test ELBO loss = 19.1641. Time taken for epoch = 8.6434s
Epoch : 16 Test ELBO loss = 19.2175. Time taken for epoch = 8.8641s
Epoch : 17 Test ELBO loss = 19.158. Time taken for epoch = 9.1083s
Epoch : 18 Test ELBO loss = 19.085. Time taken for epoch = 8.6674s
Epoch : 19 Test ELBO loss = 19.1169. Time taken for epoch = 8.6357s
Epoch : 20 Test ELBO loss = 19.0791. Time taken for epoch = 8.5512s
Epoch : 21 Test ELBO loss = 19.0395. Time taken for epoch = 8.4674s
Epoch : 22 Test ELBO loss = 18.9556. Time taken for epoch = 8.3943s
Epoch : 23 Test ELBO loss = 18.9469. Time taken for epoch = 10.2924s
Epoch : 24 Test ELBO loss = 18.924. Time taken for epoch = 9.8302s
Epoch : 25 Test ELBO loss = 18.9124. Time taken for epoch = 9.9603s
Epoch : 26 Test ELBO loss = 18.9595. Time taken for epoch = 10.9887s
Epoch : 27 Test ELBO loss = 18.9256. Time taken for epoch = 10.1402s
Epoch : 28 Test ELBO loss = 18.8708. Time taken for epoch = 9.9109s
Epoch : 29 Test ELBO loss = 18.8602. Time taken for epoch = 10.3075s
Epoch : 30 Test ELBO loss = 18.8563. Time taken for epoch = 10.474s
Epoch : 31 Test ELBO loss = 18.8127. Time taken for epoch = 9.8779s
Epoch : 32 Test ELBO loss = 18.7989. Time taken for epoch = 9.6963s
Epoch : 33 Test ELBO loss = 18.8. Time taken for epoch = 9.8848s
Epoch : 34 Test ELBO loss = 18.8095. Time taken for epoch = 10.3168s
Epoch : 35 Test ELBO loss = 18.7601. Time taken for epoch = 10.8058s
Epoch : 36 Test ELBO loss = 18.7469. Time taken for epoch = 9.9365s
Epoch : 37 Test ELBO loss = 18.7049. Time taken for epoch = 10.0343s
Epoch : 38 Test ELBO loss = 18.7084. Time taken for epoch = 10.3214s
Epoch : 39 Test ELBO loss = 18.6858. Time taken for epoch = 10.3985s
Epoch : 40 Test ELBO loss = 18.7284. Time taken for epoch = 10.9685s
Epoch : 41 Test ELBO loss = 18.6574. Time taken for epoch = 10.5241s
Epoch : 42 Test ELBO loss = 18.6388. Time taken for epoch = 10.2392s
Epoch : 43 Test ELBO loss = 18.7133. Time taken for epoch = 9.8177s
Epoch : 44 Test ELBO loss = 18.6846. Time taken for epoch = 9.6858s
Epoch : 45 Test ELBO loss = 18.6001. Time taken for epoch = 9.5588s
Epoch : 46 Test ELBO loss = 18.5897. Time taken for epoch = 10.4554s
Epoch : 47 Test ELBO loss = 18.6184. Time taken for epoch = 10.0317s
Epoch : 48 Test ELBO loss = 18.6389. Time taken for epoch = 10.311s
Epoch : 49 Test ELBO loss = 18.5918. Time taken for epoch = 10.4506s
Epoch : 50 Test ELBO loss = 18.5081. Time taken for epoch = 9.9671s
```

## 可视化结果

要可视化和解释结果，请使用辅助可视化函数（第 3-0 页）。这些辅助函数在此示例的末尾定义。

**VisualizeReconstruction** 函数显示从每个类中随机选择的一个数字，并伴随显示数字通过自编码器后的重建版本。

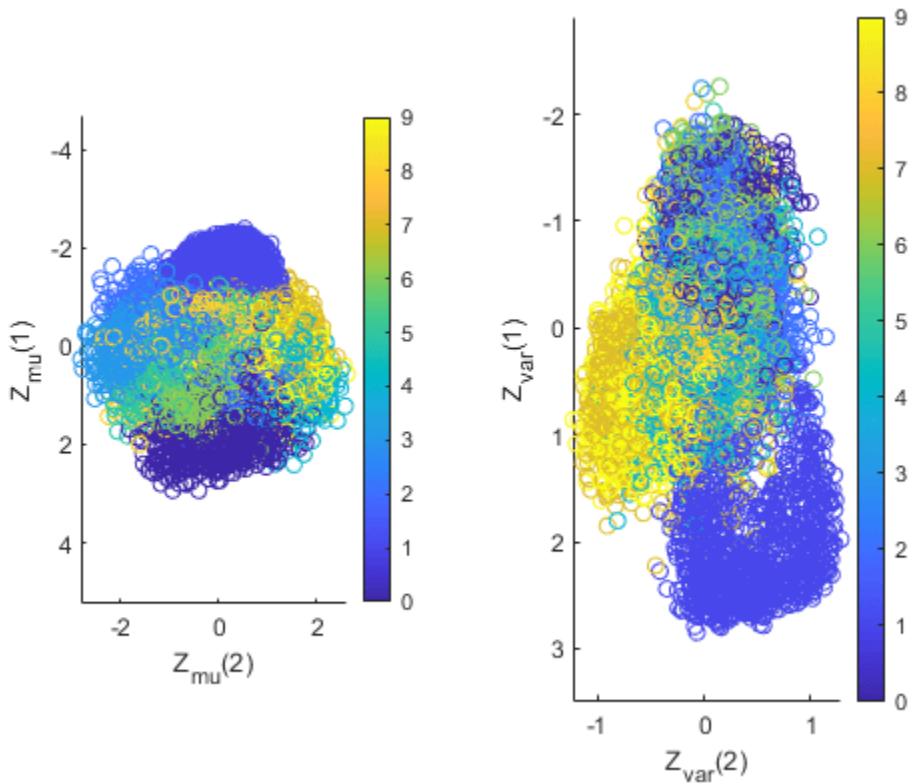
**VisualizeLatentSpace** 函数接受测试图像通过编码器网络后生成的均值和方差编码（每一个的维度都为 20），并对包含每个图像编码的矩阵执行主成分分析 (PCA)。然后，您可以在由两个第一主成分表征的两个维度中可视化由均值和方差定义的潜在空间。

**Generate** 函数初始化从正态分布采样的新编码，并输出这些编码通过解码器网络时生成的图像。

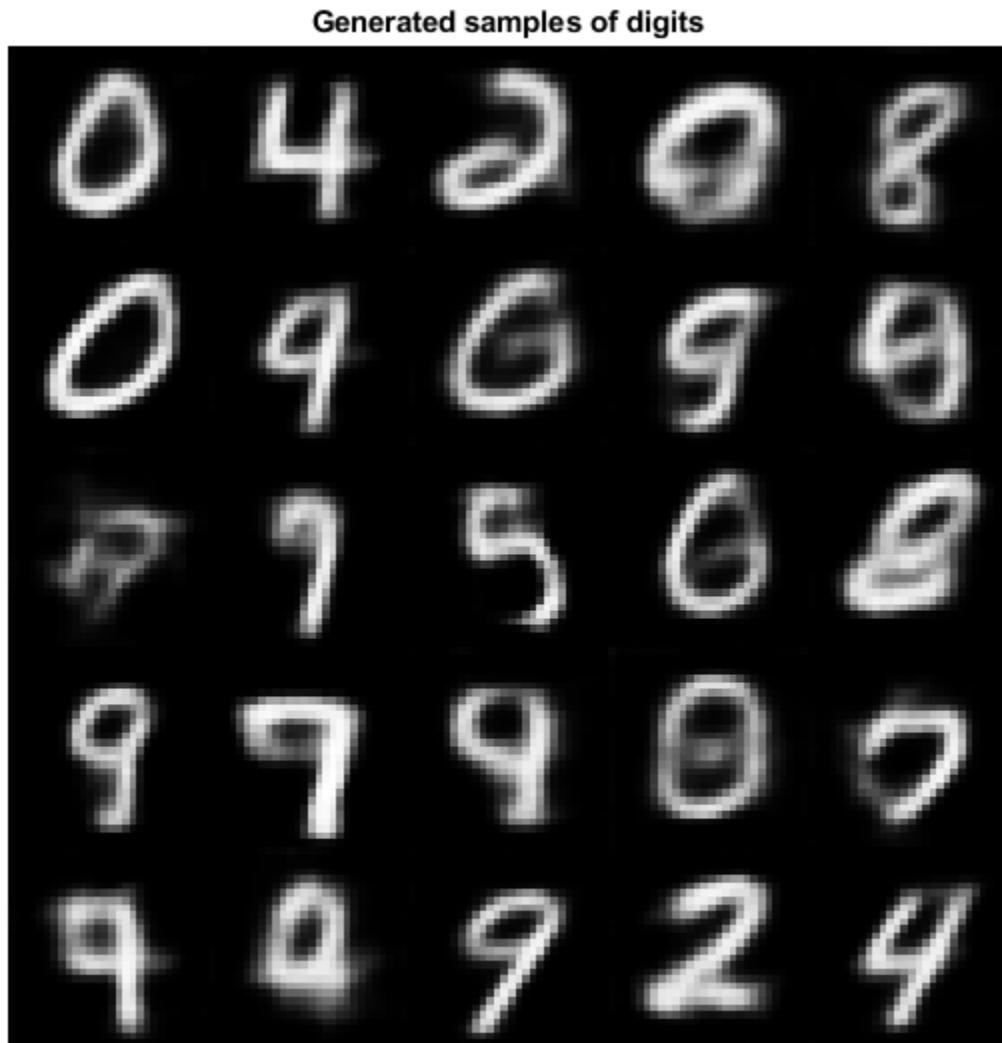
```
visualizeReconstruction(XTest, YTest, encoderNet, decoderNet)
```



```
visualizeLatentSpace(XTest, YTest, encoderNet)
```



```
generate(decoderNet, latentDim)
```



## 后续步骤

变分自编码器只是用于执行生成式任务的众多可用模型之一。它们适用于图像较小且具有清晰定义的特征的数据集（如 MNIST）。对于包含较大图像的更复杂的数据集，生成对抗网络 (GAN) 往往表现更好，生成的图像噪声更低。有关如何实现 GAN 以生成  $64 \times 64$  RGB 图像的示例，请参阅“训练生成对抗网络 (GAN)”（第 3-57 页）。

## 参考资料

- 1 LeCun, Y., C. Cortes, and C. J. C. Burges."The MNIST Database of Handwritten Digits." <http://yann.lecun.com/exdb/mnist/>.

## 辅助函数

### 模型梯度函数

**modelGradients** 函数接受编码器和解码器 **dlnetwork** 对象以及输入数据 **X** 的一个小批量，并返回损失关于网络中可学习参数的梯度。该函数执行三项操作：

- 1 对传入编码器网络进行处理的小批量图像调用 **sampling** 函数来获取编码。
- 2 传入编码，使之通过解码器网络并调用 **ELBOloss** 函数来获得损失。
- 3 通过调用 **dlgradient** 函数，计算损失关于两个网络的可学习参数的梯度。

```
function [infGrad, genGrad] = modelGradients(encoderNet, decoderNet, x)
[z, zMean, zLogvar] = sampling(encoderNet, x);
xPred = sigmoid(forward(decoderNet, z));
loss = ELBOloss(x, xPred, zMean, zLogvar);
[genGrad, infGrad] = dlgradient(loss, decoderNet.Learnables, ...
    encoderNet.Learnables);
end
```

### 采样和损失函数

**sampling** 函数从输入图像中获取编码。最初，它将一个小批量图像传入编码器网络进行处理，并将大小为  $(2 * \text{latentDim}) * \text{miniBatchSize}$  的输出拆分为均值矩阵和方差矩阵，每个矩阵的大小为  $\text{latentDim} * \text{batchSize}$ 。然后，它使用这些矩阵来实现重参数化方法并计算编码。最后，它将此编码转换为 **SSCB** 格式的 **dlarray** 对象。

```
function [zSampled, zMean, zLogvar] = sampling(encoderNet, x)
compressed = forward(encoderNet, x);
d = size(compressed,1)/2;
zMean = compressed(1:d,:);
zLogvar = compressed(1+d:end,:);

sz = size(zMean);
epsilon = randn(sz);
sigma = exp(.5 * zLogvar);
z = epsilon .* sigma + zMean;
z = reshape(z, [1,1,sz]);
zSampled = dlarray(z, 'SSCB');
end
```

**ELBOloss** 函数接受 **sampling** 函数返回的均值和方差的编码，并使用它们来计算 ELBO 损失。

```
function elbo = ELBOloss(x, xPred, zMean, zLogvar)
squares = 0.5*(xPred-x).^2;
reconstructionLoss = sum(squares, [1,2,3]);

KL = -.5 * sum(1 + zLogvar - zMean.^2 - exp(zLogvar), 1);

elbo = mean(reconstructionLoss + KL);
end
```

### 可视化函数

**VisualizeReconstruction** 函数为 MNIST 数据集的每个数字随机选择两个图像，将它们传入 VAE 进行处理，并绘制原始输入的图和重建的图。请注意，要绘制 **dlarray** 对象中包含的信息，您需要首先使用 **extractdata** 和 **gather** 函数提取它。

```

function visualizeReconstruction(XTest,YTest, encoderNet, decoderNet)
f = figure;
figure(f)
title("Example ground truth image vs. reconstructed image")
for i = 1:2
    for c=0:9
        idx = iRandomIdxOfClass(YTest,c);
        X = XTest(:,:,:,:idx);

        [z, ~, ~] = sampling(encoderNet, X);
        XPred = sigmoid(forward(decoderNet, z));

        X = gather(extractdata(X));
        XPred = gather(extractdata(XPred));

        comparison = [X, ones(size(X,1),1), XPred];
        subplot(4,5,(i-1)*10+c+1), imshow(comparison,[]);
    end
end
end

function idx = iRandomIdxOfClass(T,c)
idx = T == categorical(c);
idx = find(idx);
idx = idx(randi(numel(idx),1));
end

```

**VisualizeLatentSpace** 函数可视化由构成编码器网络输出的均值和方差矩阵定义的潜在空间，并定位由每个数字的潜在空间表示形成的簇。

该函数首先从 **dlarray** 对象中提取均值矩阵和方差矩阵。由于无法转置具有通道/批量维度 (C 和 B) 的矩阵，函数会在转置矩阵之前调用 **stripdims**。然后，它对两个矩阵进行主成分分析 (PCA)。为了在两个维度上可视化潜在空间，该函数保留前两个主成分，并绘制彼此对照的图。最后，该函数对数字类进行着色，以便您可以观察簇。

```

function visualizeLatentSpace(XTest, YTest, encoderNet)
[~, zMean, zLogvar] = sampling(encoderNet, XTest);

zMean = stripdims(zMean)';
zMean = gather(extractdata(zMean));

zLogvar = stripdims(zLogvar)';
zLogvar = gather(extractdata(zLogvar));

[~,scoreMean] = pca(zMean);
[~,scoreLogvar] = pca(zLogvar);

c = parula(10);
f1 = figure;
figure(f1)
title("Latent space")

ah = subplot(1,2,1);
scatter(scoreMean(:,2),scoreMean(:,1),[],c(double(YTest),:));
ah.YDir = 'reverse';
axis equal
xlabel("Z_m_u(2)")
ylabel("Z_m_u(1)")

```

```
cb = colorbar; cb.Ticks = 0:(1/9):1; cb.TickLabels = string(0:9);

ah = subplot(1,2,2);
scatter(scoreLogvar(:,2),scoreLogvar(:,1),[],c(double(YTest),:));
ah.YDir = 'reverse';
xlabel("Z_v_a_r(2)")
ylabel("Z_v_a_r(1)")
cb = colorbar; cb.Ticks = 0:(1/9):1; cb.TickLabels = string(0:9);
axis equal
end
```

`generate` 函数测试 VAE 的生成能力。它初始化一个包含 25 个随机生成的编码的 `dlarray` 对象，将这些编码传入解码器网络进行处理，并绘制输出。

```
function generate(decoderNet, latentDim)
randomNoise = dlarray(randn(1,1,latentDim,25),'SSCB');
generatedImage = sigmoid(predict(decoderNet, randomNoise));
generatedImage = extractdata(generatedImage);

f3 = figure;
figure(f3)
imshow(imtile(generatedImage, "ThumbnailSize", [100,100]))
title("Generated samples of digits")
drawnow
end
```

## 另请参阅

`dlnetwork` | `layerGraph` | `dlarray` | `adamupdate` | `dlfeval` | `dlgradient` | `sigmoid`

## 详细信息

- “训练生成对抗网络 (GAN)”（第 3-57 页）
- “定义自定义训练循环、损失函数和网络”（第 17-33 页）
- “Make Predictions Using Model Function”
- “Specify Training Options in Custom Training Loop”
- “Automatic Differentiation Background”

# 时序、序列和文本深度学习

---

- “使用深度学习进行序列分类” (第 4-2 页)
- “使用深度学习进行时序预测” (第 4-9 页)
- “使用深度学习进行语音命令识别” (第 4-16 页)
- “使用深度学习进行“序列到序列”分类” (第 4-33 页)
- “使用深度学习进行“序列到序列”回归” (第 4-38 页)
- “使用深度学习对视频进行分类” (第 4-45 页)
- “使用深度学习对文本数据进行分类” (第 4-55 页)
- “使用卷积神经网络对文本数据进行分类” (第 4-63 页)
- “使用深度学习生成文本” (第 4-72 页)
- “《傲慢与偏见》与 MATLAB” (第 4-78 页)
- “使用深度学习进行逐单词文本生成” (第 4-83 页)

## 使用深度学习进行序列分类

此示例说明如何使用长短期记忆 (LSTM) 网络对序列数据进行分类。

要训练深度神经网络以对序列数据进行分类，可以使用 LSTM 网络。LSTM 网络允许您将序列数据输入网络，并根据序列数据的各个时间步进行预测。

此示例使用 [1] 和 [2] 中所述的日语元音数据集。此示例训练一个 LSTM 网络，旨在根据表示连续说出的两个日语元音的时序数据来识别说话者。训练数据包含九个说话者的时序数据。每个序列有 12 个特征，且长度不同。该数据集包含 270 个训练观测值和 370 个测试观测值。

### 加载序列数据

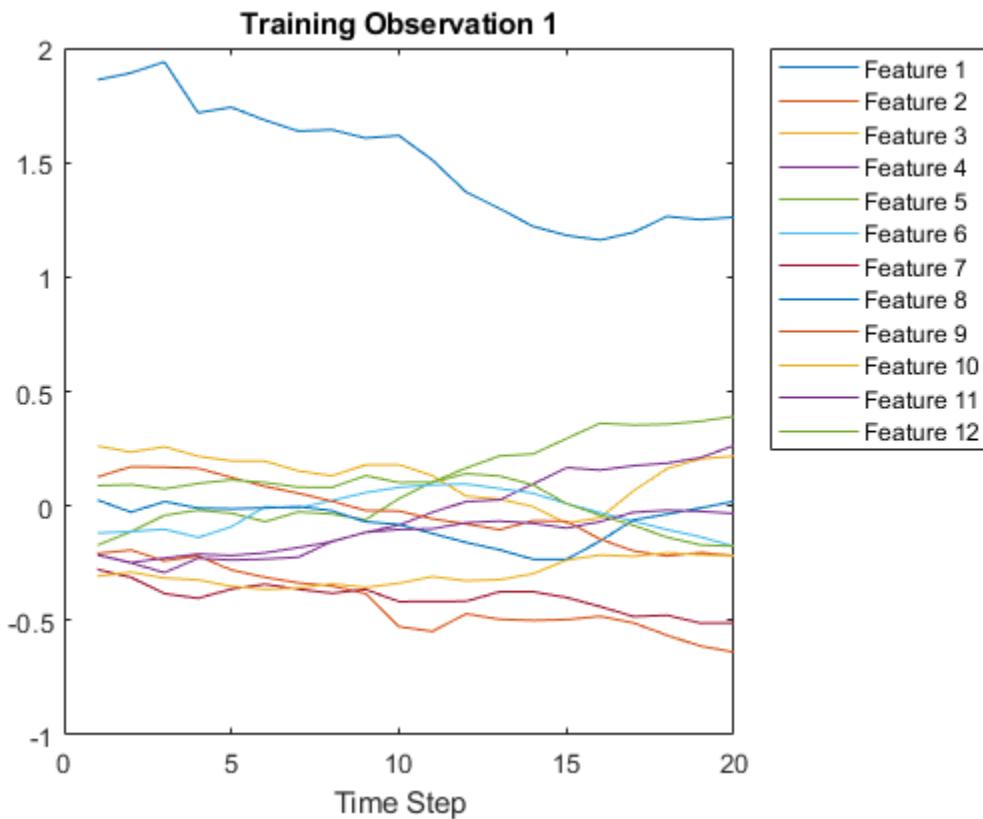
加载日语元音训练数据。**XTrain** 是包含 270 个不同长度的 12 维序列的元胞数组。**Y** 是对应于九个说话者的标签 "1"、"2"、...、"9" 的分类向量。**XTrain** 中的条目是具有 12 行（每个特征一行）和不同列数（每个时间步一列）的矩阵。

```
[XTrain,YTrain] = japaneseVowelsTrainData;
XTrain(1:5)
```

```
ans=5×1 cell array
{12×20 double}
{12×26 double}
{12×22 double}
{12×20 double}
{12×21 double}
```

在绘图中可视化第一个时序。每行对应一个特征。

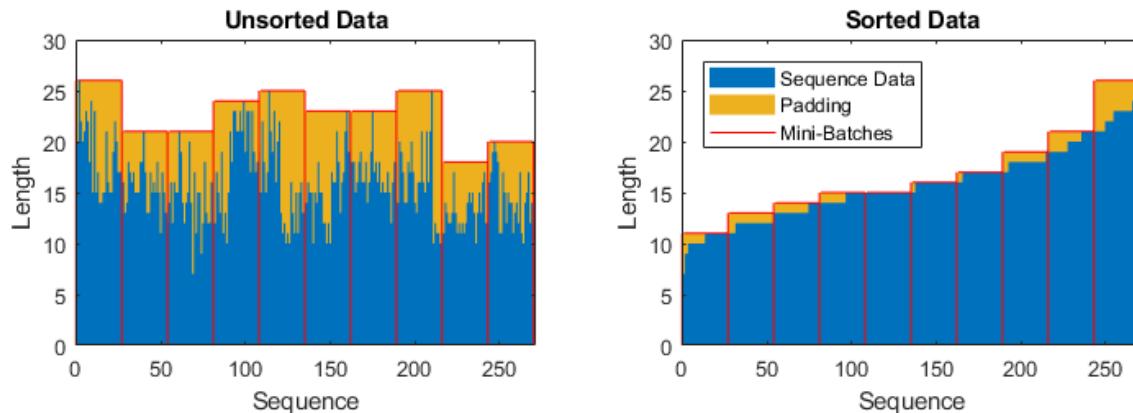
```
figure
plot(XTrain{1})
xlabel("Time Step")
title("Training Observation 1")
numFeatures = size(XTrain{1},1);
legend("Feature " + string(1:numFeatures), 'Location','northeastoutside')
```



### 准备要填充的数据

在训练过程中，默认情况下，软件将训练数据拆分成小批量并填充序列，使它们具有相同的长度。过多填充会对网络性能产生负面影响。

为了防止训练过程添加过多填充，您可以按序列长度对训练数据进行排序，并选择合适的小批量大小，以使同一个小批量中的序列长度相近。下图显示了对数据进行排序之前和之后填充序列的效果。



获取每个观测值的序列长度。

```
numObservations = numel(XTrain);
for i=1:numObservations
```

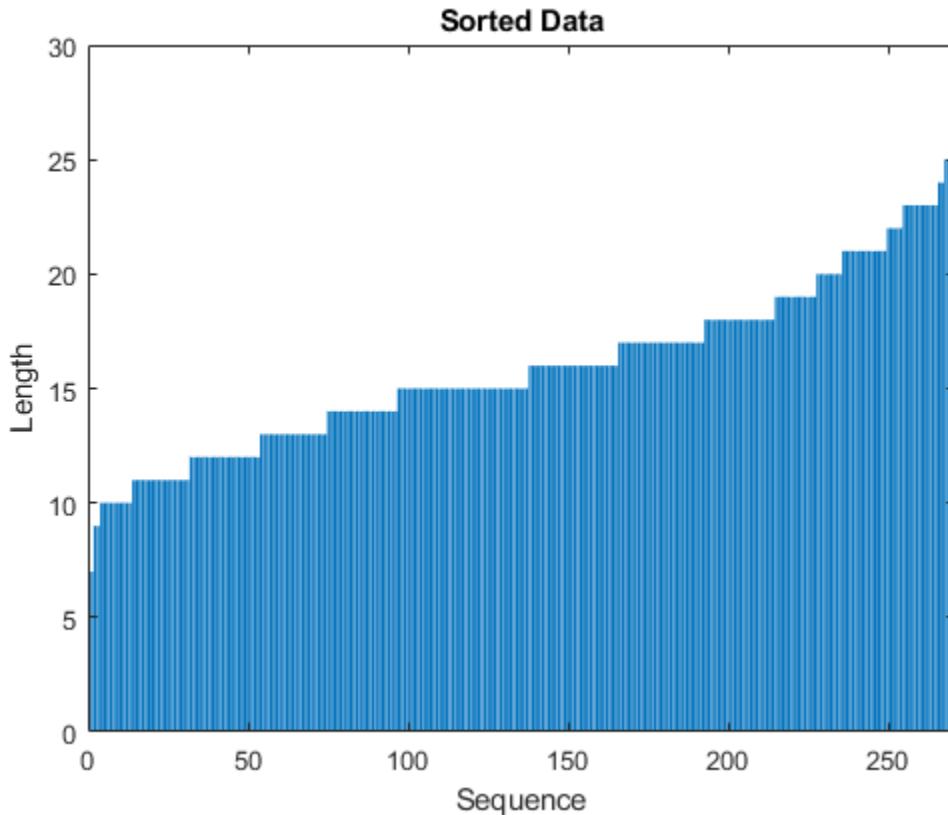
```
sequence = XTrain{i};  
sequenceLengths(i) = size(sequence,2);  
end
```

按序列长度对数据进行排序。

```
[sequenceLengths,idx] = sort(sequenceLengths);  
XTrain = XTrain(idx);  
YTrain = YTrain(idx);
```

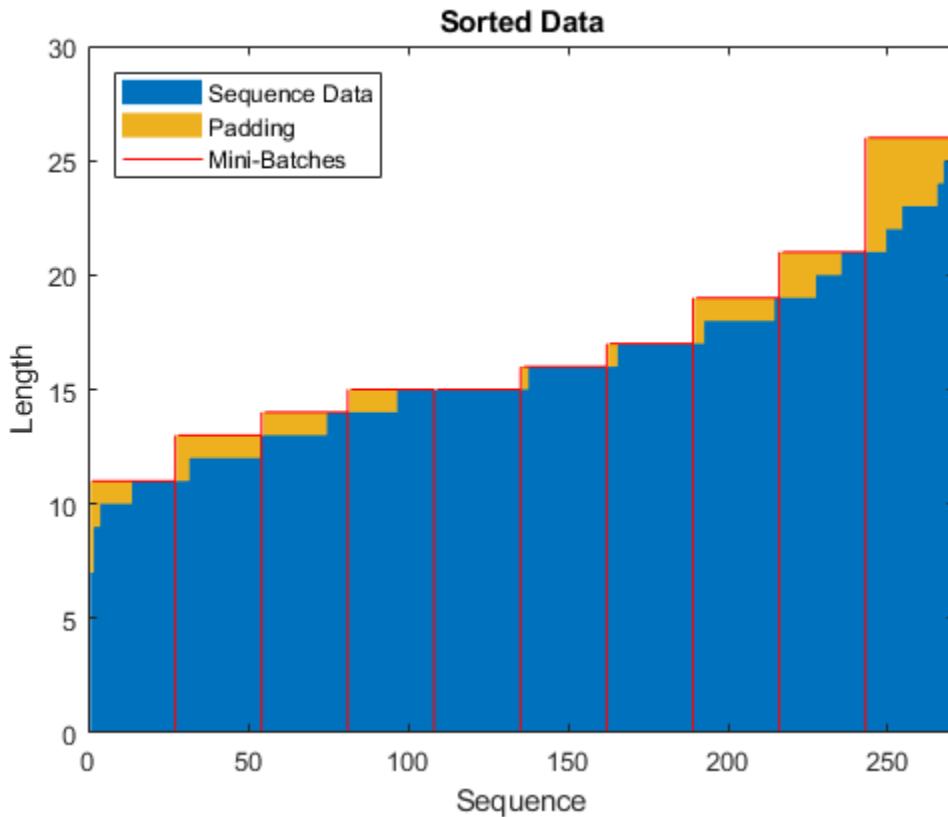
在条形图中查看排序的序列长度。

```
figure  
bar(sequenceLengths)  
ylim([0 30])  
xlabel("Sequence")  
ylabel("Length")  
title("Sorted Data")
```



选择小批量大小 27 以均匀划分训练数据，并减少小批量中的填充量。下图说明了添加到序列中的填充。

```
miniBatchSize = 27;
```



## 定义 LSTM 网络架构

定义 LSTM 网络架构。将输入大小指定为序列大小 12（输入数据的维度）。指定具有 100 个隐含单元的双向 LSTM 层，并输出序列的最后一个元素。最后，通过包含大小为 9 的全连接层，后跟 softmax 层和分类层，来指定九个类。

如果您可以在预测时访问完整序列，则可以在网络中使用双向 LSTM 层。双向 LSTM 层在每个时间步从完整序列学习。如果您不能在预测时访问完整序列，例如，您正在预测值或一次预测一个时间步时，则改用 LSTM 层。

```

inputSize = 12;
numHiddenUnits = 100;
numClasses = 9;

layers = [ ...
    sequenceInputLayer(inputSize)
    bilSTMLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]

layers =
5×1 Layer array with layers:

1 " Sequence Input      Sequence input with 12 dimensions
2 " BiLSTM              BiLSTM with 100 hidden units
3 " Fully Connected     9 fully connected layer

```

```

4 " Softmax      softmax
5 " Classification Output  crossentropyex

```

现在，指定训练选项。指定求解器为 'adam'，梯度阈值为 1，最大轮数为 100。要减少小批量中的填充量，请选择 27 作为小批量大小。要填充数据以使长度与最长序列相同，请将序列长度指定为 'longest'。要确保数据保持按序列长度排序的状态，请指定从不打乱数据。

由于小批量数据存储较小且序列较短，因此更适合在 CPU 上训练。将 'ExecutionEnvironment' 指定为 'cpu'。要在 GPU (如果可用) 上进行训练，请将 'ExecutionEnvironment' 设置为 'auto' (这是默认值)。

```

maxEpochs = 100;
miniBatchSize = 27;

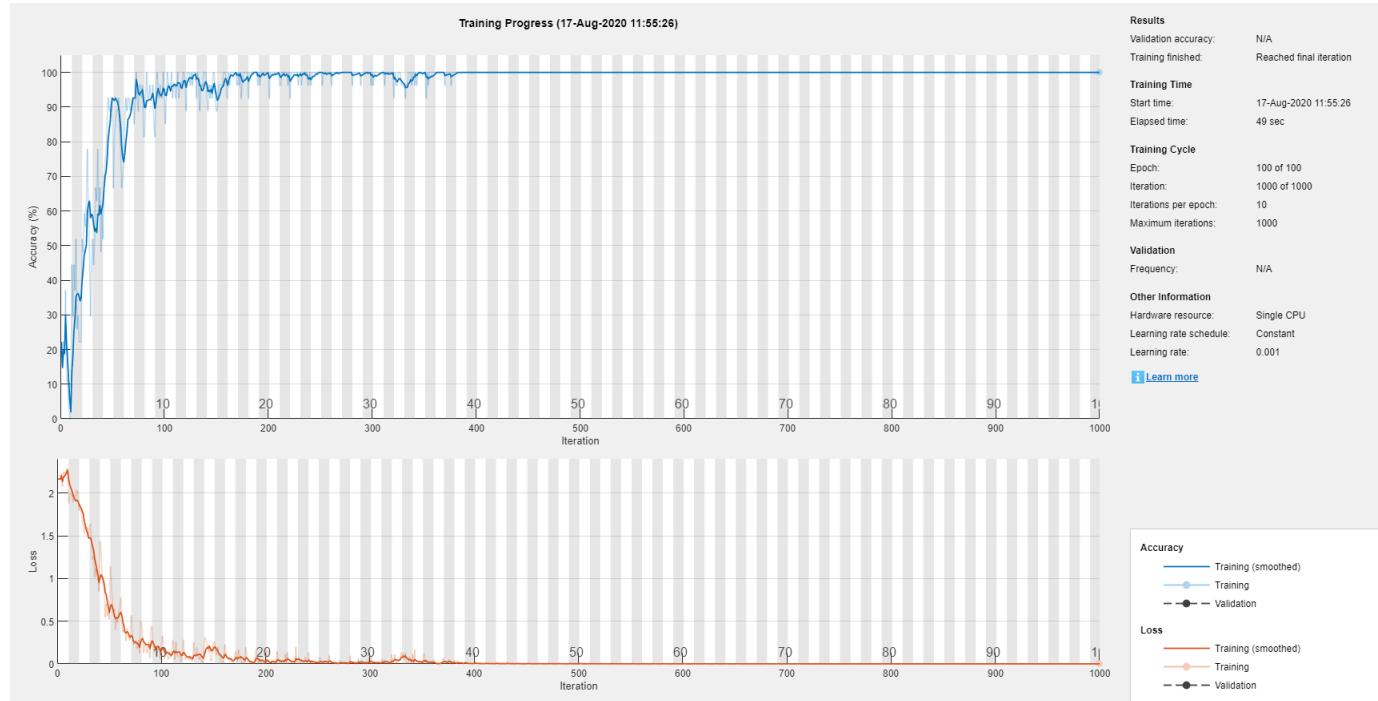
options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'GradientThreshold',1, ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','longest', ...
    'Shuffle','never', ...
    'Verbose',0, ...
    'Plots','training-progress');

```

## 训练 LSTM 网络

使用 `trainNetwork` 以指定的训练选项训练 LSTM 网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



## 测试 LSTM 网络

加载测试集并将序列分类到不同的说话者。

加载日语元音测试数据。XTest 是包含 370 个不同长度的 12 维序列的元胞数组。YTest 是由对应于九个说话者的标签 "1"、"2"、...、"9" 组成的分类向量。

```
[XTest,YTest] = japaneseVowelsTestData;
XTest(1:3)
```

```
ans=3×1 cell array
{12×19 double}
{12×17 double}
{12×19 double}
```

LSTM 网络 net 已使用相似长度的小批量序列进行训练。确保以相同的方式组织测试数据。按序列长度对测试数据进行排序。

```
numObservationsTest = numel(XTest);
for i=1:numObservationsTest
    sequence = XTest{i};
    sequenceLengthsTest(i) = size(sequence,2);
end
[sequenceLengthsTest,idx] = sort(sequenceLengthsTest);
XTest = XTest(idx);
YTest = YTest(idx);
```

对测试数据进行分类。要减少分类过程中引入的填充量，请将小批量大小设置为 27。要应用与训练数据相同的填充，请将序列长度指定为 'longest'。

```
miniBatchSize = 27;
YPred = classify(net,XTest, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','longest');
```

计算预测值的分类准确度。

```
acc = sum(YPred == YTest)./numel(YTest)
acc = 0.9730
```

## 参考

- [1] M. Kudo, J. Toyama, and M. Shimbo. "Multidimensional Curve Classification Using Passing-Through Regions." *Pattern Recognition Letters*. Vol. 20, No. 11–13, pages 1103–1111.
- [2] UCI Machine Learning Repository: Japanese Vowels Dataset. <https://archive.ics.uci.edu/ml/datasets/Japanese+Vowels>

## 另请参阅

`trainNetwork` | `trainingOptions` | `lstmLayer` | `bilstmLayer` | `sequenceInputLayer`

## 相关示例

- “使用深度学习进行时序预测”（第 4-9 页）
- “使用深度学习进行“序列到序列”分类”（第 4-33 页）
- “使用深度学习进行“序列到序列”回归”（第 4-38 页）

- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 使用深度学习进行时序预测

此示例说明如何使用长期短期记忆 (LSTM) 网络预测时序数据。

要预测序列在将来时间步的值，您可以训练“序列到序列”回归 LSTM 网络，其中响应是将值移位了一个时间步的训练序列。也就是说，在输入序列的每个时间步，LSTM 网络都学习预测下一个时间步的值。

要预测将来多个时间步的值，请使用 `predictAndUpdateState` 函数一次预测一个时间步，并在每次预测时更新网络状态。

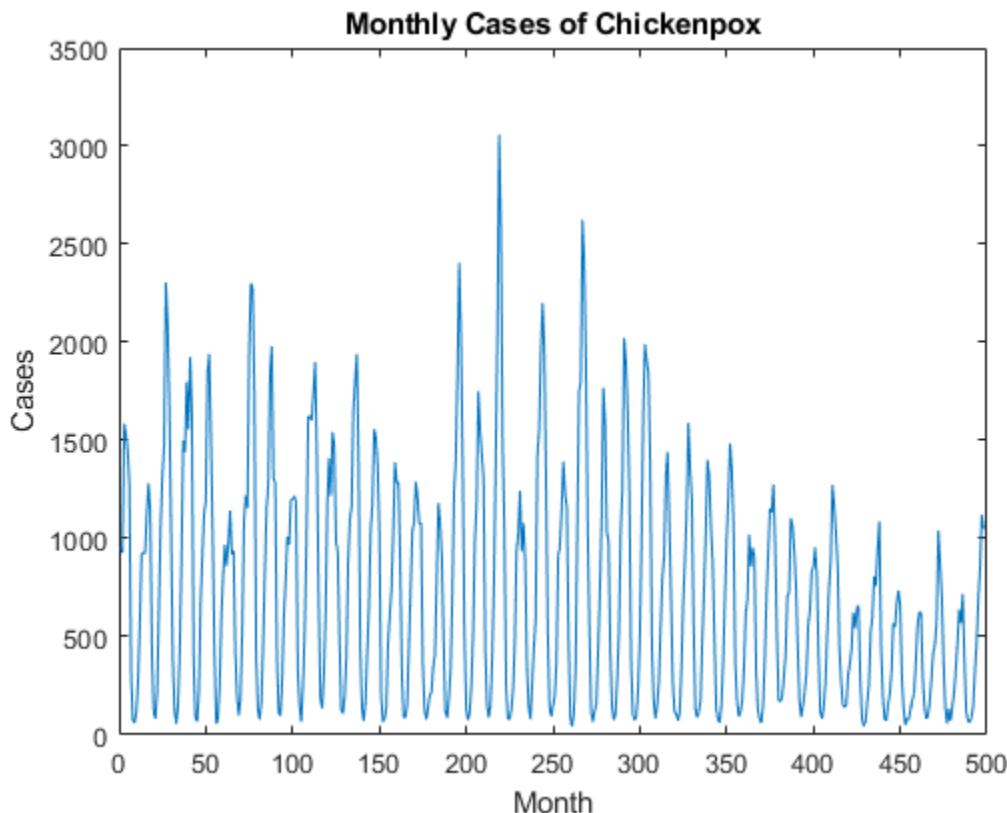
此示例使用数据集 `chickenpox_dataset`。该示例训练一个 LSTM 网络，旨在根据前几个月的水痘病例数来预测未来的水痘病例数。

## 加载序列数据

加载示例数据。`chickenpox_dataset` 包含一个时序，其时间步对应于月份，值对应于病例数。输出是一个元胞数组，其中每个元素均为单一时间步。将数据重构为行向量。

```
data = chickenpox_dataset;
data = [data{:}];

figure
plot(data)
xlabel("Month")
ylabel("Cases")
title("Monthly Cases of Chickenpox")
```



对训练数据和测试数据进行分区。序列的前 90% 用于训练，后 10% 用于测试。

```
numTimeStepsTrain = floor(0.9*numel(data));  
  
dataTrain = data(1:numTimeStepsTrain+1);  
dataTest = data(numTimeStepsTrain+1:end);
```

### 标准化数据

为了获得较好的拟合并防止训练发散，将训练数据标准化为具有零均值和单位方差。在预测时，您必须使用与训练数据相同的参数来标准化测试数据。

```
mu = mean(dataTrain);  
sig = std(dataTrain);  
  
dataTrainStandardized = (dataTrain - mu) / sig;
```

### 准备预测变量和响应

要预测序列在将来时间步的值，请将响应指定为将值移位了一个时间步的训练序列。也就是说，在输入序列的每个时间步，LSTM 网络都学习预测下一个时间步的值。预测变量是没有最终时间步的训练序列。

```
XTrain = dataTrainStandardized(1:end-1);  
YTrain = dataTrainStandardized(2:end);
```

### 定义 LSTM 网络架构

创建 LSTM 回归网络。指定 LSTM 层有 200 个隐含单元。

```
numFeatures = 1;  
numResponses = 1;  
numHiddenUnits = 200;  
  
layers = [ ...  
    sequenceInputLayer(numFeatures)  
    lstmLayer(numHiddenUnits)  
    fullyConnectedLayer(numResponses)  
    regressionLayer];
```

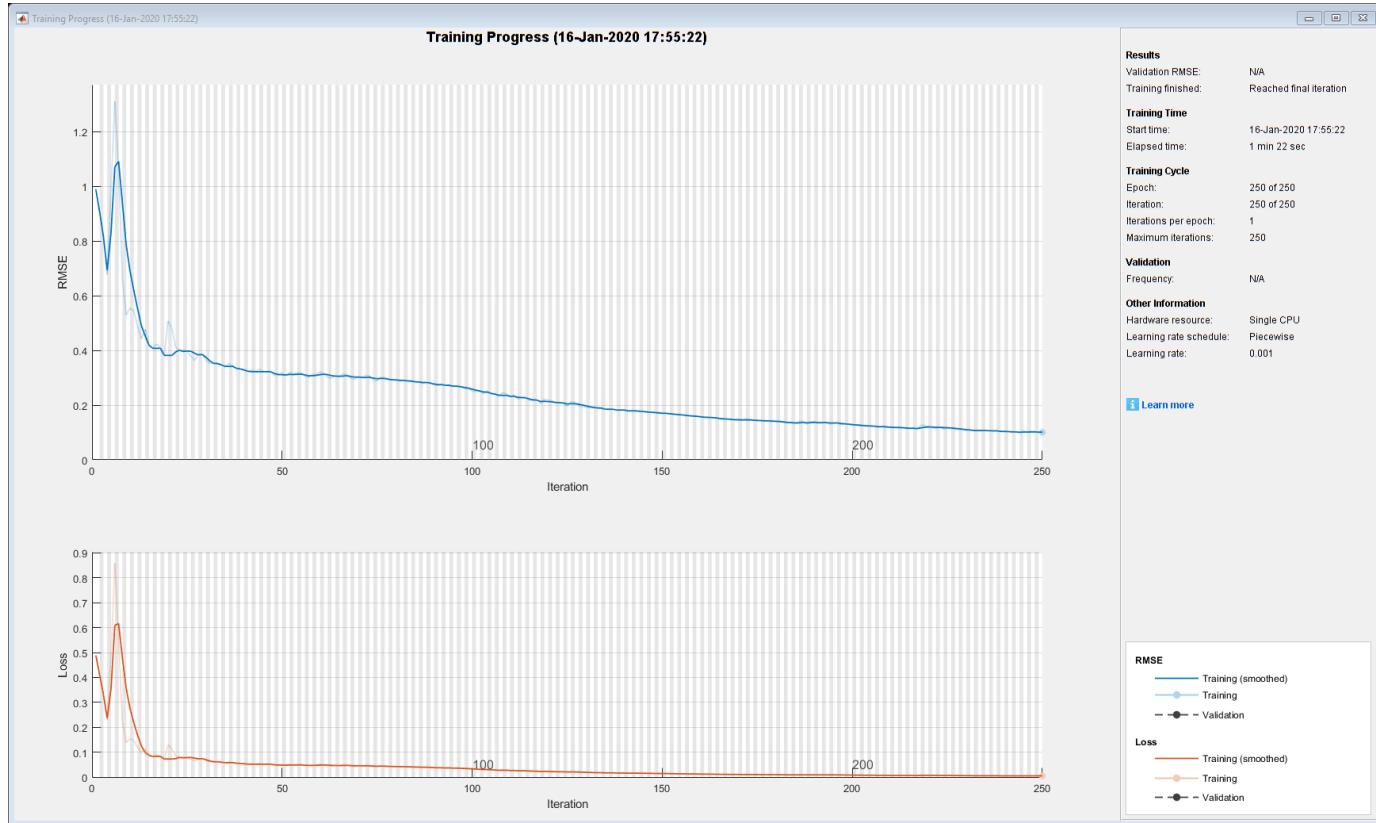
指定训练选项。将求解器设置为 'adam' 并进行 250 轮训练。要防止梯度爆炸，请将梯度阈值设置为 1。指定初始学习率 0.005，在 125 轮训练后通过乘以因子 0.2 来降低学习率。

```
options = trainingOptions('adam', ...  
    'MaxEpochs',250, ...  
    'GradientThreshold',1, ...  
    'InitialLearnRate',0.005, ...  
    'LearnRateSchedule','piecewise', ...  
    'LearnRateDropPeriod',125, ...  
    'LearnRateDropFactor',0.2, ...  
    'Verbose',0, ...  
    'Plots','training-progress');
```

### 训练 LSTM 网络

使用 `trainNetwork` 以指定的训练选项训练 LSTM 网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



## 预测将来时间步

要预测将来多个时间步的值，请使用 `predictAndUpdateState` 函数一次预测一个时间步，并在每次预测时更新网络状态。对于每次预测，使用前一次预测作为函数的输入。

使用与训练数据相同的参数来标准化测试数据。

```
dataTestStandardized = (dataTest - mu) / sig;
XTest = dataTestStandardized(1:end-1);
```

要初始化网络状态，请先对训练数据 `XTrain` 进行预测。接下来，使用训练响应的最后一个时间步 `YTrain(end)` 进行第一次预测。循环其余预测并将前一次预测输入到 `predictAndUpdateState`。

对于大型数据集合、长序列或大型网络，在 GPU 上进行预测计算通常比在 CPU 上快。其他情况下，在 CPU 上进行预测计算通常更快。对于单时间步预测，请使用 CPU。要使用 CPU 进行预测，请将 `predictAndUpdateState` 的 'ExecutionEnvironment' 选项设置为 'cpu'。

```
net = predictAndUpdateState(net,XTrain);
[net,YPred] = predictAndUpdateState(net,YTrain(end));

numTimeStepsTest = numel(XTest);
for i = 2:numTimeStepsTest
    [net,YPred(:,i)] = predictAndUpdateState(net,YPred(:,i-1),'ExecutionEnvironment','cpu');
end
```

使用先前计算的参数对预测去标准化。

```
YPred = sig*YPred + mu;
```

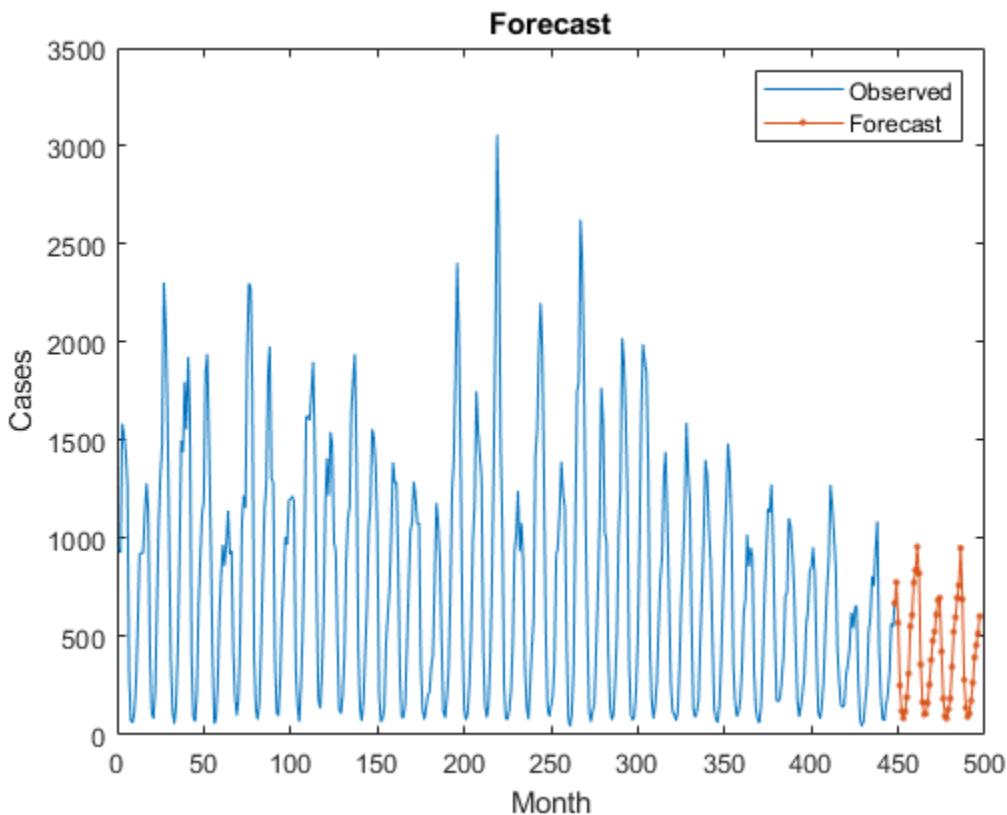
训练进度图会报告根据标准化数据计算出的均方根误差 (RMSE)。根据去标准化的预测值计算 RMSE。

```
YTest = dataTest(2:end);
rmse = sqrt(mean((YPred-YTest).^2))

rmse = single
248.5531
```

使用预测值绘制训练时序。

```
figure
plot(dataTrain(1:end-1))
hold on
idx = numTimeStepsTrain:(numTimeStepsTrain+numTimeStepsTest);
plot(idx,[data(numTimeStepsTrain) YPred],'.-')
hold off
xlabel("Month")
ylabel("Cases")
title("Forecast")
legend(["Observed" "Forecast"])
```



将预测值与测试数据进行比较。

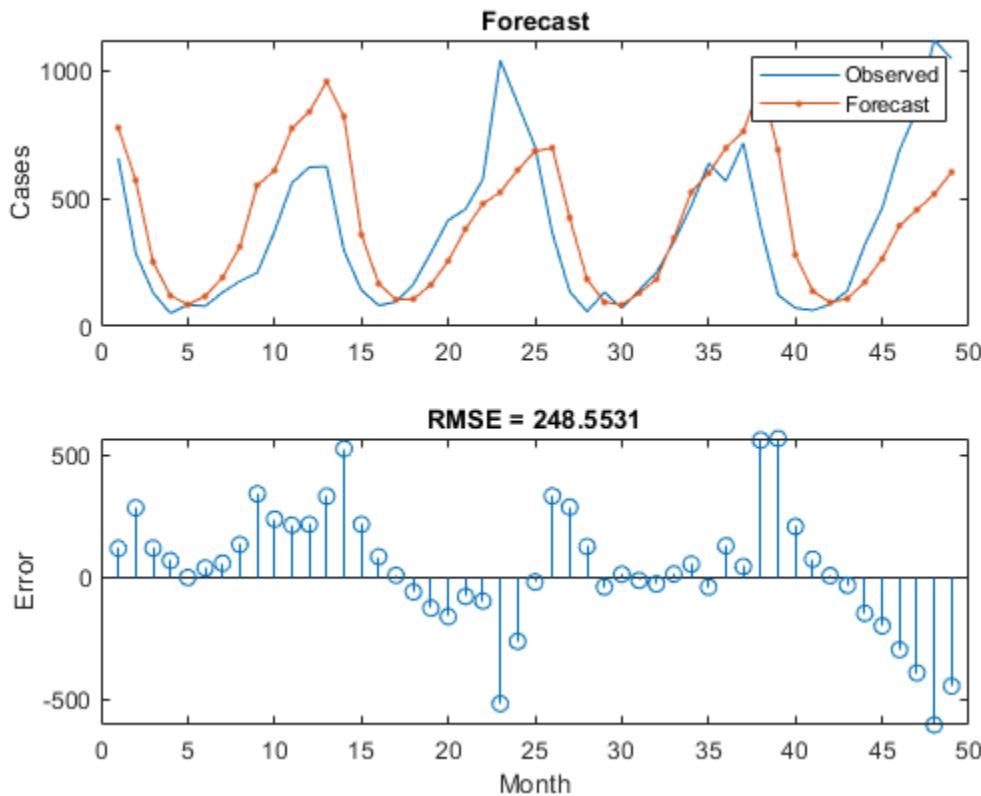
```
figure
subplot(2,1,1)
plot(YTest)
hold on
plot(YPred,'.-')
```

```

hold off
legend(["Observed" "Forecast"])
ylabel("Cases")
title("Forecast")

subplot(2,1,2)
stem(YPred - YTest)
xlabel("Month")
ylabel("Error")
title("RMSE = " + rmse)

```



### 使用观测值更新网络状态

如果您可以访问预测之间的时间步的实际值，则可以使用观测值而不是预测值更新网络状态。

首先，初始化网络状态。要对新序列进行预测，请使用 `resetState` 重置网络状态。重置网络状态可防止先前的预测影响对新数据的预测。重置网络状态，然后通过对训练数据进行预测来初始化网络状态。

```

net = resetState(net);
net = predictAndUpdateState(net,XTrain);

```

对每个时间步进行预测。对于每次预测，使用前一时间步的观测值预测下一个时间步。将 `predictAndUpdateState` 的 `'ExecutionEnvironment'` 选项设置为 `'cpu'`。

```

YPred = [];
numTimeStepsTest = numel(XTest);
for i = 1:numTimeStepsTest

```

```
[net,YPred(:,i)] = predictAndUpdateState(net,XTest(:,i),'ExecutionEnvironment','cpu');  
end
```

使用先前计算的参数对预测去标准化。

```
YPred = sig*YPred + mu;
```

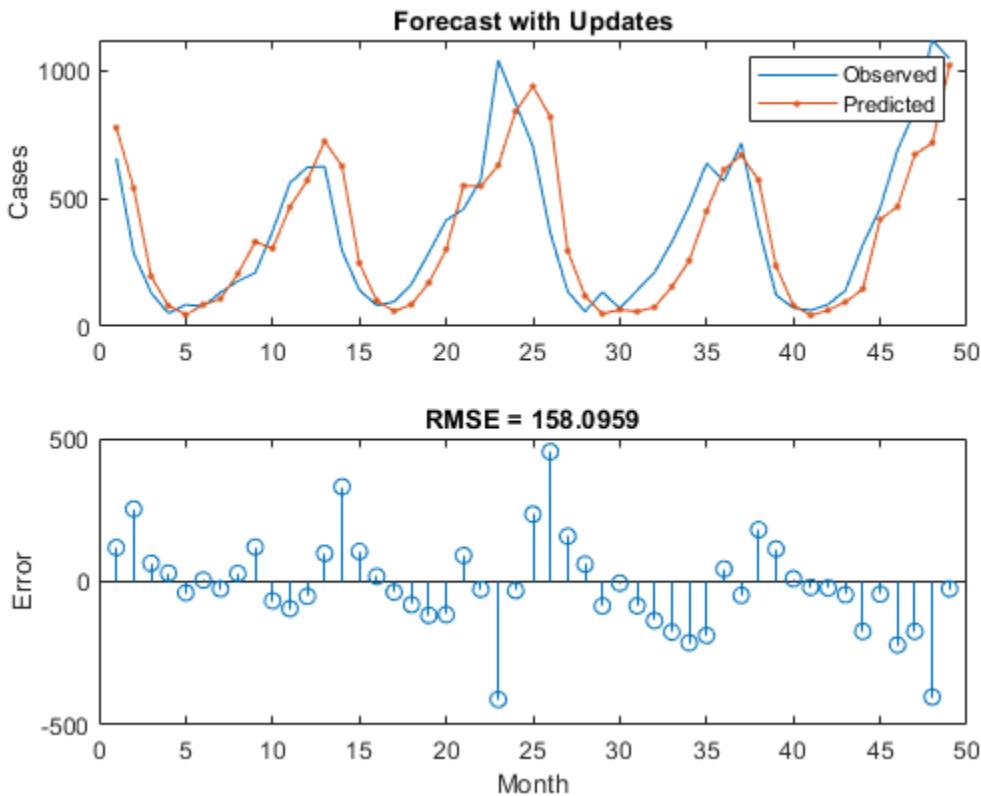
计算均方根误差 (RMSE)。

```
rmse = sqrt(mean((YPred-YTest).^2))
```

```
rmse = 158.0959
```

将预测值与测试数据进行比较。

```
figure  
subplot(2,1,1)  
plot(YTest)  
hold on  
plot(YPred,'.-')  
hold off  
legend(["Observed" "Predicted"])  
ylabel("Cases")  
title("Forecast with Updates")  
  
subplot(2,1,2)  
stem(YPred - YTest)  
xlabel("Month")  
ylabel("Error")  
title("RMSE = " + rmse)
```



这里，当使用观测值而不是预测值更新网络状态时，预测更准确。

## 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [lstmLayer](#) | [sequenceInputLayer](#)

## 相关示例

- “使用深度学习生成文本” (第 4-72 页)
- “使用深度学习进行序列分类” (第 4-2 页)
- “使用深度学习进行“序列到序列”分类” (第 4-33 页)
- “使用深度学习进行“序列到序列”回归” (第 4-38 页)
- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 使用深度学习进行语音命令识别

此示例说明如何训练一个深度学习模型来检测音频中是否存在语音命令。此示例使用语音命令数据集 [1] 来训练卷积神经网络，以识别给定的一组命令。

要从头开始训练网络，您必须首先下载数据集。如果您不想下载数据集或训练网络，则可以加载随此示例提供的一个预训练网络，并执行此示例的接下来的两个部分：通过预训练网络识别命令和使用来自麦克风的流音频检测命令。

### 通过预训练网络识别命令

在进入详细的训练过程之前，您将使用一个预训练语音识别网络来识别语音命令。

加载该预训练网络。

```
load('commandNet.mat')
```

该网络经过训练以识别下列语音命令：

- "yes"
- "no"
- "up"
- "down"
- "left"
- "right"
- "on"
- "off"
- "stop"
- "go"

在有人说 "stop" 的位置加载一个简短语音信号。

```
[x,fs] = audioread('stop_command.flac');
```

收听命令。

```
sound(x,fs)
```

预训练网络将基于听觉的频谱图作为输入。您首先将语音波形转换为基于听觉的频谱图。

使用函数 `extractAuditoryFeature` 计算听觉频谱图。在示例的后面部分，您将详细了解特征提取。

```
auditorySpect = helperExtractAuditoryFeatures(x,fs);
```

根据听觉频谱图对命令进行分类。

```
command = classify(trainedNet,auditorySpect)
```

```
command =
```

```
categorical
```

```
stop
```

该网络经过训练以将不属于该集合的单词分类为 “unknown” 。

现在，您将对命令列表中未包含的一个单词 (“play”) 分类以进行识别。

加载并收听语音信号。

```
x = audioread('play_command.flac');
sound(x,fs)
```

计算听觉频谱图。

```
auditorySpect = helperExtractAuditoryFeatures(x,fs);
```

对信号进行分类。

```
command = classify(trainedNet,auditorySpect)
```

```
command =
```

```
categorical
```

```
unknown
```

网络经过训练以将背景噪声分类为 “background” 。

创建由随机噪声组成的时长为一秒的信号。

```
x = pinknoise(16e3);
```

计算听觉频谱图。

```
auditorySpect = helperExtractAuditoryFeatures(x,fs);
```

对背景噪声进行分类。

```
command = classify(trainedNet,auditorySpect)
```

```
command =
```

```
categorical
```

```
background
```

## 使用来自麦克风的流音频检测命令

基于来自麦克风的流音频测试预训练的命令检测网络。尝试说出其中一个命令，例如 yes、no 或 stop。然后，尝试说一个未知的单词，如 Marvin、Sheila、bed、house、cat、bird 或从 0 到 9 的任意数字。

指定分类率（以 Hz 为单位），并创建一个可以从麦克风读取音频的音频设备读取器。

```
classificationRate = 20;
adr = audioDeviceReader('SampleRate',fs,'SamplesPerFrame',floor(fs/classificationRate));
```

初始化一个音频缓冲区。提取网络的分类标签。分别为流音频标签和分类概率初始化时长半秒的缓冲区。通过这些缓冲区来比较较长时间内的分类结果，并就是否检测到了命令达成“一致”。指定决策逻辑的阈值。

```
audioBuffer = dsp.AsyncBuffer(fs);

labels = trainedNet.Layers(end).Classes;
YBuffer(1:classificationRate/2) = categorical("background");

probBuffer = zeros([numel(labels),classificationRate/2]);

countThreshold = ceil(classificationRate*0.2);
probThreshold = 0.7;
```

创建一个图窗，在图窗存在期间一直检测命令。要无限期运行循环，请将 timeLimit 设置为 Inf。要停止实时检测，只需关闭图窗。

```
h = figure('Units','normalized','Position',[0.2 0.1 0.6 0.8]);

timeLimit = 20;

tic
while ishandle(h) && toc < timeLimit

    % Extract audio samples from the audio device and add the samples to
    % the buffer.
    x = adr();
    write(audioBuffer,x);
    y = read(audioBuffer,fs,fs-adr.SamplesPerFrame);

    spec = helperExtractAuditoryFeatures(y,fs);

    % Classify the current spectrogram, save the label to the label buffer,
    % and save the predicted probabilities to the probability buffer.
    [YPredicted,probs] = classify(trainedNet,spec,'ExecutionEnvironment','cpu');
    YBuffer = [YBuffer(2:end),YPredicted];
    probBuffer = [probBuffer(:,2:end),probs(:)];

    % Plot the current waveform and spectrogram.
    subplot(2,1,1)
    plot(y)
    axis tight
    ylim([-1,1])

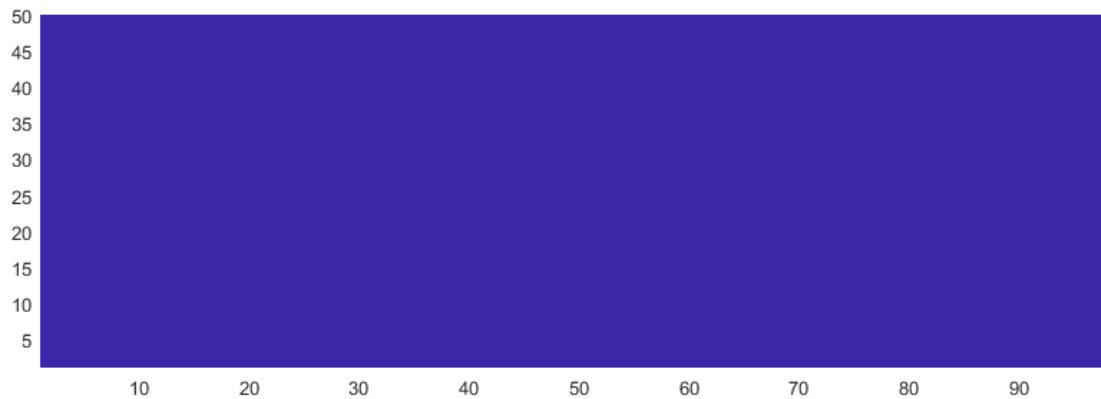
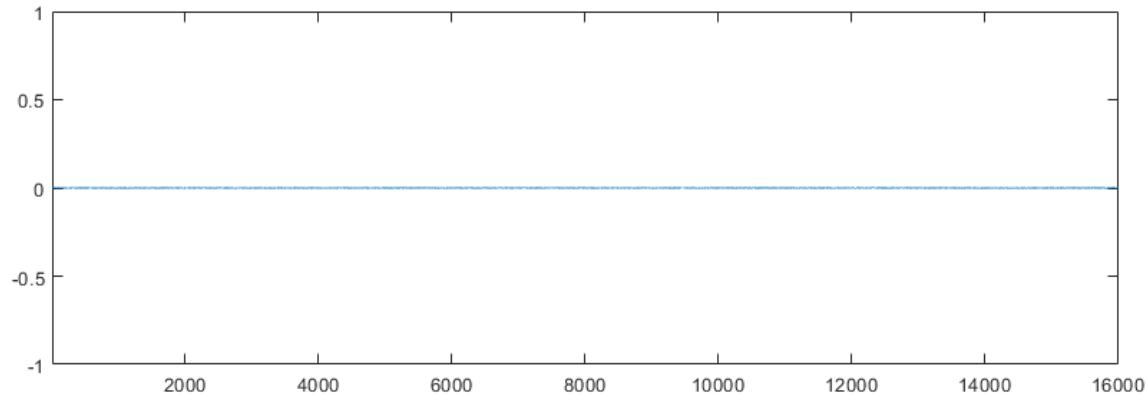
    subplot(2,1,2)
    pcolor(spec')
    caxis([-4 2.6445])
    shading flat

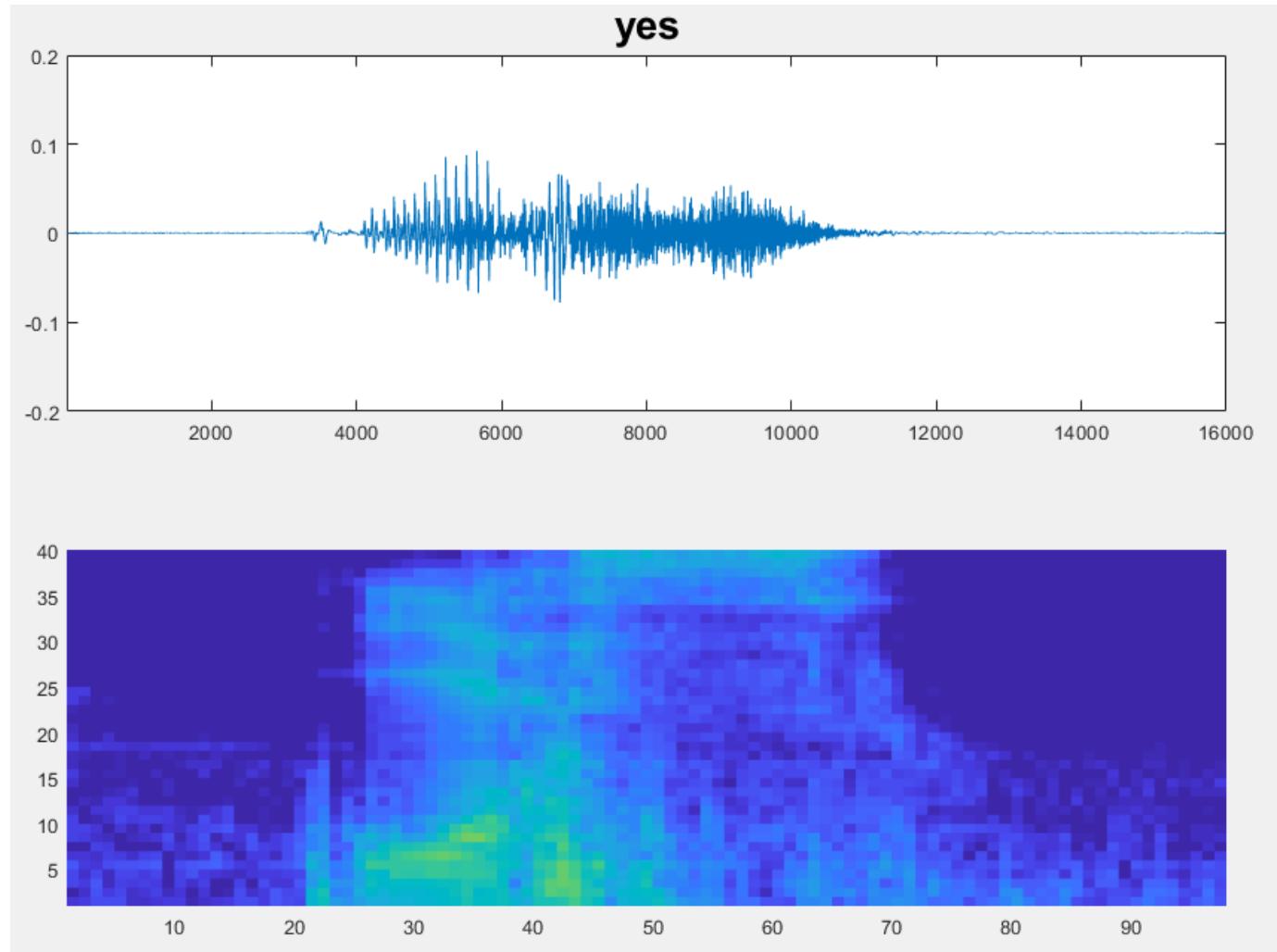
    % Now do the actual command detection by performing a very simple
    % thresholding operation. Declare a detection and display it in the
    % figure title if all of the following hold: 1) The most common label
    % is not background. 2) At least countThreshold of the latest frame
    % labels agree. 3) The maximum probability of the predicted label is at
    % least probThreshold. Otherwise, do not declare a detection.
    [YMode,count] = mode(YBuffer);

    maxProb = max(probBuffer(labels == YMode,:));
```

```
subplot(2,1,1)
if YMode == "background" || count < countThreshold || maxProb < probThreshold
    title(" ")
else
    title(string(YMode), 'FontSize', 20)
end

drawnow
end
```





### 加载语音命令数据集

此示例使用 Google Speech Commands Dataset [1]。下载该数据集并解压缩下载的文件。将 PathToDatabase 设置为数据的位置。

```
url = 'https://ssd.mathworks.com/supportfiles/audio/google_speech.zip';
downloadFolder = tempdir;
dataFolder = fullfile(downloadFolder,'google_speech');

if ~exist(dataFolder,'dir')
    disp('Downloading data set (1.4 GB) ...')
    unzip(url,downloadFolder)
end
```

### 创建训练数据存储

创建一个指向该训练数据集的 `audioDatastore` (Audio Toolbox)。

```
ads = audioDatastore(fullfile(dataFolder, 'train'), ...
    'IncludeSubfolders',true, ...
    'FileExtensions','.wav', ...
    'LabelSource','foldernames')
```

```

ads =
audioDatastore with properties:

    Files: {
        '...\\AppData\\Local\\Temp\\google_speech\\train\\bed\\00176480_nohash_0.wav';
        '...\\AppData\\Local\\Temp\\google_speech\\train\\bed\\004ae714_nohash_0.wav';
        '...\\AppData\\Local\\Temp\\google_speech\\train\\bed\\004ae714_nohash_1.wav'
        ... and 51085 more
    }
    Folders: {
        'C:\\Users\\jibrahim\\AppData\\Local\\Temp\\google_speech\\train'
    }
    Labels: [bed; bed; bed ... and 51085 more categorical]
    AlternateFileSystemRoots: {}
    OutputDataType: 'double'
    SupportedOutputFormats: ["wav" "flac" "ogg" "mp4" "m4a"]
    DefaultOutputFormat: "wav"

```

### 选择要识别的单词

指定您希望模型识别为命令的单词。将所有非命令单词标注为 **unknown**。将非命令单词标注为 **unknown** 会创建一个单词组，用来逼近除命令之外的所有单词的分布。网络使用该组来学习命令与所有其他单词之间的差异。

为了减少已知单词和未知单词之间的类不平衡并加快处理速度，只在训练集中包括未知单词的一小部分。

使用 **subset** (Audio Toolbox) 创建一个仅包含命令和未知单词子集的数据存储。计算属于每个类别的示例的数量。

```

commands = categorical(["yes","no","up","down","left","right","on","off","stop","go"]);
isCommand = ismember(ads.Labels,commands);
isUnknown = ~isCommand;

includeFraction = 0.2;
mask = rand(numel(ads.Labels),1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

adsTrain = subset(ads,isCommand | isUnknown);
countEachLabel(adsTrain)

```

ans =

11×2 table

Label	Count
down	1842
go	1861
left	1839
no	1853
off	1839

Label	Count
down	1842
go	1861
left	1839
no	1853
off	1839

```

on      1864
right   1852
stop    1885
unknown 6483
up      1843
yes     1860

```

### 创建验证数据存储

创建一个指向该验证数据集的 **audioDatastore** (Audio Toolbox)。按照用于创建训练数据存储的相同步骤进行操作。

```

ads = audioDatastore(fullfile(dataFolder, 'validation'), ...
    'IncludeSubfolders', true, ...
    'FileExtensions', '.wav', ...
    'LabelSource', 'foldernames')

isCommand = ismember(ads.Labels, commands);
isUnknown = ~isCommand;

includeFraction = 0.2;
mask = rand(numel(ads.Labels), 1) < includeFraction;
isUnknown = isUnknown & mask;
ads.Labels(isUnknown) = categorical("unknown");

adsValidation = subset(ads, isCommand | isUnknown);
countEachLabel(adsValidation)

ads =
audioDatastore with properties:

    Files: {
        '...\\AppData\\Local\\Temp\\google_speech\\validation\\bed\\026290a7_nohash_0.wav';
        '...\\AppData\\Local\\Temp\\google_speech\\validation\\bed\\060cd039_nohash_0.wav';
        '...\\AppData\\Local\\Temp\\google_speech\\validation\\bed\\060cd039_nohash_1.wav'
        ... and 6795 more
    }
    Folders: {
        'C:\\Users\\jibrahim\\AppData\\Local\\Temp\\google_speech\\validation'
    }
    Labels: [bed; bed; bed ... and 6795 more categorical]
    AlternateFileSystemRoots: {}
    OutputDataType: 'double'
    SupportedOutputFormats: ["wav" "flac" "ogg" "mp4" "m4a"]
    DefaultOutputFormat: "wav"

ans =
11×2 table

    Label    Count
    ____    ____

    down     264

```

```

go      260
left    247
no      270
off     256
on      257
right   256
stop    246
unknown 850
up      260
yes    261

```

要使用整个数据集来训练网络并达到尽可能最高的准确度，请将 `reduceDataset` 设置为 `false`。要快速运行此示例，请将 `reduceDataset` 设置为 `true`。

```

reduceDataset = false;
if reduceDataset
    numUniqueLabels = numel(unique(adsTrain.Labels));
    % Reduce the dataset by a factor of 20
    adsTrain = splitEachLabel(adsTrain,round(numel(adsTrain.Files) / numUniqueLabels / 20));
    adsValidation = splitEachLabel(adsValidation,round(numel(adsValidation.Files) / numUniqueLabels / 20));
end

```

## 计算听觉频谱图

为了准备能够高效训练卷积神经网络的数据，请将语音波形转换为基于听觉的频谱图。

定义特征提取的参数。`segmentDuration` 是每个语音段的持续时间（以秒为单位）。`frameDuration` 是用于计算频谱的每个帧的持续时间。`hopDuration` 是每个频谱之间的时间步。`numBands` 是听觉频谱图中的滤波器的数量。

创建一个 `audioFeatureExtractor` (Audio Toolbox) 对象来执行特征提取。

```

fs = 16e3; % Known sample rate of the data set.

segmentDuration = 1;
frameDuration = 0.025;
hopDuration = 0.010;

segmentSamples = round(segmentDuration*fs);
frameSamples = round(frameDuration*fs);
hopSamples = round(hopDuration*fs);
overlapSamples = frameSamples - hopSamples;

FFTLength = 512;
numBands = 50;

afe = audioFeatureExtractor( ...
    'SampleRate',fs, ...
    'FFTLength',FFTLength, ...
    'Window',hann(frameSamples,'periodic'), ...
    'OverlapLength',overlapSamples, ...
    'barkSpectrum',true);
setExtractorParams(afe,'barkSpectrum','NumBands',numBands,'WindowNormalization',false);

```

从数据集中读取一个文件。训练卷积神经网络要求输入大小一致。数据集中一些文件的长度不到 1 秒。在音频信号的前后应用零填充，使其长度为 `segmentSamples`。

```

x = read(adsTrain);

numSamples = size(x,1);

numToPadFront = floor( (segmentSamples - numSamples)/2 );
numToPadBack = ceil( (segmentSamples - numSamples)/2 );

xPadded = [zeros(numToPadFront,1,'like',x);x;zeros(numToPadBack,1,'like',x)];

```

要提取音频特征，请调用 `extract`。输出是列向为时间的 Bark 谱。

```

features = extract(afe,xPadded);
[numHops,numFeatures] = size(features)

```

```
numHops =
```

```
98
```

```
numFeatures =
```

```
50
```

在此示例中，您通过应用对数对听觉频谱图进行后处理。对小数字取对数可能会导致舍入误差。

为了加快处理速度，您可以使用 `parfor` 在多个进程之间分配特征提取。

首先，确定数据集的分区数量。如果您没有 Parallel Computing Toolbox™，请使用单一分区。

```

if ~isempty(ver('parallel')) && ~reduceDataset
    pool = gcp;
    numPar = numpartitions(adsTrain,pool);
else
    numPar = 1;
end

```

对于每个分区，从数据存储中读取，对信号进行零填充，然后提取特征。

```

parfor ii = 1:numPar
    subds = partition(adsTrain,numPar,ii);
    XTrain = zeros(numHops,numBands,1,numel(subds.Files));
    for idx = 1:numel(subds.Files)
        x = read(subds);
        xPadded = [zeros(floor((segmentSamples-size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];
        XTrain(:,:,idx) = extract(afe,xPadded);
    end
    XTrainC{ii} = XTrain;
end

```

将输出转换为 4 维数组，第 4 维为听觉频谱图。

```

XTrain = cat(4,XTrainC{:});
[numHops,numBands,numChannels,numSpec] = size(XTrain)

```

```
numHops =
```

98

```
numBands =
```

```
50
```

```
numChannels =
```

```
1
```

```
numSpec =
```

```
25021
```

按窗口幂缩放特征，然后取对数。要获得具有更平滑分布的数据，请使用小偏移取频谱图的对数。

```
epsil = 1e-6;
XTrain = log10(XTrain + epsil);
```

对验证集执行上述特征提取步骤。

```
if ~isempty(ver('parallel'))
    pool = gcp;
    numPar = numpartitions(adsValidation,pool);
else
    numPar = 1;
end
parfor ii = 1:numPar
    subds = partition(adsValidation,numPar,ii);
    XValidation = zeros(numHops,numBands,1,numel(subds.Files));
    for idx = 1:numel(subds.Files)
        x = read(subds);
        xPadded = [zeros(floor((segmentSamples-size(x,1))/2),1);x;zeros(ceil((segmentSamples-size(x,1))/2),1)];
        XValidation(:,:,idx) = extract(afe,xPadded);
    end
    XValidationC{ii} = XValidation;
end
XValidation = cat(4,XValidationC{:});
XValidation = log10(XValidation + epsil);
```

对训练标签和验证标签进行隔离。删除空类别。

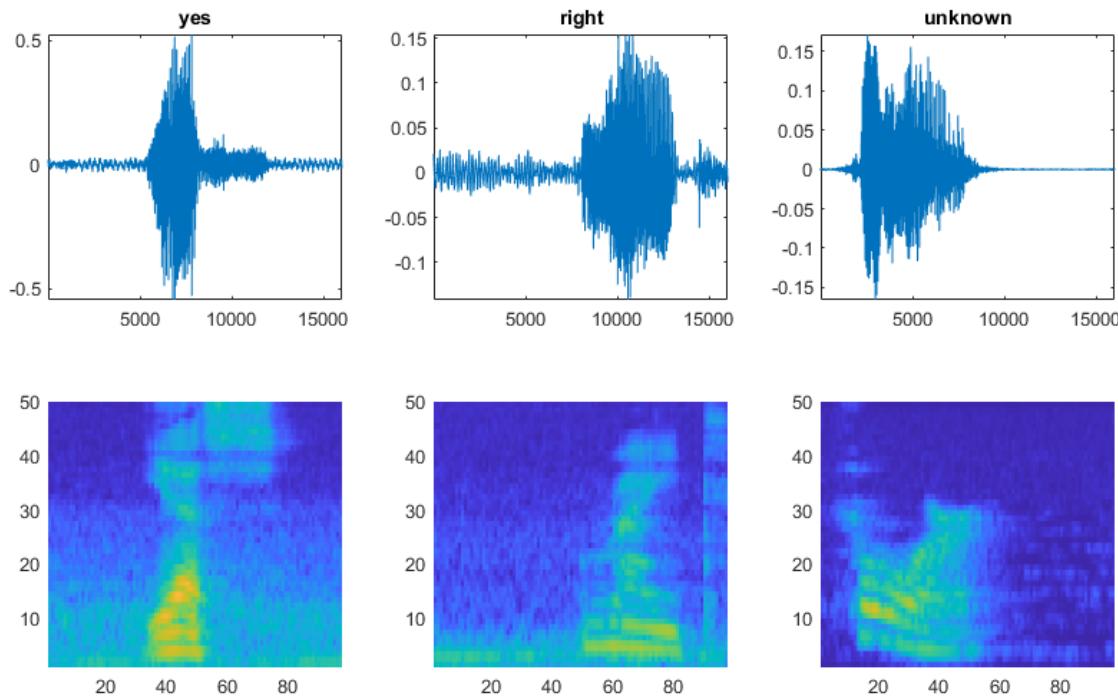
```
YTrain = removecats(adsTrain.Labels);
YValidation = removecats(adsValidation.Labels);
```

## 可视化数据

绘制几个训练样本的波形和听觉频谱图。播放对应的音频片段。

```
specMin = min(XTrain,[],'all');
specMax = max(XTrain,[],'all');
idx = randperm(numel(adsTrain.Files),3);
figure('Units','normalized','Position',[0.2 0.2 0.6 0.6]);
for i = 1:3
```

```
[x,fs] = audioread(adsTrain.Files{idx(i)});  
subplot(2,3,i)  
plot(x)  
axis tight  
title(string(adsTrain.Labels(idx(i))))  
  
subplot(2,3,i+3)  
spect = (XTrain(:,:,1, idx(i))');  
pcolor(spect)  
caxis([specMin specMax])  
shading flat  
  
sound(x,fs)  
pause(2)  
end
```



### 添加背景噪声数据

网络必须不仅能够识别不同发音的单词，还要能够检测输入是否包含静音或背景噪声。

使用 `_background_` 文件夹中的音频文件创建一秒背景噪声的片段采样。根据每个背景噪声文件创建相同数量的背景片段。您还可以创建自己的背景噪声录音，并将它们添加到 `_background_` 文件夹。在计算频谱图之前，该函数将使用从 `volumeRange` 给出的范围内的对数均匀分布中采样的因子重新调整每个音频片段。

```
adsBkg = audioDatastore(fullfile(dataFolder, 'background'))  
numBkgClips = 4000;  
if reduceDataset  
    numBkgClips = numBkgClips/20;  
end
```

```

volumeRange = log10([1e-4,1]);

numBkgFiles = numel(adsBkg.Files);
numClipsPerFile = histcounts(1:numBkgClips,linspace(1,numBkgClips,numBkgFiles+1));
Xbkg = zeros(size(XTrain,1),size(XTrain,2),1,numBkgClips,'single');
bkgAll = readall(adsBkg);
ind = 1;

for count = 1:numBkgFiles
    bkg = bkgAll{count};
    idxStart = randi(numel(bkg)-fs,numClipsPerFile(count),1);
    idxEnd = idxStart+fs-1;
    gain = 10.^((volumeRange(2)-volumeRange(1))*rand(numClipsPerFile(count),1) + volumeRange(1));
    for j = 1:numClipsPerFile(count)

        x = bkg(idxStart(j):idxEnd(j))*gain(j);

        x = max(min(x,1),-1);

        Xbkg(:,:,:,:ind) = extract(afe,x);

        if mod(ind,1000)==0
            disp("Processed " + string(ind) + " background clips out of " + string(numBkgClips))
        end
        ind = ind + 1;
    end
end
Xbkg = log10(Xbkg + epsil);

```

adsBkg =

audioDatastore with properties:

```

Files: {
    '\AppData\Local\Temp\google_speech\background\doing_the_dishes.wav';
    '\AppData\Local\Temp\google_speech\background\dude_miaowing.wav';
    '\AppData\Local\Temp\google_speech\background\exercise_bike.wav'
    ... and 3 more
}
Folders: {
    'C:\Users\jibrahim\AppData\Local\Temp\google_speech\background'
}
AlternateFileSystemRoots: {}
OutputDataType: 'double'
Labels: {}
SupportedOutputFormats: ["wav" "flac" "ogg" "mp4" "m4a"]
DefaultOutputFormat: "wav"

```

Processed 1000 background clips out of 4000  
 Processed 2000 background clips out of 4000  
 Processed 3000 background clips out of 4000  
 Processed 4000 background clips out of 4000

对背景噪声的频谱图进行拆分，以用于训练集、验证集和测试集。由于 `_background_noise_` 文件夹仅包含大约五分半钟的背景噪声，因此不同数据集中的背景采样高度相关。要增加背景噪声的变化，您可以创建自己的背景文件并添加到该文件夹中。要增强网络的抗噪稳健性，您还可以尝试将背景噪声混合到语音文件中。

```

numTrainBkg = floor(0.85*numBkgClips);
numValidationBkg = floor(0.15*numBkgClips);

XTrain(:,:,end+1:end+numTrainBkg) = Xbkg(:,:,1:numTrainBkg);
YTrain(end+1:end+numTrainBkg) = "background";

XValidation(:,:,end+1:end+numValidationBkg) = Xbkg(:,:,numTrainBkg+1:end);
YValidation(end+1:end+numValidationBkg) = "background";

```

绘制训练集和验证集中不同类标签的分布。

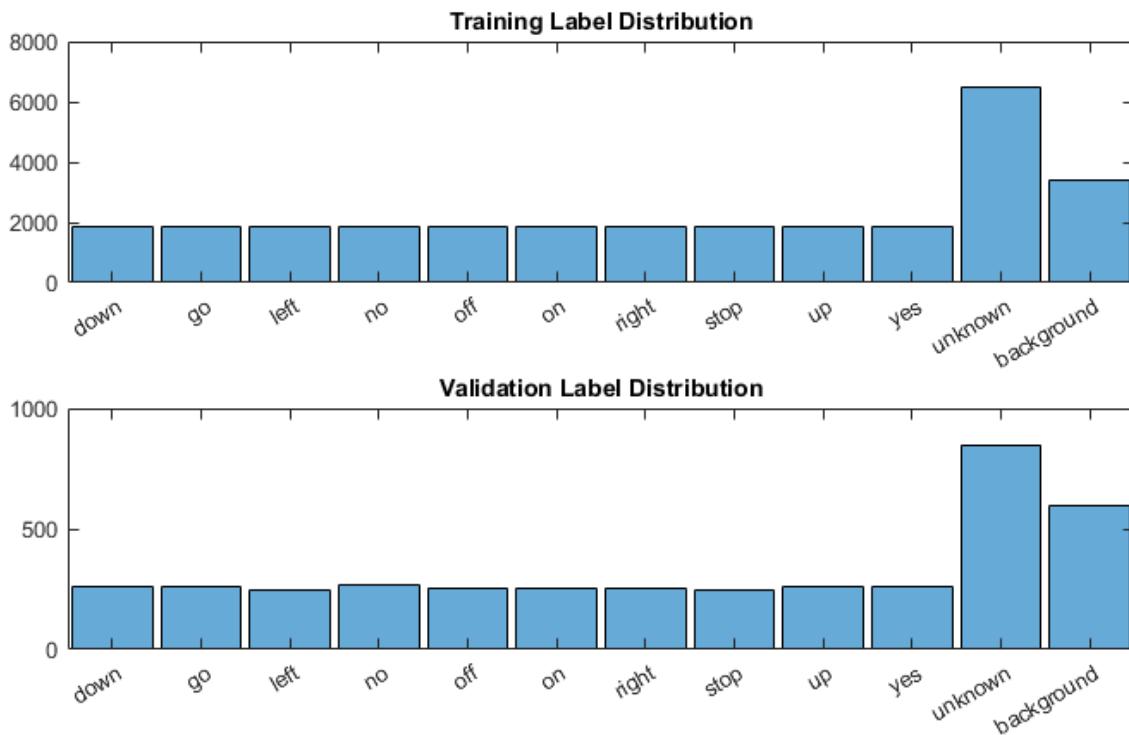
```

figure('Units','normalized','Position',[0.2 0.2 0.5 0.5])

subplot(2,1,1)
histogram(YTrain)
title("Training Label Distribution")

subplot(2,1,2)
histogram(YValidation)
title("Validation Label Distribution")

```



### 定义神经网络架构

创建一个层数组形式的简单网络架构。使用卷积层和批量归一化层，并使用最大池化层“在空间上”（即，在时间和频率上）对特征图进行下采样。添加最终的最大池化层，它随时间对输入特征图进行全局池化。这会在输入频谱图中强制实施（近似的）时间平移不变性，从而使网络在对语音进行分类时不依赖于语音的准确时间位置，得到相同的分类结果。全局池化还可以显著减少最终全连接层中的参数数量。为了降低网络记住训练数据特定特征的可能性，可为最后一个全连接层的输入添加一个小的丢弃率。

该网络很小，因为它只有五个卷积层和几个滤波器。`numF` 控制卷积层中的滤波器数量。要提高网络的准确度，请尝试通过添加一些相同的块（由卷积层、批量归一化层和 ReLU 层组成）来增加网络深度。还可以尝试通过增大 `numF` 来增加卷积滤波器的数量。

使用加权交叉熵分类损失。`weightedClassificationLayer(classWeights)` 可创建一个自定义分类层，用于计算按 `classWeights` 加权的观测值的交叉熵损失。按照 `categories(YTrain)` 中类的显示顺序指定相同顺序的类权重。为了使每个类在损失中的总权重相等，使用的类权重应与每个类的训练样本数成反比。使用 Adam 优化器训练网络时，训练算法与类权重的整体归一化无关。

```
classWeights = 1./countcats(YTrain);
classWeights = classWeights'/mean(classWeights);
numClasses = numel(categories(YTrain));

timePoolSize = ceil(numHops/8);

dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer([numHops numBands])

    convolution2dLayer(3,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer([timePoolSize,1])

    dropoutLayer(dropoutProb)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    weightedClassificationLayer(classWeights)];
```

## 训练网络

指定训练选项。使用小批量大小为 128 的 Adam 优化器。进行 25 轮训练，并在 20 轮后将学习率降低十分之一。

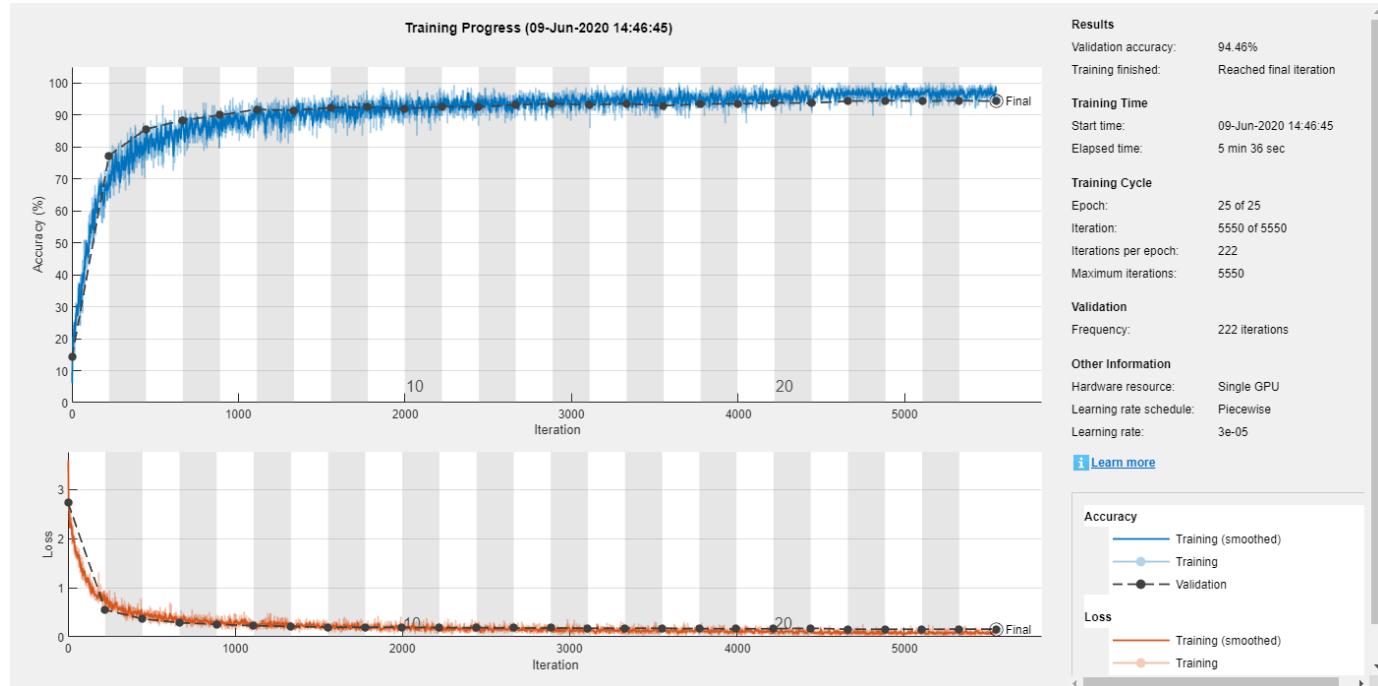
```

miniBatchSize = 128;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('adam', ...
    'InitialLearnRate',3e-4, ...
    'MaxEpochs',25, ...
    'MiniBatchSize',miniBatchSize, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose',false, ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',validationFrequency, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',20);

```

训练网络。如果您没有 GPU，则训练网络可能需要较长的时间。

```
trainedNet = trainNetwork(XTrain,YTrain,layers,options);
```



### 评估经过训练的网络

基于训练集（无数据增强）和验证集计算网络的最终准确度。网络对于此数据集非常准确。但是，训练数据、验证数据和测试数据全都具有相似的分布，不一定能反映真实环境。尤其是对仅包含少量单词读音的 **unknown** 类别，更是如此。

```

if reduceDataset
    load('commandNet.mat','trainedNet');
end
YValPred = classify(trainedNet,XValidation);
validationError = mean(YValPred ~= YValidation);
YTrainPred = classify(trainedNet,XTrain);
trainError = mean(YTrainPred ~= YTrain);

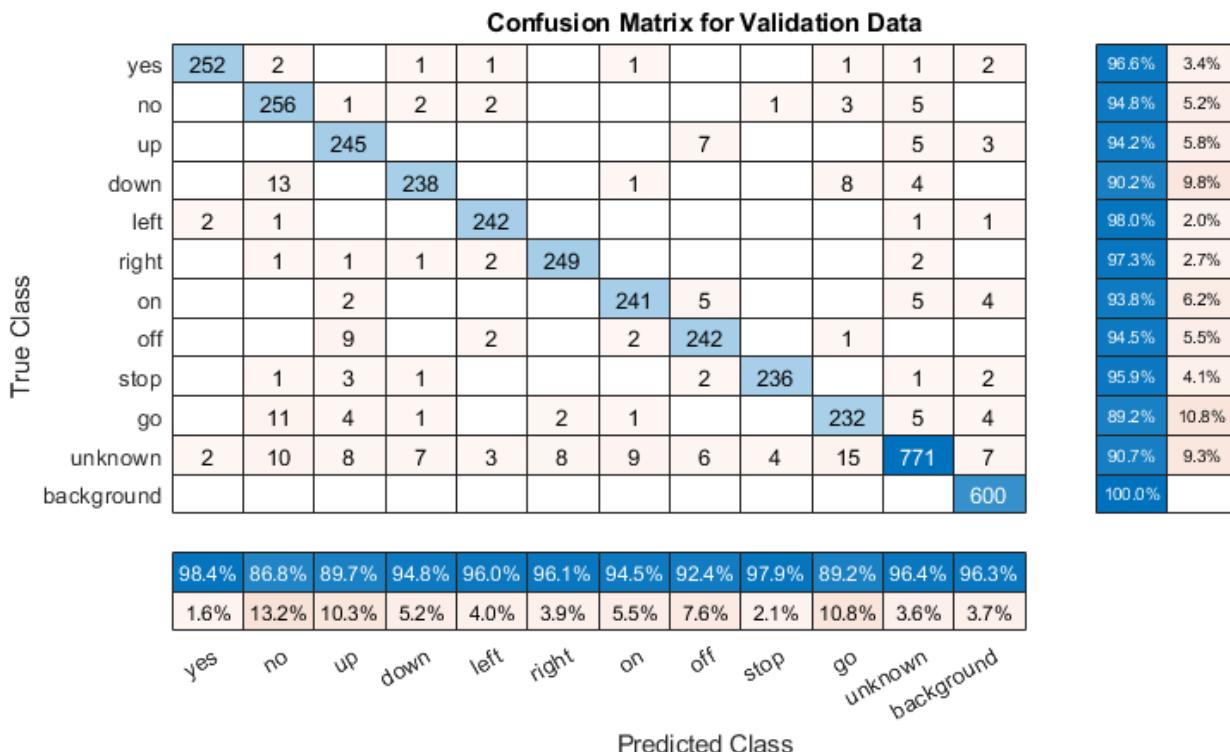
```

```
disp("Training error: " + trainError*100 + "%")
disp("Validation error: " + validationError*100 + "%")
```

Training error: 1.907%  
Validation error: 5.5376%

绘制混淆矩阵。使用列汇总和行汇总显示每个类的准确率和召回率。对混淆矩阵的类进行排序。最大的混淆发生在未知单词与命令之间，以及 up 和 off、down 和 no，以及 go 和 no 这三对命令之间。

```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
cm = confusionchart(YValidation,YValPred);
cm.Title = 'Confusion Matrix for Validation Data';
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
sortClasses(cm, [commands,"unknown","background"])
```



在处理硬件资源受限的应用（如移动应用）时，请考虑可用内存和计算资源的限制。当使用 CPU 时，以 KB 为单位计算网络总大小，并测试网络的预测速度。预测时间是指对单个输入图像进行分类的时间。如果向网络中输入多个图像，可以同时对它们进行分类，从而缩短每个图像的预测时间。然而，在对流音频进行分类时，单个图像预测时间是最相关的。

```
info = whos('trainedNet');
disp("Network size: " + info.bytes/1024 + " kB")

for i = 1:100
    x = randn([numHops,numBands]);
    tic
    [YPredicted,probs] = classify(trainedNet,x,"ExecutionEnvironment",'cpu');
    time(i) = toc;
```

```
end  
disp("Single-image prediction time on CPU: " + mean(time(11:end))*1000 + " ms")
```

Network size: 286.7402 kB  
Single-image prediction time on CPU: 2.5119 ms

### 参考资料

[1] Warden P."Speech Commands:A public dataset for single-word speech recognition", 2017.可从 [https://storage.googleapis.com/download.tensorflow.org/data/speech\\_commands\\_v0.01.tar.gz](https://storage.googleapis.com/download.tensorflow.org/data/speech_commands_v0.01.tar.gz) 获得。Copyright Google 2017.Speech Commands Dataset 是根据 Creative Commons Attribution 4.0 许可证授权的，可通过 <https://creativecommons.org/licenses/by/4.0/legalcode> 获得。

### 参考

[1] Warden P. "Speech Commands: A public dataset for single-word speech recognition", 2017.  
Available from [http://download.tensorflow.org/data/speech\\_commands\\_v0.01.tar.gz](http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz).  
Copyright Google 2017. The Speech Commands Dataset is licensed under the Creative Commons Attribution 4.0 license, available here: <https://creativecommons.org/licenses/by/4.0/legalcode>.

### 另请参阅

[trainNetwork](#) | [classify](#) | [analyzeNetwork](#)

### 详细信息

- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 使用深度学习进行“序列到序列”分类

此示例说明如何使用长短期记忆 (LSTM) 网络对序列数据的每个时间步进行分类。

要训练深度神经网络以对序列数据的每个时间步进行分类，可以使用“序列到序列” LSTM 网络。通过“序列到序列” LSTM 网络，您可以对序列数据的每个时间步进行不同预测。

此示例使用从佩戴在身体上的智能手机获得的传感器数据。该示例训练一个 LSTM 网络，旨在根据表示三个不同方向上的加速度计读数的时间序列数据来识别佩戴者的活动。训练数据包含七个人的时序数据。每个序列有三个特征，且长度不同。该数据集包含六个训练观测值和一个测试观测值。

## 加载序列数据

加载人体活动识别数据。该数据包含从佩戴在身体上的智能手机获得的七个时序的传感器数据。每个序列有三个特征，且长度不同。这三个特征对应于三个不同方向上的加速度计读数。

```
load HumanActivityTrain
XTrain
```

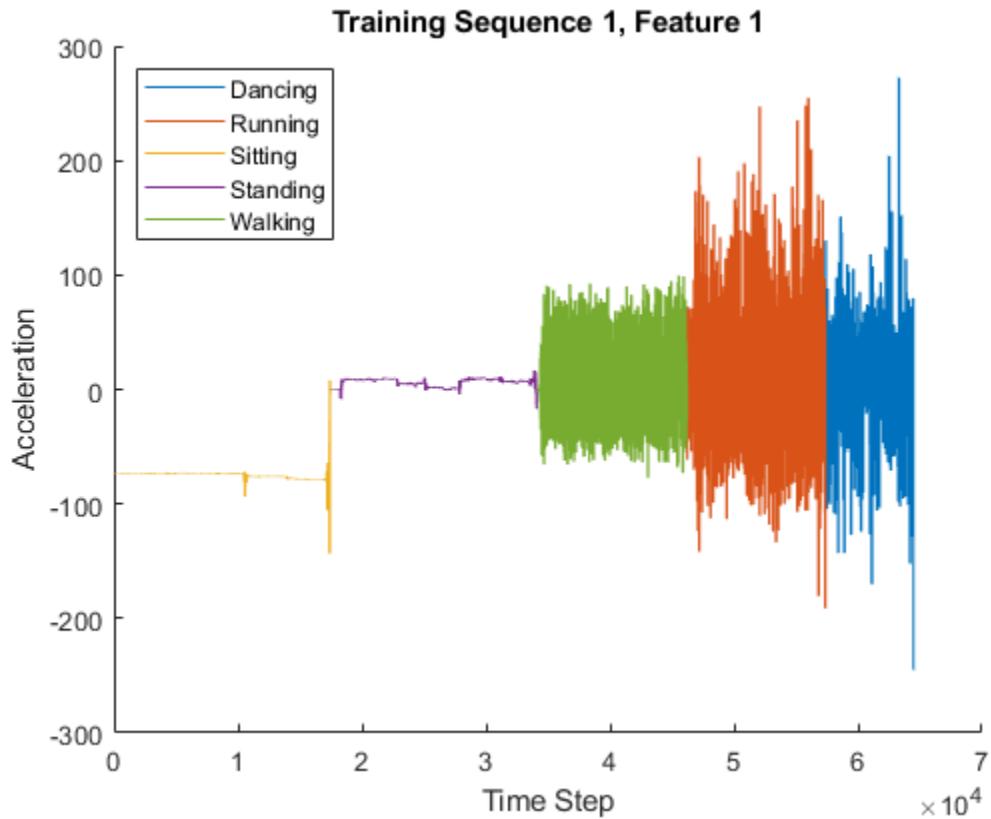
```
XTrain=6×1 cell array
{3×64480 double}
{3×53696 double}
{3×56416 double}
{3×50688 double}
{3×51888 double}
{3×54256 double}
```

在绘图中可视化一个训练序列。绘制第一个训练序列的第一个特征，并按照对应的活动为绘图着色。

```
X = XTrain{1}(1,:);
classes = categories(YTrain{1});

figure
for j = 1:numel(classes)
    label = classes(j);
    idx = find(YTrain{1} == label);
    hold on
    plot(idx,X(idx))
end
hold off

xlabel("Time Step")
ylabel("Acceleration")
title("Training Sequence 1, Feature 1")
legend(classes,'Location','northwest')
```



### 定义 LSTM 网络架构

定义 LSTM 网络架构。将输入指定为大小为 3 (输入数据的特征数量) 的序列。指定包含 200 个隐含单元的 LSTM 层，并输出完整序列。最后，在网络中包含一个大小为 5 的全连接层，后跟 softmax 层和分类层，以此来指定五个类。

```
numFeatures = 3;
numHiddenUnits = 200;
numClasses = 5;

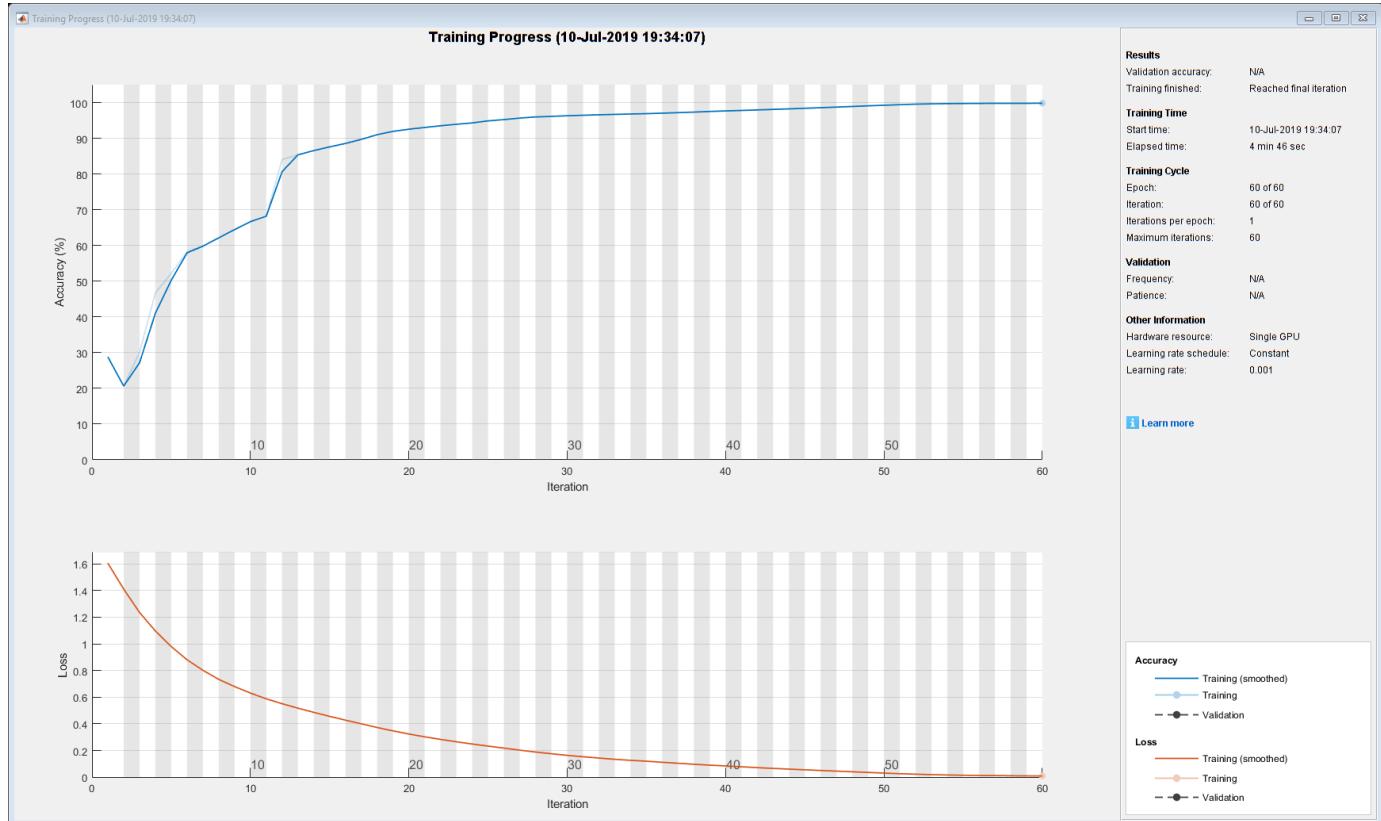
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

指定训练选项。将求解器设置为 'adam'。进行 60 轮训练。要防止梯度爆炸，请将梯度阈值设置为 2。

```
options = trainingOptions('adam',...
    'MaxEpochs',60, ...
    'GradientThreshold',2, ...
    'Verbose',0, ...
    'Plots','training-progress');
```

使用 `trainNetwork` 以指定的训练选项训练 LSTM 网络。每个小批量都包含整个训练集，因此每训练一轮便更新一次绘图。序列非常长，因此处理每个小批量并更新绘图可能需要一些时间。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```

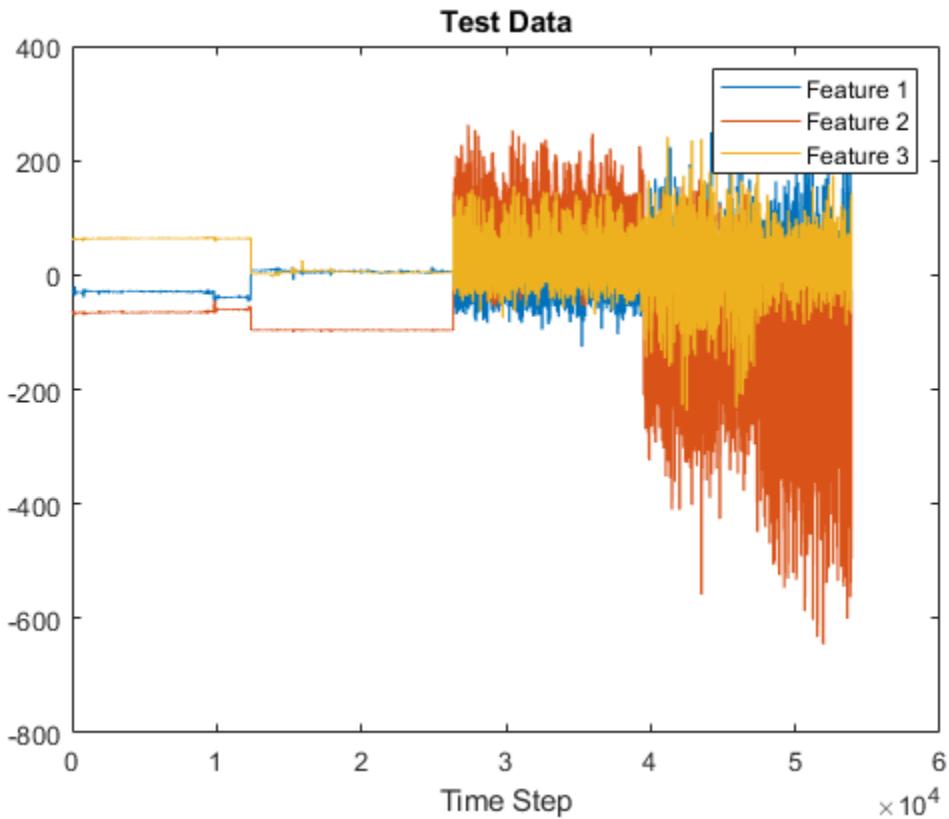


## 测试 LSTM 网络

加载测试数据并对每个时间步的活动进行分类。

加载人体活动测试数据。XTest 包含一个维度为 3 的序列。YTest 包含对于每个时间步的活动的分类标签序列。

```
load HumanActivityTest
figure
plot(XTest{1})
xlabel("Time Step")
legend("Feature " + (1:numFeatures))
title("Test Data")
```



使用 `classify` 对测试数据进行分类。

```
YPred = classify(net,XTest{1});
```

您也可以使用 `classifyAndUpdateState` 一次对一个时间步进行预测。这在时间步的值以流的方式到达时非常有用。通常，对完整序列进行预测比一次对一个时间步进行预测更快。有关如何通过在相邻的单个时间步预测之间更新网络来预测将来时间步的示例，请参阅“使用深度学习进行时序预测”（第 4-9 页）。

计算预测的准确度。

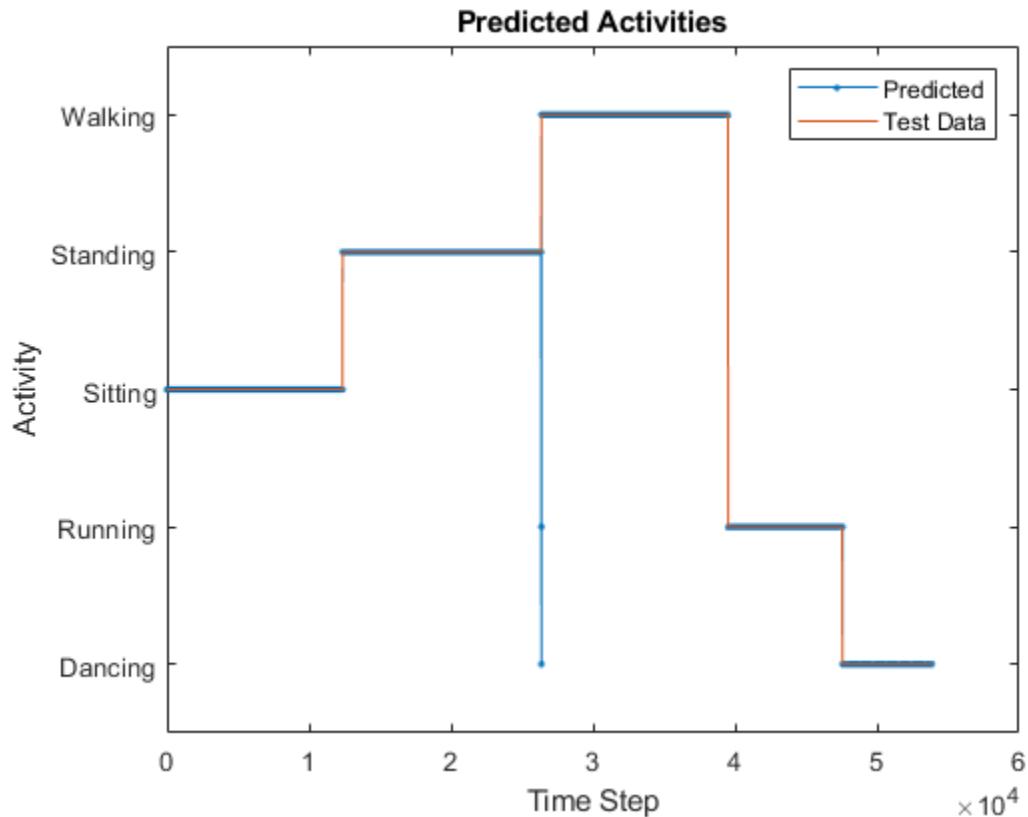
```
acc = sum(YPred == YTest{1})./numel(YTest{1})
```

```
acc = 0.9998
```

通过绘图将预测值与测试数据进行比较。

```
figure
plot(YPred,'.-')
hold on
plot(YTest{1})
hold off

xlabel("Time Step")
ylabel("Activity")
title("Predicted Activities")
legend(["Predicted" "Test Data"])
```



## 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [lstmLayer](#) | [sequenceInputLayer](#)

## 相关示例

- “使用深度学习进行序列分类” (第 4-2 页)
- “使用深度学习进行时序预测” (第 4-9 页)
- “使用深度学习进行“序列到序列”回归” (第 4-38 页)
- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 使用深度学习进行“序列到序列”回归

此示例说明如何使用深度学习预测发动机的剩余使用寿命 (RUL)。

要训练深度神经网络以根据时序数据或序列数据预测数值，可以使用长短期记忆 (LSTM) 网络。

此示例使用 [1] 中所述的涡轮风扇发动机退化仿真数据集。该示例训练一个 LSTM 网络，旨在根据表示发动机中各种传感器的时序数据来预测发动机的剩余使用寿命（预测性维护，以周期为单位度量）。训练数据包含 100 台发动机的仿真时序数据。每个序列的长度各不相同，对应于完整的运行至故障 (RTF) 实例。测试数据包含 100 个不完整序列，每个序列的末尾为相应的剩余使用寿命值。

该数据集包含 100 个训练观测值和 100 个测试观测值。

### 下载数据

从 <https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/> [2] 下载并解压缩涡轮风扇发动机退化仿真数据集。

涡轮风扇发动机退化仿真数据集的每个时序表示一个发动机。每台发动机启动时的初始磨损程度和制造变差均未知。发动机在每个时序开始时运转正常，在到达序列中的某一时刻时出现故障。在训练集中，故障的规模不断增大，直到出现系统故障。

数据是 ZIP 压缩的文本文件，其中包含 26 列以空格分隔的数值。每一行是在一个运转周期中截取的数据快照，每一列代表一个不同的变量。这些列分别对应于以下数据：

- 第 1 列 - 单元编号
- 第 2 列 - 周期时间
- 第 3-5 列 - 操作设置
- 第 6-26 列 - 传感器测量值 1-21

创建一个目录来存储涡轮风扇发动机退化仿真数据集。

```
dataFolder = fullfile(tempdir,"turbofan");
if ~exist(dataFolder,'dir')
    mkdir(dataFolder);
end
```

从 <https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/> 下载并提取涡轮风扇发动机退化仿真数据集。

从文件 CMAPSSData.zip 中解压缩数据。

```
filename = "CMAPSSData.zip";
unzip(filename,dataFolder)
```

### 准备训练数据

使用此示例附带的函数 `processTurboFanDataTrain` 加载数据。函数 `processTurboFanDataTrain` 从 `filenamePredictors` 中提取数据并返回元胞数组 `XTrain` 和 `YTrain`，其中包含训练预测变量和响应序列。

```
filenamePredictors = fullfile(dataFolder,"train_FD001.txt");
[XTrain,YTrain] = processTurboFanDataTrain(filenamePredictors);
```

### 删除具有常量值的特征

在所有时间步都保持不变的特征可能对训练产生负面影响。找到最小值和最大值相同的数据行，然后删除这些行。

```
m = min([XTrain{:}],[],2);
M = max([XTrain{:}],[],2);
idxConstant = M == m;

for i = 1:numel(XTrain)
    XTrain{i}(idxConstant,:) = [];
end
```

查看序列中其余特征的数量。

```
numFeatures = size(XTrain{1},1)

numFeatures = 17
```

## 归一化训练预测变量

将训练预测变量归一化为具有零均值和单位方差。要计算所有观测值的均值和标准差，请水平串联序列数据。

```
mu = mean([XTrain{:}],2);
sig = std([XTrain{:}],0,2);

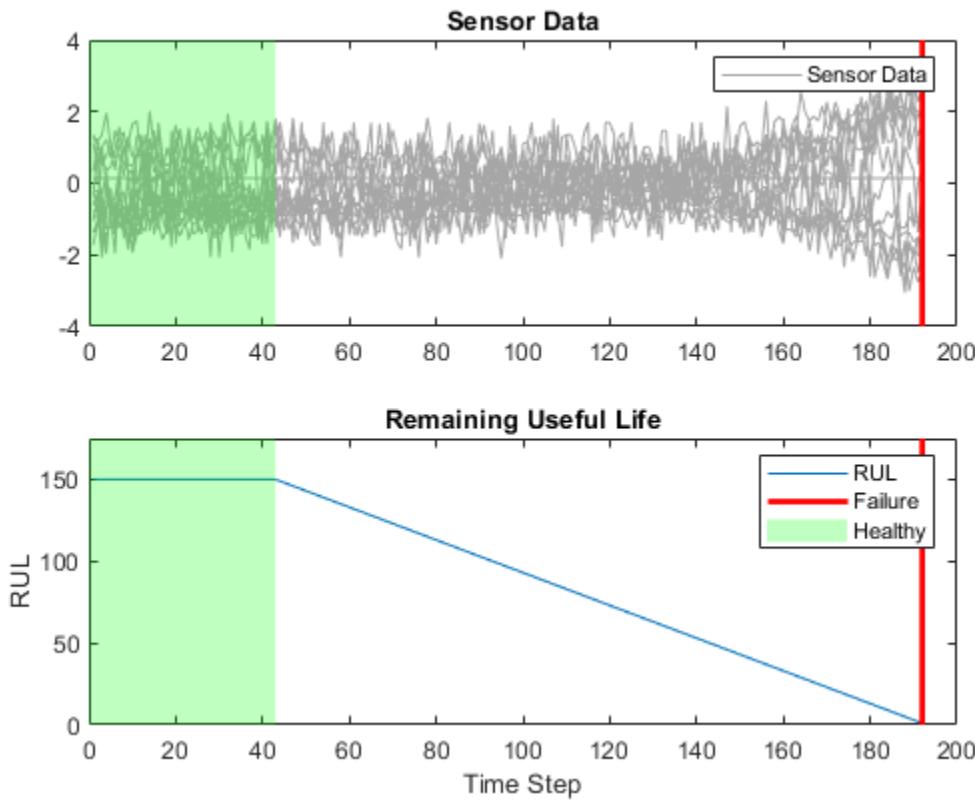
for i = 1:numel(XTrain)
    XTrain{i} = (XTrain{i} - mu) ./ sig;
end
```

## 裁剪响应

要更多地从发动机快要出现故障时的序列数据中进行学习，请以阈值 150 对响应进行裁剪。这会使网络将具有更高 RUL 值的实例视为等同。

```
thr = 150;
for i = 1:numel(YTrain)
    YTrain{i}(YTrain{i} > thr) = thr;
end
```

下图显示了第一个观测值及其对应的裁剪响应。



### 准备要填充的数据

要最大程度地减少添加到小批量的填充量，请按序列长度对训练数据进行排序。然后，选择可均匀划分训练数据的小批量大小，并减少小批量中的填充量。

按序列长度对训练数据进行排序。

```

for i=1:numel(XTrain)
    sequence = XTrain{i};
    sequenceLengths(i) = size(sequence,2);
end

[sequenceLengths,idx] = sort(sequenceLengths,'descend');
XTrain = XTrain(idx);
YTrain = YTrain(idx);

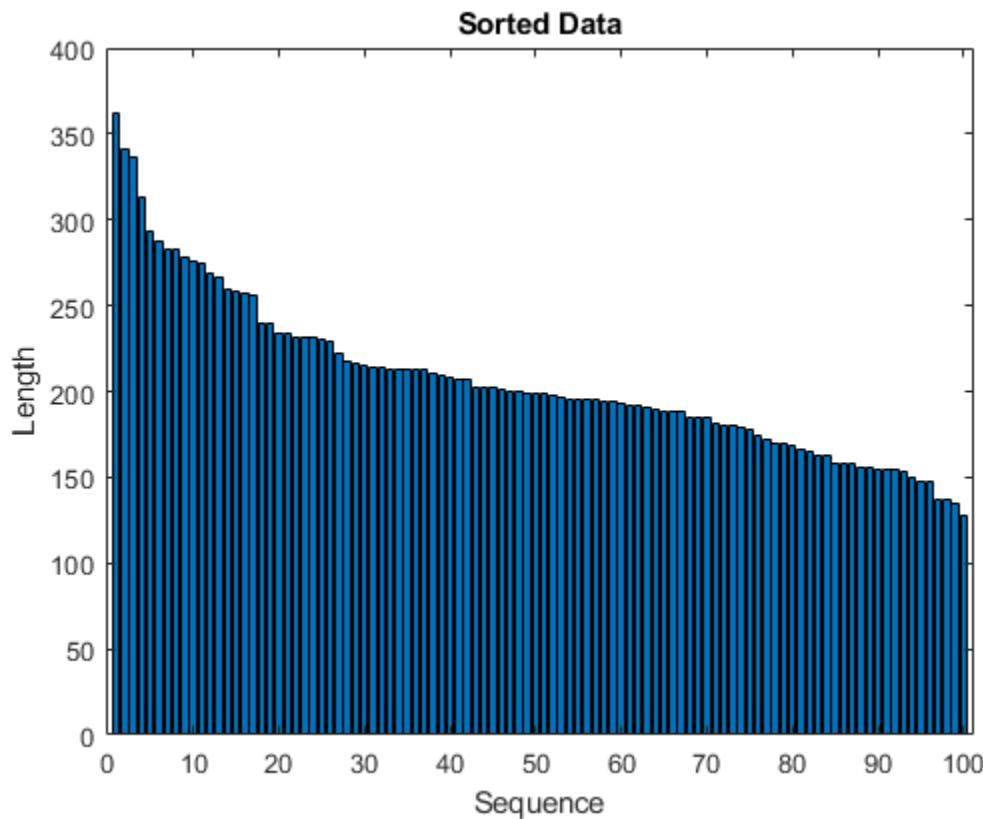
```

在条形图中查看排序的序列长度。

```

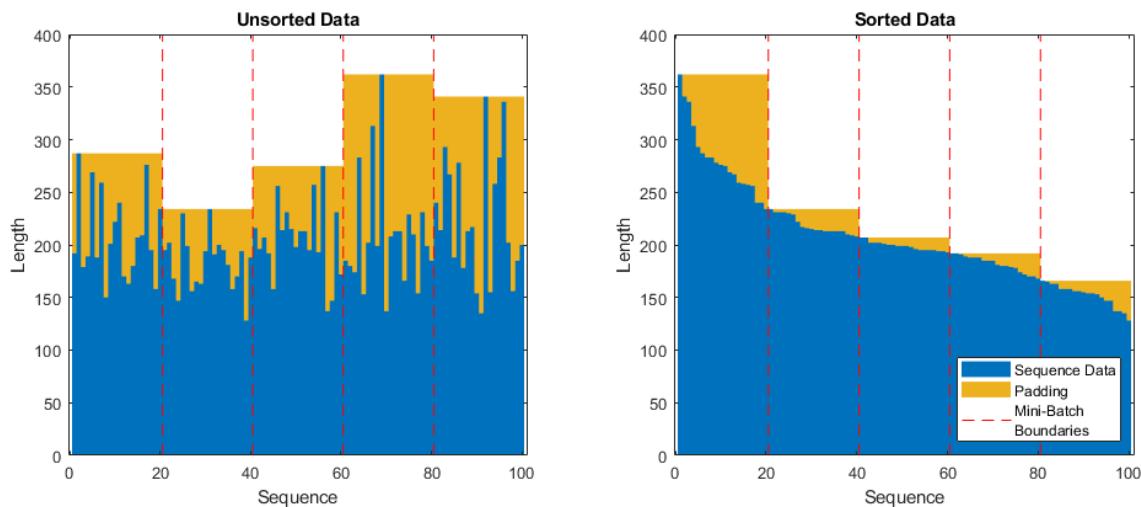
figure
bar(sequenceLengths)
xlabel("Sequence")
ylabel("Length")
title("Sorted Data")

```



选择可均匀划分训练数据的小批量大小，并减少小批量中的填充量。指定小批量大小为 20。下图显示了未排序序列和已排序序列的填充情况。

**miniBatchSize = 20;**



## 定义网络架构

定义网络架构。创建一个 LSTM 网络，该网络包含一个具有 200 个隐藏单元的 LSTM 层，然后是一个大小为 50 的全连接层和一个丢弃概率为 0.5 的丢弃层。

```
numResponses = size(YTrain{1},1);
numHiddenUnits = 200;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(50)
    dropoutLayer(0.5)
    fullyConnectedLayer(numResponses)
    regressionLayer];
```

指定训练选项。使用求解器 'adam' 以大小为 20 的小批量进行 60 轮训练。指定学习率为 0.01。要防止梯度爆炸，请将梯度阈值设置为 1。要使序列保持按长度排序，请将 'Shuffle' 设置为 'never'。

```
maxEpochs = 60;
miniBatchSize = 20;

options = trainingOptions('adam', ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'InitialLearnRate',0.01, ...
    'GradientThreshold',1, ...
    'Shuffle','never', ...
    'Plots','training-progress',...
    'Verbose',0);
```

## 训练网络

使用 `trainNetwork` 训练网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```

## 测试网络

使用此示例附带的函数 `processTurboFanDataTest` 准备测试数据。函数 `processTurboFanDataTest` 从 `filenamePredictors` 和 `filenameResponses` 中提取数据并返回元胞数组 `XTest` 和 `YTest`，其中分别包含测试预测变量和响应序列。

```
filenamePredictors = fullfile(dataFolder,"test_FD001.txt");
filenameResponses = fullfile(dataFolder,"RUL_FD001.txt");
[XTest,YTest] = processTurboFanDataTest(filenamePredictors,filenameResponses);
```

使用根据训练数据计算出的 `idxConstant` 删除具有常量值的特征。使用与训练数据相同的参数来归一化测试预测变量。使用与训练数据相同的阈值对测试响应进行裁剪。

```
for i = 1:numel(XTest)
    XTest{i}(idxConstant,:) = [];
    XTest{i} = (XTest{i} - mu) ./ sig;
    YTest{i}(YTest{i} > thr) = thr;
end
```

使用 `predict` 对测试数据进行预测。为防止函数向数据添加填充，请指定小批量大小为 1。

```
YPred = predict(net,XTest,'MiniBatchSize',1);
```

LSTM 网络对不完整序列进行预测，一次预测一个时间步。在每个时间步，网络使用此时间步的值进行预测，网络状态仅根据先前的时间步进行计算。网络在各次预测之间更新其状态。**predict** 函数返回这些预测值的序列。预测值的最后一个元素对应于不完整序列的预测 RUL。

您也可以使用 **predictAndUpdateState** 一次对一个时间步进行预测。这在时间步的值以流的方式到达时非常有用。通常，对完整序列进行预测比一次对一个时间步进行预测更快。有关如何通过在相邻的单个时间步预测之间更新网络来预测将来时间步的示例，请参阅“使用深度学习进行时序预测”（第 4-9 页）。

在绘图中可视化一些预测值。

```
idx = randperm(numel(YPred),4);
figure
for i = 1:numel(idx)
    subplot(2,2,i)

    plot(YTest{idx(i)},'--')
    hold on
    plot(YPred{idx(i)},'.-')
    hold off

    ylim([0 thr + 25])
    title("Test Observation " + idx(i))
    xlabel("Time Step")
    ylabel("RUL")
end
legend(["Test Data" "Predicted"],'Location','southeast')
```

对于给定的不完整序列，预测的当前 RUL 是预测序列的最后一个元素。计算预测值的均方根误差 (RMSE)，并在直方图中可视化预测误差。

```
for i = 1:numel(YTest)
    YTestLast(i) = YTest{i}(end);
    YPredLast(i) = YPred{i}(end);
end
figure
rmse = sqrt(mean((YPredLast - YTestLast).^2))
histogram(YPredLast - YTestLast)
title("RMSE = " + rmse)
ylabel("Frequency")
xlabel("Error")
```

## 参考资料

- 1 Saxena, Abhinav, Kai Goebel, Don Simon, and Neil Eklund."Damage propagation modeling for aircraft engine run-to-failure simulation."In Prognostics and Health Management, 2008.PHM 2008.International Conference on, pp. 1-9.IEEE, 2008.
- 2 Saxena, Abhinav, Kai Goebel."Turbofan Engine Degradation Simulation Data Set."NASA Ames Prognostics Data Repository <https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/>, NASA Ames Research Center, Moffett Field, CA

## 另请参阅

**trainNetwork** | **trainingOptions** | **lstmLayer** | **sequenceInputLayer** |  
**predictAndUpdateState**

### 另请参阅

#### 相关示例

- “使用深度学习进行序列分类” (第 4-2 页)
- “使用深度学习进行时序预测” (第 4-9 页)
- “使用深度学习进行“序列到序列”分类” (第 4-33 页)
- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 使用深度学习对视频进行分类

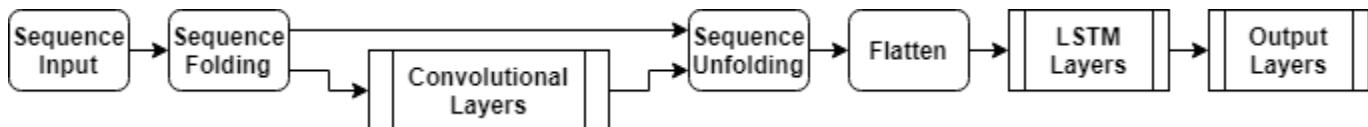
此示例说明如何通过将预训练图像分类模型和 LSTM 网络相结合来创建视频分类网络。

要为视频分类创建深度学习网络，请执行以下操作：

- 1 使用预训练卷积神经网络（如 GoogLeNet）将视频转换为特征向量序列，以从每帧中提取特征。
- 2 基于序列训练 LSTM 网络来预测视频标签。
- 3 通过合并来自两个网络的层，组合一个直接对视频进行分类的网络。

下图说明网络架构。

- 要将图像序列输入到网络，请使用序列输入层。
- 要使用卷积层来提取特征，也就是说，要将卷积运算独立地应用于视频的每帧，请使用一个后跟卷积层的序列折叠层。
- 要还原序列结构体并将输出重构为向量序列，请使用序列展开层和扁平化层。
- 要对得到的向量序列进行分类，请包括 LSTM 层，并在其后添加输出层。



## 加载预训练卷积网络

要将视频帧转换为特征向量，请使用预训练网络的激活函数。

使用 `googlenet` 函数加载预训练的 GoogLeNet 模型。此函数需要 Deep Learning Toolbox™ Model for GoogLeNet Network 支持包。如果未安装此支持包，则函数会提供下载链接。

```
netCNN = googlenet;
```

## 加载数据

从 HMDB：大型人体运动数据库下载 HMBD51 数据集，并将 RAR 文件提取到名为 "hmdb51\_org" 的文件夹中。该数据集包含 51 个类的 7000 个片段、大约 2 GB 的视频数据，例如 "drink"、"run" 和 "shake\_hands"。

提取 RAR 文件后，使用支持函数 `hmdb51Files` 获取视频的文件名和标签。

```
dataFolder = "hmdb51_org";
[files,labels] = hmdb51Files(dataFolder);
```

使用在此示例末尾定义的 `readVideo` 辅助函数读取第一段视频，并查看该视频的大小。该视频是  $H \times W \times C \times S$  数组，其中  $H$ 、 $W$ 、 $C$  和  $S$  分别是视频的高度、宽度、通道数和帧数。

```
idx = 1;
filename = files(idx);
video = readVideo(filename);
size(video)

ans = 1×4
```

240 320 3 409

查看对应的标签。

```
labels(idx)

ans = categorical
    brush_hair
```

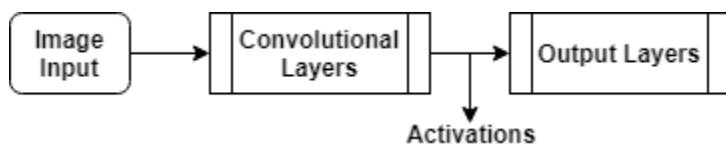
要查看视频，请使用 `implay` 函数（需要 Image Processing Toolbox™）。此函数需要数据在 [0,1] 范围内，因此您必须先将数据除以 255。您也可以在各个帧之间循环，并使用 `imshow` 函数。

```
numFrames = size(video,4);
figure
for i = 1:numFrames
    frame = video(:,:,:i);
    imshow(frame/255);
    drawnow
end
```

### 将帧转换为特征向量

当将视频帧输入到网络时，通过获取激活函数，将卷积网络用作特征提取器。将视频转换为特征向量序列，其中特征向量是 GoogLeNet 网络的最后一个池化层 ("pool5-7x7\_s1") 上 `activations` 函数的输出。

下图说明通过网络的数据流。



要读取视频数据并调整其大小以匹配 GoogLeNet 网络的输入大小，请使用在此示例末尾定义的 `readVideo` 和 `centerCrop` 辅助函数。此步骤可能需要很长时间才能完成运行。在将视频转换为序列后，将序列保存在 `tempdir` 文件夹中的一个 MAT 文件中。如果该 MAT 文件已存在，则从 MAT 文件加载序列，而不必重新转换它们。

```
inputSize = netCNN.Layers(1).InputSize(1:2);
layerName = "pool5-7x7_s1";

tempFile = fullfile(tempdir,"hmdb51_org.mat");

if exist(tempFile,'file')
    load(tempFile,"sequences")
else
    numFiles = numel(files);
    sequences = cell(numFiles,1);

    for i = 1:numFiles
        fprintf("Reading file %d of %d...\n", i, numFiles)

        video = readVideo(files(i));
        video = centerCrop(video,inputSize);

        sequences{i,1} = activations(netCNN,video,layerName,'OutputAs','columns');
    end
```

```

    save(tempFile,"sequences","-v7.3");
end

```

查看前几个序列的大小。每个序列是一个  $D \times S$  数组，其中  $D$  是特征数量（池化层的输出大小）， $S$  是视频的帧数。

**sequences(1:10)**

```

ans = 10×1 cell array
{1024×409 single}
{1024×395 single}
{1024×323 single}
{1024×246 single}
{1024×159 single}
{1024×137 single}
{1024×359 single}
{1024×191 single}
{1024×439 single}
{1024×528 single}

```

## 准备训练数据

通过将数据划分为训练分区和验证分区并删除任何长序列，为训练准备数据。

### 创建训练分区和验证分区

对数据进行分区。将 90% 的数据分配给训练分区，将 10% 分配给验证分区。

```

numObservations = numel(sequences);
idx = randperm(numObservations);
N = floor(0.9 * numObservations);

idxTrain = idx(1:N);
sequencesTrain = sequences(idxTrain);
labelsTrain = labels(idxTrain);

idxValidation = idx(N+1:end);
sequencesValidation = sequences(idxValidation);
labelsValidation = labels(idxValidation);

```

### 删除长序列

比网络中典型序列长得得多的序列会在训练过程中引入大量填充。填充过多会对分类准确度产生负面影响。

获取训练数据的序列长度，并在训练数据的直方图中可视化它们。

```

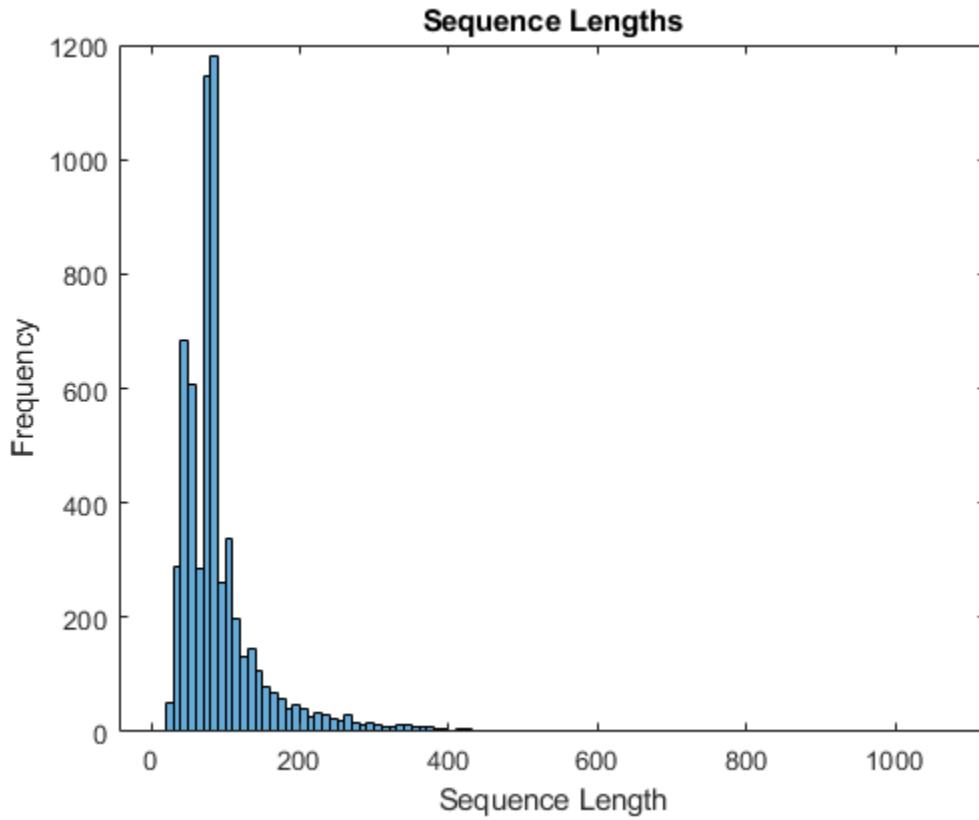
numObservationsTrain = numel(sequencesTrain);
sequenceLengths = zeros(1,numObservationsTrain);

for i = 1:numObservationsTrain
    sequence = sequencesTrain{i};
    sequenceLengths(i) = size(sequence,2);
end

figure
histogram(sequenceLengths)
title("Sequence Lengths")

```

```
xlabel("Sequence Length")
ylabel("Frequency")
```



只有少数序列有超过 400 个时间步。为了提高分类准确度，请删除具有超过 400 个时间步的训练序列及其对应的标签。

```
maxLength = 400;
idx = sequenceLengths > maxLength;
sequencesTrain(idx) = [];
labelsTrain(idx) = [];
```

### 创建 LSTM 网络

接下来，创建一个 LSTM 网络，它可以对表示视频的特征向量的序列进行分类。

定义 LSTM 网络架构。指定以下网络层。

- 序列输入层，其输入大小对应于特征向量的特征维度
- 具有 2000 个隐含单元的 BiLSTM 层，后跟一个丢弃层。通过将 BiLSTM 层的 'OutputMode' 选项设置为 'last'，为每个序列仅输出一个标签
- 输出大小对应于类数量的全连接层、softmax 层和分类层。

```
numFeatures = size(sequencesTrain{1},1);
numClasses = numel(categories(labelsTrain));

layers = [
    sequenceInputLayer(numFeatures,'Name','sequence')
    ...
    fullyConnectedLayer(numClasses,'Name','fc')
    softmaxLayer('Name','softmax')
    classificationLayer('Name','output')]
```

```

bilstmLayer(2000,'OutputMode','last','Name','bilstm')
dropoutLayer(0.5,'Name','drop')
fullyConnectedLayer(numClasses,'Name','fc')
softmaxLayer('Name','softmax')
classificationLayer('Name','classification']);

```

### 指定训练选项

使用 `trainingOptions` 函数指定训练选项。

- 设置小批量大小为 16，初始学习率为 0.0001，梯度阈值为 2（以防止梯度爆炸）。
- 每轮训练都会打乱数据。
- 每轮训练后对网络进行一次验证。
- 在绘图中显示训练进度，并隐藏详细输出。

```

miniBatchSize = 16;
numObservations = numel(sequencesTrain);
numIterationsPerEpoch = floor(numObservations / miniBatchSize);

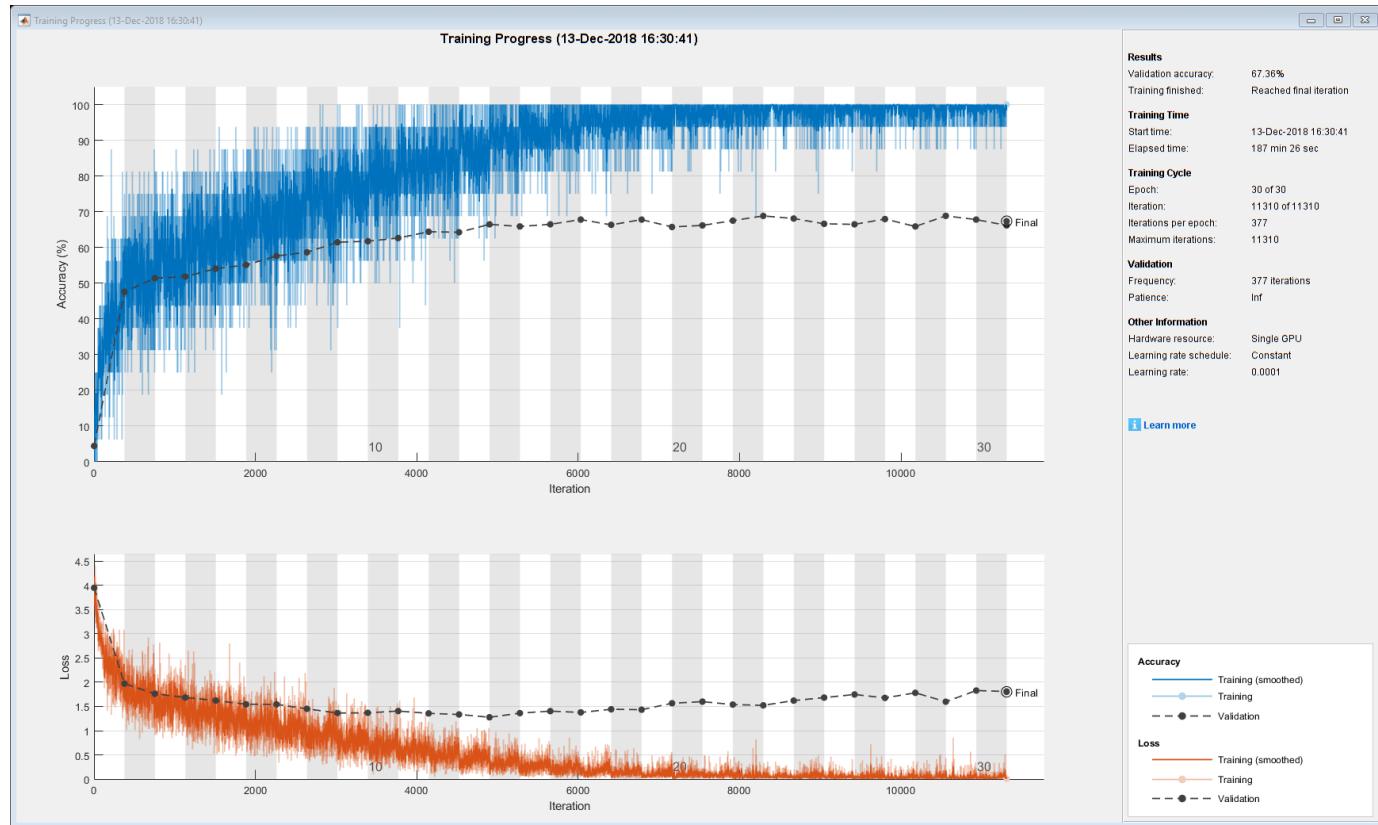
options = trainingOptions('adam',...
    'MiniBatchSize',miniBatchSize,... ...
    'InitialLearnRate',1e-4,... ...
    'GradientThreshold',2,... ...
    'Shuffle','every-epoch',...
    'ValidationData',{sequencesValidation,labelsValidation},...
    'ValidationFrequency',numIterationsPerEpoch,... ...
    'Plots','training-progress',...
    'Verbose',false);

```

### 训练 LSTM 网络

使用 `trainNetwork` 函数训练网络。这可能需要很长时间才能运行完毕。

```
[netLSTM,info] = trainNetwork(sequencesTrain,labelsTrain,layers,options);
```



基于验证集计算网络分类准确度。使用与训练选项相同的小批量大小。

```
YPred = classify(netLSTM,sequencesValidation,'MiniBatchSize',miniBatchSize);
YValidation = labelsValidation;
accuracy = mean(YPred == YValidation)

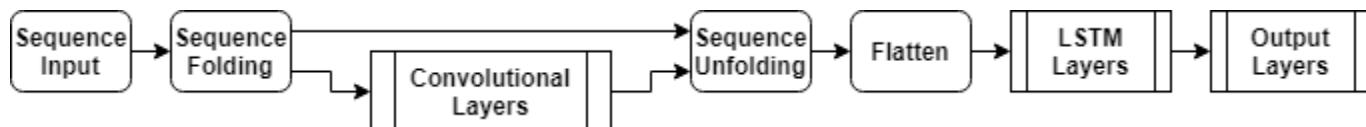
accuracy = 0.6647
```

### 组合视频分类网络

要创建直接对视频进行分类的网络，请用创建的两个网络中的层组合成一个网络。使用来自卷积网络的层将视频变换为向量序列，使用来自 LSTM 网络的层对向量序列进行分类。

下图说明网络架构。

- 要将图像序列输入到网络，请使用序列输入层。
- 要使用卷积层来提取特征，也就是说，要将卷积运算独立地应用于视频的每帧，请使用一个后跟卷积层的序列折叠层。
- 要还原序列结构体并将输出重构为向量序列，请使用序列展开层和扁平化层。
- 要对得到的向量序列进行分类，请包括 LSTM 层，并在其后添加输出层。



### 添加卷积层

首先，创建 GoogLeNet 网络的层次图。

```
cnnLayers = layerGraph(netCNN);
```

删除用于激活的输入层 ("data") 和池化层后面的层 ("pool5-drop\_7x7\_s1"、"loss3-classifier"、"prob" 和 "output") 。

```
layerNames = ["data" "pool5-drop_7x7_s1" "loss3-classifier" "prob" "output"];
cnnLayers = removeLayers(cnnLayers,layerNames);
```

### 添加序列输入层

创建一个序列输入层，它接受包含与 GoogLeNet 网络输入大小相同的图像的图像序列。要使用与 GoogLeNet 网络相同的平均图像归一化图像，请将序列输入层的 'Normalization' 选项设置为 'zerocenter' 选项，将 'Mean' 选项设置为 GoogLeNet 输入层的平均图像。

```
inputSize = netCNN.Layers(1).InputSize(1:2);
averageImage = netCNN.Layers(1).Mean;

inputLayer = sequenceInputLayer([inputSize 3], ...
    'Normalization','zerocenter',...
    'Mean',averageImage,...  

    'Name','input');
```

将序列输入层添加到层次图中。为了将卷积层独立地应用于序列的图像，请通过在序列输入层和卷积层之间包括序列折叠层来删除图像序列的序列结构体。将序列折叠层的输出连接到第一个卷积层 ("conv1-7x7\_s2") 的输入。

```
layers = [
    inputLayer
    sequenceFoldingLayer('Name','fold')];

lgraph = addLayers(cnnLayers,layers);
lgraph = connectLayers(lgraph,"fold/out","conv1-7x7_s2");
```

### 添加 LSTM 层

通过删除 LSTM 网络的序列输入层，将 LSTM 层添加到层次图中。要还原由序列折叠层删除的序列结构体，请在卷积层后包含一个序列展开层。LSTM 层要求使用向量序列。要将序列展开层的输出重构为向量序列，请在序列展开层后包含一个扁平化层。

从 LSTM 网络中提取层，并删除序列输入层。

```
lstmLayers = netLSTM.Layers;
lstmLayers(1) = [];
```

将序列折叠层、扁平化层和 LSTM 层添加到层次图中。将最后一个卷积层 ("pool5-7x7\_s1") 连接到序列展开层 ("unfold/in") 的输入。

```
layers = [
    sequenceUnfoldingLayer('Name','unfold')
    flattenLayer('Name','flatten')
    lstmLayers];

lgraph = addLayers(lgraph,layers);
lgraph = connectLayers(lgraph,"pool5-7x7_s1","unfold/in");
```

要使展开层能够还原序列结构体，请将序列折叠层的 "miniBatchSize" 输出连接到序列展开层的对应输入。

```
lgraph = connectLayers(lgraph,"fold/miniBatchSize","unfold/miniBatchSize");
```

### 组合网络

使用 `analyzeNetwork` 函数检查网络是否有效。

```
analyzeNetwork(lgraph)
```

组合网络，准备就绪，以便使用 `assembleNetwork` 函数进行预测。

```
net = assembleNetwork(lgraph)
```

```
net =  
DAGNetwork with properties:
```

```
Layers: [148×1 nnet.cnn.layer.Layer]  
Connections: [175×2 table]
```

### 使用新数据进行分类

使用与之前相同的步骤读取并居中裁剪视频 "pushup.mp4"。

```
filename = "pushup.mp4";  
video = readVideo(filename);
```

要查看视频，请使用 `implay` 函数（需要 Image Processing Toolbox）。此函数需要数据在 [0,1] 范围内，因此您必须先将数据除以 255。您也可以在各个帧之间循环，并使用 `imshow` 函数。

```
numFrames = size(video,4);  
figure  
for i = 1:numFrames  
    frame = video(:, :, :, i);  
    imshow(frame/255);  
    drawnow  
end
```



使用组合好的网络对视频进行分类。`classify` 函数需要包含输入视频的元胞数组，因此您必须输入包含视频的  $1 \times 1$  元胞数组。

```
video = centerCrop(video,inputSize);
YPred = classify(net,{video})

YPred = categorical
    pushup
```

### 辅助函数

`readVideo` 函数读取 `filename` 中的视频，并返回一个  $H \times W \times C \times S$  数组，其中  $H$ 、 $W$ 、 $C$  和  $S$  分别是视频的高度、宽度、通道数和帧数。

```
function video = readVideo(filename)

vr = VideoReader(filename);
H = vr.Height;
W = vr.Width;
C = 3;

% Preallocate video array
numFrames = floor(vr.Duration * vr.FrameRate);
video = zeros(H,W,C,numFrames);

% Read frames
i = 0;
while hasFrame(vr)
    i = i + 1;
```

```

video(:, :, :, i) = readFrame(vr);
end

```

```

% Remove unallocated frames
if size(video, 4) > i
    video(:, :, :, i+1:end) = [];
end
end

```

**centerCrop** 函数裁剪视频的最长边，并调整其大小，使其具有 **inputSize** 的大小。

```

function videoResized = centerCrop(video, inputSize)

sz = size(video);

if sz(1) < sz(2)
    % Video is landscape
    idx = floor((sz(2) - sz(1))/2);
    video(:, 1:(idx-1), :, :) = [];
    video(:, (sz(1)+1):end, :, :) = [];
else
    if sz(2) < sz(1)
        % Video is portrait
        idx = floor((sz(1) - sz(2))/2);
        video(1:(idx-1), :, :, :) = [];
        video((sz(2)+1):end, :, :, :) = [];
    end
end

videoResized = imresize(video, inputSize(1:2));

```

## 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [lstmLayer](#) | [sequenceInputLayer](#) |  
[sequenceFoldingLayer](#) | [sequenceUnfoldingLayer](#) | [flattenLayer](#)

## 相关示例

- “使用深度学习进行时序预测”（第 4-9 页）
- “使用深度学习进行“序列到序列”分类”（第 4-33 页）
- “使用深度学习进行“序列到序列”回归”（第 4-38 页）
- “Classify Videos Using Deep Learning with Custom Training Loop”
- “长短期记忆网络”（第 1-38 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）

# 使用深度学习对文本数据进行分类

此示例说明如何使用深度学习长短期记忆 (LSTM) 网络对文本数据进行分类。

文本数据本身就是有序的。一段文本是一个单词序列，这些单词之间可能存在依存关系。要学习和使用长期依存关系来对序列数据进行分类，请使用 LSTM 神经网络。LSTM 网络是一种循环神经网络 (RNN)，可以学习序列数据的时间步之间的长期依存关系。

要将文本输入到 LSTM 网络，首先将文本数据转换为数值序列。您可以使用将文档映射为数值索引序列的单词编码来实现此目的。为了获得更好的结果，还要在网络中包含一个单词嵌入层。单词嵌入将词汇表中的单词映射为数值向量而不是标量索引。这些嵌入会捕获单词的语义细节，以便具有相似含义的单词具有相似的向量。它们还通过向量算术运算对单词之间的关系进行建模。例如，关系 "Rome is to Italy as Paris is to France" 通过公式  $\text{Italy} - \text{Rome} + \text{Paris} = \text{France}$  进行描述。

在此示例中，训练和使用 LSTM 网络有四个步骤：

- 导入并预处理数据。
- 使用单词编码将单词转换为数值序列。
- 创建和训练具有单词嵌入层的 LSTM 网络。
- 使用经过训练的 LSTM 网络对新文本数据进行分类。

## 导入数据

导入工厂报告数据。该数据包含已标注的工厂事件文本描述。要将文本数据作为字符串导入，请将文本类型指定为 'string'。

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
head(data)
```

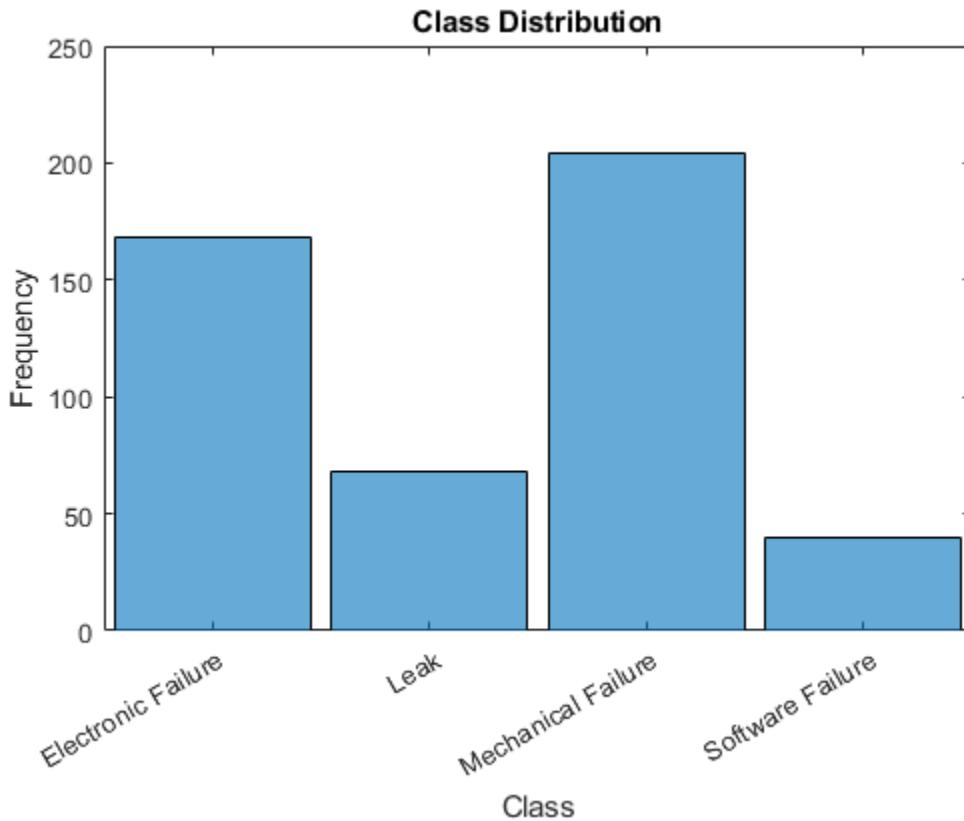
ans=8x5 table	Description	Category	Urgency	Resolution	Cost
"Items are occasionally getting stuck in the scanner spools."		"Mechanical Failure"	"Medium"	"Readjust Spools"	
"Loud rattling and banging sounds are coming from assembler pistons."		"Mechanical Failure"	"Medium"	"Replace Components"	
"There are cuts to the power when starting the plant."		"Electronic Failure"	"High"	"Full Replacement"	
"Fried capacitors in the assembler."		"Electronic Failure"	"High"	"Replace Components"	
"Mixer tripped the fuses."		"Electronic Failure"	"Low"	"Add to Watch List"	55
"Burst pipe in the constructing agent is spraying coolant."		"Leak"	"High"	"Replace Component"	
"A fuse is blown in the mixer."		"Electronic Failure"	"Low"	"Replace Components"	
"Things continue to tumble off of the belt."		"Mechanical Failure"	"Low"	"Readjust Machine"	

此示例的目标是按 Category 列中的标签对事件进行分类。要将数据划分到各个类，请将这些标签转换为分类。

```
data.Category = categorical(data.Category);
```

使用直方图查看数据中类的分布。

```
figure
histogram(data.Category);
xlabel("Class")
ylabel("Frequency")
title("Class Distribution")
```



下一步是将其划分为训练集和验证集。将数据划分为训练分区和用于验证和测试的保留分区。将保留百分比指定为 20%。

```
cvp = cvpartition(data.Category,'Holdout',0.2);
dataTrain = data(training(cvp,:));
dataValidation = data(test(cvp,:));
```

从分区后的表中提取文本数据和标签。

```
textDataTrain = dataTrain.Description;
textDataValidation = dataValidation.Description;
YTrain = dataTrain.Category;
YValidation = dataValidation.Category;
```

要检查是否已正确导入数据，请使用文字云将训练文本数据可视化。

```
figure
wordcloud(textDataTrain);
title("Training Data")
```

## Training Data



## 预处理文本数据

创建一个对文本数据进行分词和预处理的函数。在示例末尾列出的函数 `preprocessText` 执行以下步骤：

- 1 使用 `tokenizedDocument` 对文本进行分词。
  - 2 使用 `lower` 将文本转换为小写。
  - 3 使用 `erasePunctuation` 删除标点符号。

使用 `preprocessText` 函数预处理训练数据和验证数据。

```
documentsTrain = preprocessText(textDataTrain);  
documentsValidation = preprocessText(textDataValidation);
```

查看前几个预处理的训练文档。

documentsTrain(1:5)

ans =

- 9 tokens: items are occasionally getting stuck in the scanner spools
- 10 tokens: loud rattling and banging sounds are coming from assembler pistons
- 10 tokens: there are cuts to the power when starting the plant
- 5 tokens: fried capacitors in the assembler
- 4 tokens: mixer tripped the fuses

## 将文档转换为序列

要将文档输入到 LSTM 网络中，请使用单词编码将文档转换为数值索引序列。

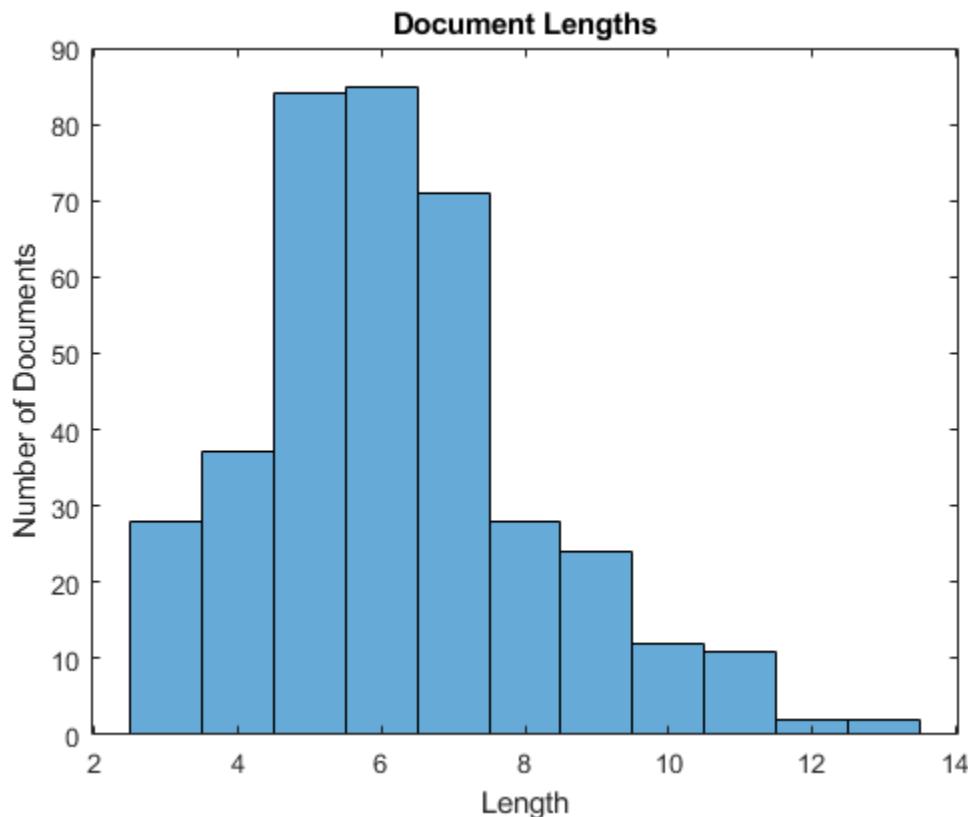
要创建单词编码，请使用 **wordEncoding** 函数。

```
enc = wordEncoding(documentsTrain);
```

下一个转换步骤是填充和截断文档，使全部文档的长度相同。**trainingOptions** 函数提供了自动填充和截断输入序列的选项。但是，这些选项不太适合单词向量序列。请改为手动填充和截断序列。如果对单词向量序列进行左填充和截断，训练效果可能会得到改善。

要填充和截断文档，请先选择目标长度，然后对长于它的文档进行截断，并对短于它的文档进行左填充。为获得最佳结果，目标长度应该较短，但又不至于丢弃大量数据。要找到合适的目标长度，请查看训练文档长度的直方图。

```
documentLengths = doclength(documentsTrain);
figure
histogram(documentLengths)
title("Document Lengths")
xlabel("Length")
ylabel("Number of Documents")
```



大多数训练文档的词数少于 10 个。将此数字用作截断和填充的目标长度。

使用 **doc2sequence** 将文档转换为数值索引序列。要对长度为 10 的序列进行截断或左填充，请将 'Length' 选项设置为 10。

```

sequenceLength = 10;
XTrain = doc2sequence(enc,documentsTrain,'Length',sequenceLength);
XTrain(1:5)

ans=5×1 cell array
{1×10 double}
{1×10 double}
{1×10 double}
{1×10 double}
{1×10 double}

```

使用相同选项将验证文档转换为序列。

```
XValidation = doc2sequence(enc,documentsValidation,'Length',sequenceLength);
```

## 创建和训练 LSTM 网络

定义 LSTM 网络架构。要将序列数据输入到网络中，请包含一个序列输入层并将输入大小设置为 1。接下来，包含一个维度为 50 且与单词编码具有相同单词数的单词嵌入层。然后，包含一个 LSTM 层并将隐含单元个数设置为 80。要将该 LSTM 层用于“序列到标签”分类问题，请将输出模式设置为 'last'。最后，添加一个大小与类数相同的全连接层、一个 softmax 层和一个分类层。

```

inputSize = 1;
embeddingDimension = 50;
numHiddenUnits = 80;

numWords = enc.NumWords;
numClasses = numel(categories(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension,numWords)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]

layers =
6x1 Layer array with layers:

```

1	" Sequence Input "	Sequence input with 1 dimensions
2	" Word Embedding Layer "	Word embedding layer with 50 dimensions and 423 unique words
3	" LSTM "	LSTM with 80 hidden units
4	" Fully Connected "	4 fully connected layer
5	" Softmax "	softmax
6	" Classification Output "	crossentropyex

## 指定训练选项

指定训练选项：

- 使用 Adam 求解器进行训练。
- 指定小批量大小为 16。
- 每轮训练都会打乱数据。
- 通过将 'Plots' 选项设置为 'training-progress'，监控训练进度。

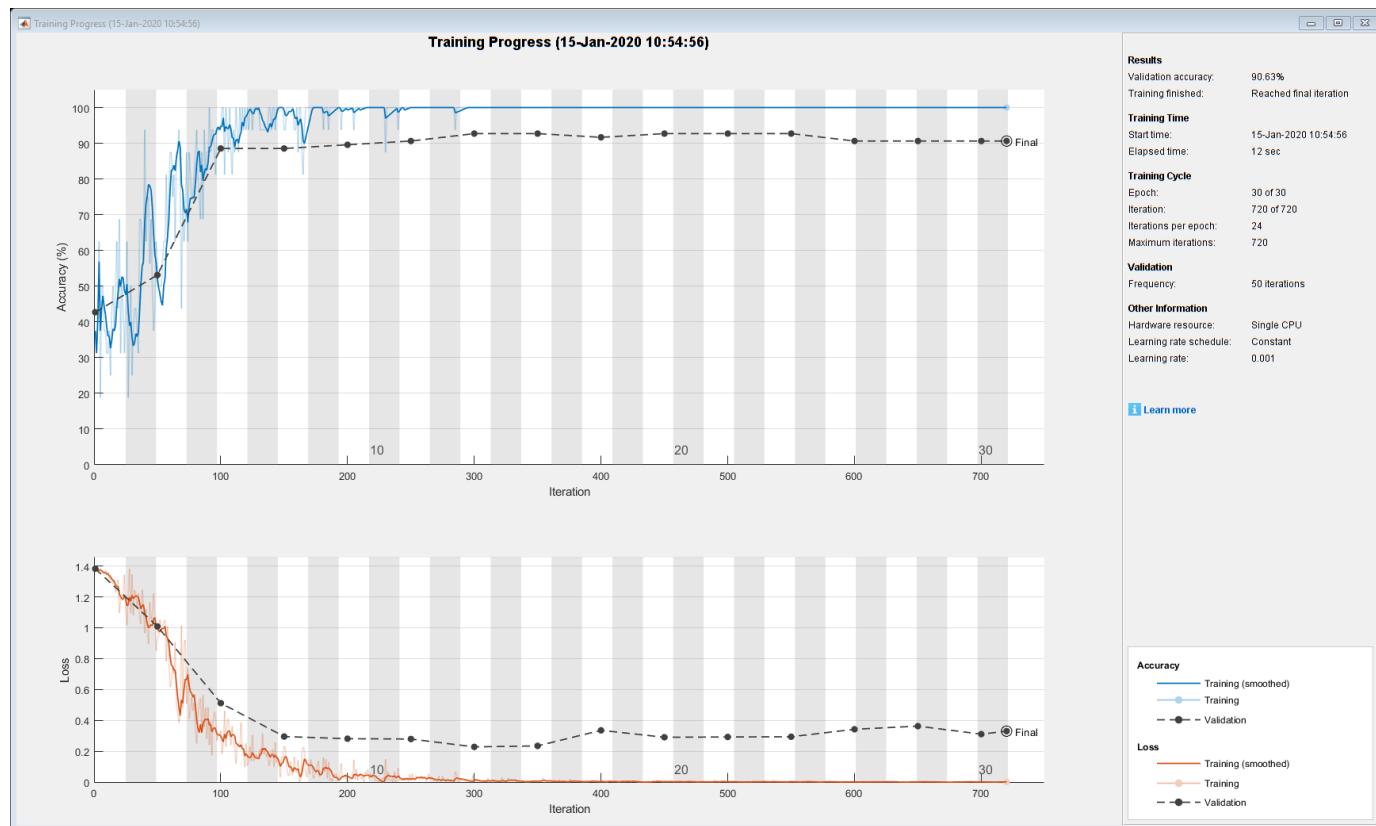
- 使用 'ValidationData' 选项指定验证数据。
- 通过将 'Verbose' 选项设置为 `false`, 隐藏详细输出。

默认情况下, `trainNetwork` 使用 GPU (如果有)。否则将使用 CPU。要手动指定执行环境, 请使用 `trainingOptions` 的 'ExecutionEnvironment' 名称-值对组参数。在 CPU 上进行训练所需的时间要明显长于在 GPU 上进行训练所需的时间。使用 GPU 进行训练需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息, 请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。

```
options = trainingOptions('adam', ...
    'MiniBatchSize',16, ...
    'GradientThreshold',2, ...
    'Shuffle','every-epoch', ...
    'ValidationData',{XValidation,YValidation}, ...
    'Plots','training-progress', ...
    'Verbose',false);
```

使用 `trainNetwork` 函数训练 LSTM 网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



### 使用新数据进行预测

对三个新报告的事件类型进行分类。创建包含新报告的字符串数组。

```
reportsNew = [ ...
    "Coolant is pooling underneath sorter."]
```

```
"Sorter blows fuses at start up."
"There are some very loud rattling sounds coming from the assembler."];
```

使用与预处理训练文档相同的步骤来预处理文本数据。

```
documentsNew = preprocessText(reportsNew);
```

使用 **doc2sequence** 将文本数据转换为序列，所用选项与创建训练序列时的选项相同。

```
XNew = doc2sequence(enc,documentsNew,'Length',sequenceLength);
```

使用经过训练的 LSTM 网络对新序列进行分类。

```
labelsNew = classify(net,XNew)
```

```
labelsNew = 3×1 categorical
```

```
Leak
```

```
Electronic Failure
```

```
Mechanical Failure
```

## 预处理函数

函数 **preprocessText** 执行以下步骤：

- 1 使用 **tokenizedDocument** 对文本进行分词。
- 2 使用 **lower** 将文本转换为小写。
- 3 使用 **erasePunctuation** 删除标点符号。

```
function documents = preprocessText(textData)

% Tokenize the text.
documents = tokenizedDocument(textData);

% Convert to lowercase.
documents = lower(documents);

% Erase punctuation.
documents = erasePunctuation(documents);

end
```

## 另请参阅

[fastTextWordEmbedding](#) | [wordEmbeddingLayer](#) | [tokenizedDocument](#) | [lstmLayer](#) | [trainNetwork](#) | [trainingOptions](#) | [doc2sequence](#) | [sequenceInputLayer](#) | [wordcloud](#)

## 相关示例

- “使用深度学习生成文本” (第 4-72 页)
- “Word-By-Word Text Generation Using Deep Learning” (Text Analytics Toolbox)
- “Classify Out-of-Memory Text Data Using Custom Mini-Batch Datastore” (Text Analytics Toolbox)
- “Create Simple Text Model for Classification” (Text Analytics Toolbox)
- “Analyze Text Data Using Topic Models” (Text Analytics Toolbox)

- “Analyze Text Data Using Multiword Phrases” (Text Analytics Toolbox)
- “Train a Sentiment Classifier” (Text Analytics Toolbox)
- “使用深度学习进行序列分类” (第 4-2 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 使用卷积神经网络对文本数据进行分类

此示例说明如何使用卷积神经网络对文本数据进行分类。

要使用卷积对文本数据进行分类，必须将文本数据转换为图像。为此，请填充或截断观测值，使其具有恒定长度  $S$ ，并使用单词嵌入将文档转换为长度为  $C$  的单词向量序列。然后，您可以将文档表示为  $1 \times S \times C$  的图像（高度为 1、宽度为  $S$  且具有  $C$  个通道的图像）。

要将 CSV 文件中的文本数据转换为图像，请创建一个 `tabularTextDatastore` 对象。通过使用自定义变换函数调用 `transform`，将从 `tabularTextDatastore` 对象读取的数据转换为图像以便进行深度学习。在示例末尾列出的 `transformTextData` 函数接受从数据存储中读取的数据和一个预训练的单词嵌入，并将每个观测值转换为单词向量数组。

此示例训练具有不同宽度的一维卷积滤波器的网络。每个滤波器的宽度对应于滤波器可以检测到的单词数（ $n$  元分词长度）。网络有多个卷积层分支，因此它可以使用不同  $n$  元分词长度。

## 加载预训练的单词嵌入

加载预训练的 fastText 单词嵌入。此函数需要 Text Analytics Toolbox™ Model for fastText English 16 Billion Token Word Embedding 支持包。如果未安装此支持包，则函数会提供下载链接。

```
emb = fastTextWordEmbedding;
```

## 加载数据

根据 `factoryReports.csv` 中的数据创建一个表格文本数据存储。仅读取 "Description" 和 "Category" 列中的数据。

```
filenameTrain = "factoryReports.csv";
textName = "Description";
labelName = "Category";
ttdsTrain = tabularTextDatastore(filenameTrain,'SelectedVariableNames',[textName labelName]);
```

预览数据存储。

```
ttdsTrain.ReadSize = 8;
preview(ttdsTrain)
```

	Description	Category
{'Items are occasionally getting stuck in the scanner spools.'		{'Mechanical Failure'}
{'Loud rattling and banging sounds are coming from assembler pistons.'}		{'Mechanical Failure'}
{'There are cuts to the power when starting the plant.'		{'Electronic Failure'}
{'Fried capacitors in the assembler.'		{'Electronic Failure'}
{'Mixer tripped the fuses.'		{'Electronic Failure'}
{'Burst pipe in the constructing agent is spraying coolant.'		{'Leak'}
{'A fuse is blown in the mixer.'		{'Electronic Failure'}
{'Things continue to tumble off of the belt.'		{'Mechanical Failure'}

创建一个自定义变换函数，该函数将从数据存储中读取的数据转换为包含预测变量和响应的表。在示例末尾列出的 `transformTextData` 函数接受从 `tabularTextDatastore` 对象读取的数据，并返回包含预测变量和响应的表。预测变量是由单词嵌入 `emb` 给出的  $1 \times \text{sequenceLength} \times C$  单词向量数组，其中  $C$  是嵌入维度。这些响应是 `classNames` 中的类的分类标签。

使用在示例末尾列出的 `readLabels` 函数从训练数据中读取标签，并找出具有唯一性的类名。

```
labels = readLabels(ttdsTrain,labelName);
classNames = unique(labels);
numObservations = numel(labels);
```

使用 `transformTextData` 函数变换数据存储，并将序列长度指定为 14。

```
sequenceLength = 14;
tdsTrain = transform(ttdsTrain, @(data) transformTextData(data,sequenceLength,emb,classNames))
```

```
tdsTrain =
    TransformedDatastore with properties:
```

```
UnderlyingDatastore: [1×1 matlab.io.datastore.TabularTextDatastore]
SupportedOutputFormats: ["txt" "csv" "xlsx" "xls" "parquet" "parq" "png" "jpg" "jpeg" "tif" "tiff" "v"]
Transforms: {@(data)transformTextData(data,sequenceLength,emb,classNames)}
IncludeInfo: 0
```

预览变换后的数据存储。预测变量是  $1 \times S \times C$  数组，其中  $S$  是序列长度， $C$  是特征数（嵌入维度）。响应是分类标签。

```
preview(tdsTrain)
```

```
ans=8×2 table
  Predictors      Responses
  _____
  {1×14×300 single}  Mechanical Failure
  {1×14×300 single}  Mechanical Failure
  {1×14×300 single}  Electronic Failure
  {1×14×300 single}  Electronic Failure
  {1×14×300 single}  Electronic Failure
  {1×14×300 single}  Leak
  {1×14×300 single}  Electronic Failure
  {1×14×300 single}  Mechanical Failure
```

## 定义网络架构

为分类任务定义网络架构。

以下步骤说明如何定义网络架构。

- 指定  $1 \times S \times C$  的输入大小，其中  $S$  是序列长度， $C$  是特征数（嵌入维度）。
- 对于  $n$  元分词长度 2、3、4 和 5，创建包含卷积层、批量归一化层、ReLU 层、丢弃层和最大池化层的层块。
- 对于每个块，指定 200 个大小为  $1 \times N$  的卷积滤波器和大小为  $1 \times S$  的池化区域，其中  $N$  是  $n$  元分词长度。
- 将输入层连接到每个块，并使用深度连接层串联各块的输出。
- 要对输出进行分类，请包括一个输出大小为  $K$  的全连接层、一个 softmax 层和一个分类层，其中  $K$  是类的数量。

首先，在一个层数组中，指定输入层、首个一元分词块、深度串联层、全连接层、softmax 层和分类层。

```

numFeatures = emb.Dimension;
inputSize = [1 sequenceLength numFeatures];
numFilters = 200;

ngramLengths = [2 3 4 5];
numBlocks = numel(ngramLengths);

numClasses = numel(classNames);

```

创建一个包含输入层的层次图。将归一化选项设置为 'none'，层名称设置为 'input'。

```

layer = imageInputLayer(inputSize,'Normalization','none','Name','input');
lgraph = layerGraph(layer);

```

对于每个 n 元分词长度，创建一个由卷积层、批量归一化层、ReLU 层、丢弃层和最大池化层构成的块。  
将每个块连接到输入层。

```

for j = 1:numBlocks
    N = ngramLengths(j);

    block = [
        convolution2dLayer([1 N],numFilters,'Name',"conv"+N,'Padding','same')
        batchNormalizationLayer('Name',"bn"+N)
        reluLayer('Name',"relu"+N)
        dropoutLayer(0.2,'Name',"drop"+N)
        maxPooling2dLayer([1 sequenceLength],'Name',"max"+N)];

    lgraph = addLayers(lgraph,block);
    lgraph = connectLayers(lgraph,'input',"conv"+N);
end

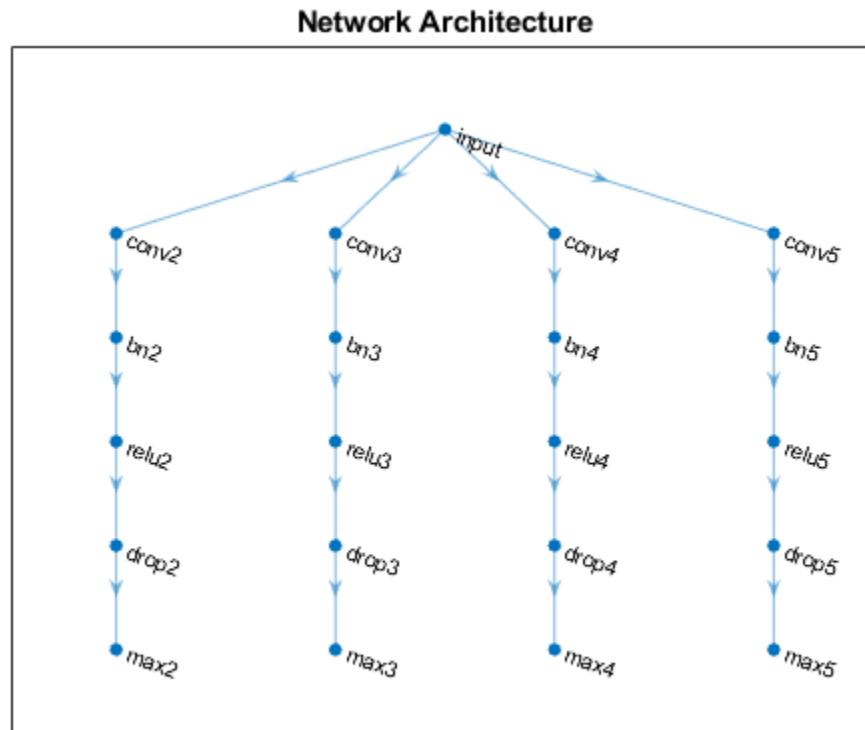
```

在图中查看网络架构。

```

figure
plot(lgraph)
title("Network Architecture")

```



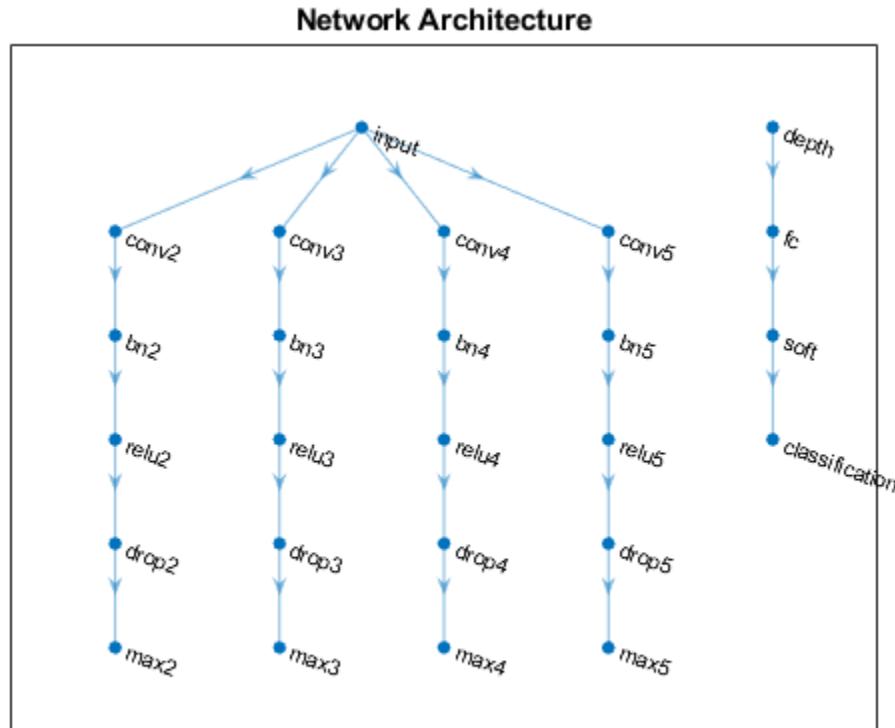
添加深度串联层、全连接层、softmax 层和分类层。

```

layers = [
    depthConcatenationLayer(numBlocks,'Name','depth')
    fullyConnectedLayer(numClasses,'Name','fc')
    softmaxLayer('Name','soft')
    classificationLayer('Name','classification')];

lgraph = addLayers(lgraph,layers);

figure
plot(lgraph)
title("Network Architecture")
  
```

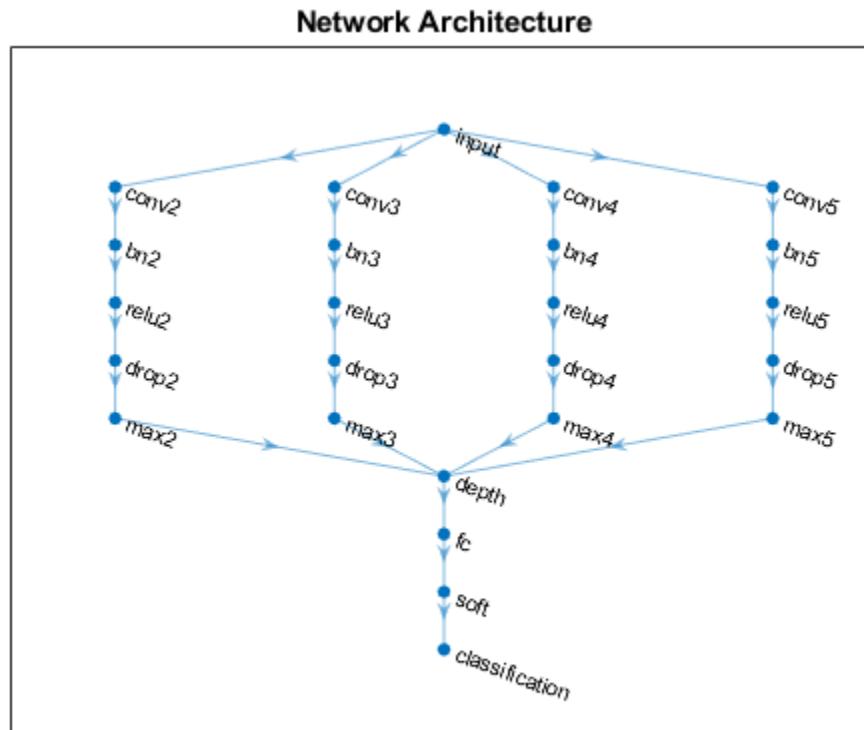


将最大池化层连接到深度串联层，并在图中查看最终网络架构。

```

for j = 1:numBlocks
    N = ngramLengths(j);
    lgraph = connectLayers(lgraph,"max"+N,"depth/in"+j);
end

figure
plot(lgraph)
title("Network Architecture")
  
```



## 训练网络

指定训练选项：

- 使用小批量大小 128 进行训练。
- 不要打乱数据，因为数据存储不可乱序。
- 显示训练进度图并隐藏详细输出。

```

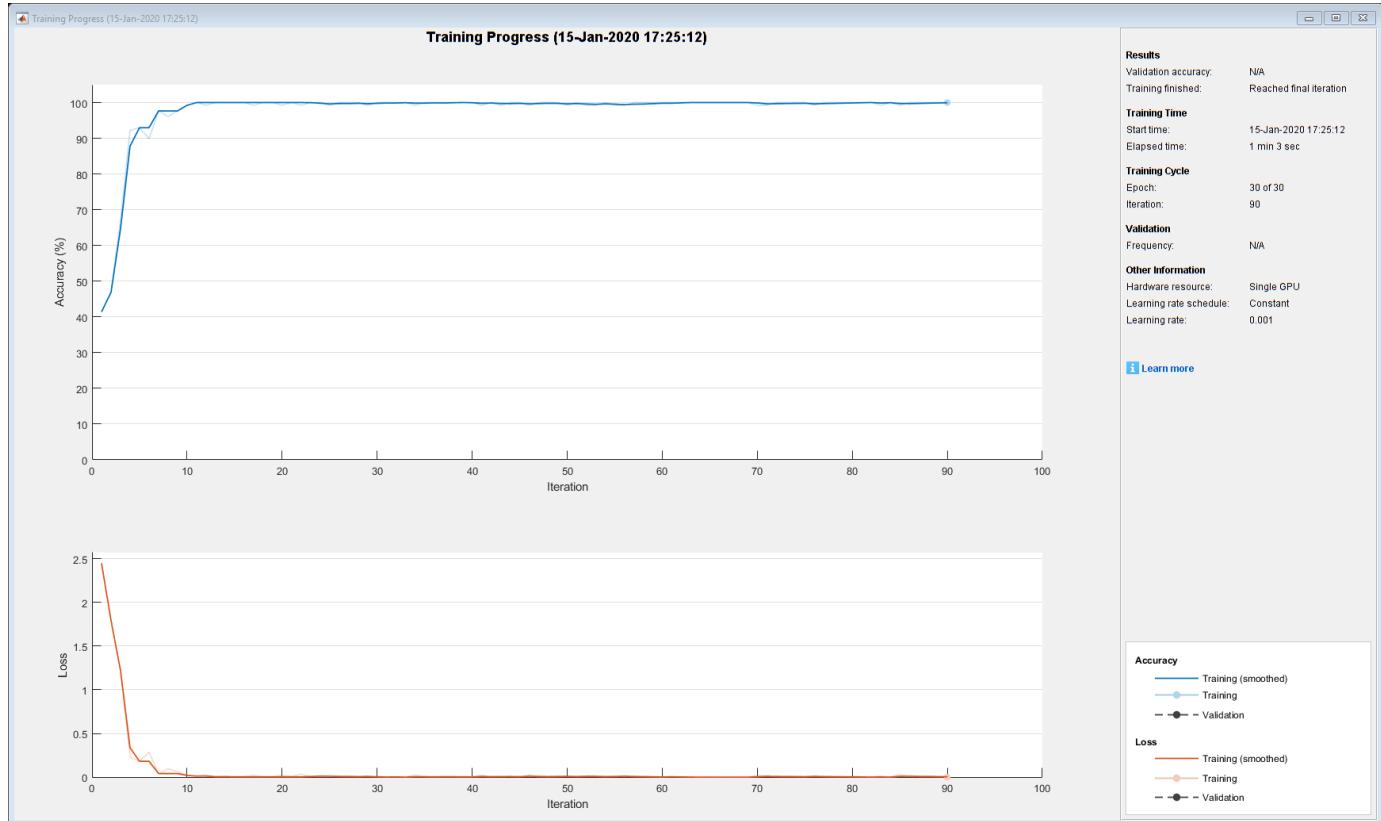
miniBatchSize = 128;
numIterationsPerEpoch = floor(numObservations/miniBatchSize);

options = trainingOptions('adam',...
    'MiniBatchSize',miniBatchSize,...  

    'Shuffle','never',...
    'Plots','training-progress',...
    'Verbose',false);
  
```

使用 **trainNetwork** 函数训练网络。

```
net = trainNetwork(tdsTrain,lgraph,options);
```



## 使用新数据进行预测

对三个新报告的事件类型进行分类。创建包含新报告的字符串数组。

```
reportsNew = [
    "Coolant is pooling underneath sorter."
    "Sorter blows fuses at start up."
    "There are some very loud rattling sounds coming from the assembler."];
```

使用与预处理训练文档相同的步骤来预处理文本数据。

```
XNew = preprocessText(reportsNew,sequenceLength,emb);
```

使用经过训练的 LSTM 网络对新序列进行分类。

```
labelsNew = classify(net,XNew)
```

labelsNew = 3×1 categorical

Leak

Electronic Failure

Mechanical Failure

## 读取标签函数

`readLabels` 函数创建 `tabularTextDatastore` 对象 `ttds` 的一个副本，并读取 `labelName` 列中的标签。

```
function labels = readLabels(ttds,labelName)
```

```
ttdsNew = copy(ttds);
ttdsNew.SelectedVariableNames = labelName;
tbl = readall(ttdsNew);
labels = tbl.(labelName);

end
```

### 变换文本数据函数

`transformTextData` 函数获取从 `tabularTextDatastore` 对象读取的数据，并返回包含预测变量和响应的表。预测变量是由单词嵌入 `emb` 给出的  $1 \times \text{sequenceLength} \times C$  单词向量数组，其中  $C$  是嵌入维度。这些响应是 `classNames` 中的类的分类标签。

```
function dataTransformed = transformTextData(data,sequenceLength,emb,classNames)

% Preprocess documents.
textData = data{:,:};

% Preprocess text
dataTransformed = preprocessText(textData,sequenceLength,emb);

% Read labels.
labels = data{:,2};
responses = categorical(labels,classNames);

% Convert data to table.
dataTransformed.Responses = responses;

end
```

### 预处理文本函数

`preprocessTextData` 函数获取文本数据、序列长度和单词嵌入，并执行以下步骤：

- 1 对文本进行分词。
- 2 将文本转换为小写。
- 3 使用嵌入将文档转换为指定长度的单词向量序列。
- 4 重构单词向量序列以输入到网络中。

```
function tbl = preprocessText(textData,sequenceLength,emb)

documents = tokenizedDocument(textData);
documents = lower(documents);

% Convert documents to embeddingDimension-by-sequenceLength-by-1 images.
predictors = doc2sequence(emb,documents,'Length',sequenceLength);

% Reshape images to be of size 1-by-sequenceLength-embeddingDimension.
predictors = cellfun(@(X) permute(X,[3 2 1]),predictors,'UniformOutput',false);

tbl = table;
tbl.Predictors = predictors;
```

**end**

## 另请参阅

[fastTextWordEmbedding](#) | [wordcloud](#) | [wordEmbedding](#) | [layerGraph](#) |  
[convolution2dLayer](#) | [batchNormalizationLayer](#) | [trainingOptions](#) | [trainNetwork](#) |  
[doc2sequence](#) | [tokenizedDocument](#) | [transform](#)

## 相关示例

- “Classify Text Data Using Deep Learning” (Text Analytics Toolbox)
- “Classify Out-of-Memory Text Data Using Custom Mini-Batch Datastore” (Text Analytics Toolbox)
- “Create Simple Text Model for Classification” (Text Analytics Toolbox)
- “Analyze Text Data Using Topic Models” (Text Analytics Toolbox)
- “Analyze Text Data Using Multiword Phrases” (Text Analytics Toolbox)
- “Train a Sentiment Classifier” (Text Analytics Toolbox)
- “使用深度学习进行序列分类” (第 4-2 页)
- “Datastores for Deep Learning”
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 使用深度学习生成文本

此示例说明如何训练深度学习长短期记忆 (LSTM) 网络以生成文本。

要训练深度学习网络以生成文本，请训练“序列到序列”的 LSTM 网络，以预测字符序列中的下一个字符。要训练网络以预测下一个字符，请将移位一个时间步的输入序列指定为响应。

要将字符序列输入到 LSTM 网络中，请将每个训练观测值转换为由向量  $x \in \mathbb{R}^D$  表示的字符序列，其中 D 是词汇表中唯一字符的数量。对于每个向量，如果  $x$  对应于给定词汇表中索引为  $i$  的字符，则  $x_i = 1$ ；如果  $j \neq i$ ，则  $x_j = 0$ 。

### 加载训练数据

从文本文件 `sonnets.txt` 中提取文本数据。

```
filename = "sonnets.txt";
textData = fileread(filename);
```

十四行诗缩进两个空白字符，并用两个换行符分隔。使用 `replace` 删除缩进，并使用 `split` 将文本拆分为单独的十四行诗。删除前三个元素中的主标题和每一首十四行诗之前出现的十四行诗标题。

```
textData = replace(textData, " ", "");
textData = split(textData,[newline newline]);
textData = textData(5:2:end);
```

查看前几个观测值。

`textData(1:10)`

```
ans = 10×1 cell array
{'From fairest creatures we desire increase, —That thereby beauty's rose might never die, —But as the riper sh...'}
{'When forty winters shall besiege thy brow, —And dig deep trenches in thy beauty's field, —Thy youth's proud...'}
{'Look in thy glass and tell the face thou viewest. —Now is the time that face should form another; —Whose fresh...'}
{'Unthrifty loveliness, why dost thou spend —Upon thy self thy beauty's legacy? —Nature's bequest gives nothin...'}
{'Those hours, that with gentle work did frame —The lovely gaze where every eye doth dwell, —Will play the ty...'}
{'Then let not winter's ragged hand deface, —In thee thy summer, ere thou be distill'd: —Make sweet some vial;...'}
{'Lo! in the orient when the gracious light —Lifts up his burning head, each under eye —Doth homage to his ne...'}
{'Music to hear, why hear'st thou music sadly? —Sweets with sweets war not, joy delights in joy: —Why lov'st th...'}
{'Is it for fear to wet a widow's eye, —That thou consum'st thy self in single life? —Ah! if thou issueless shalt hap...'}
{'For shame! deny that thou bear'st love to any, —Who for thy self art so unprovident. —Grant, if thou wilt, thou...
```

### 将文本数据转换为序列

将文本数据转换为预测变量的向量序列和响应的分类序列。

创建特殊字符来表示“文本开始”、“空白”、“文本结束”和“换行符”。分别使用特殊字符 "`\x0002`" (文本开始)、"`\x00B7`" ( "`.`"，间隔点)、"`\x2403`" ( "`ETX`"，文本结束) 和 "`\x00B6`" ( "`¶`"，段落符号)。为防止出现歧义，您必须选择文本中未出现的特殊字符。由于这些字符未出现在训练数据中，因此可用于此目的。

```
startOfTextCharacter = compose("\x0002");
whitespaceCharacter = compose("\x00B7");
endOfTextCharacter = compose("\x2403");
newlineCharacter = compose("\x00B6");
```

对于每个观测值，在开头插入文本开始字符，并用对应的字符替换空白和换行符。

```
textData = startOfTextCharacter + textData;
textData = replace(textData,[" newline],[whitespaceCharacter newlineCharacter]);
```

创建文本中唯一字符的词汇表。

```
uniqueCharacters = unique([textData{:}]);
numUniqueCharacters = numel(uniqueCharacters);
```

循环处理文本数据，并创建表示每个观测值的字符的向量序列以及响应的字符分类序列。要表示每个观测值的结束，请包含文本结束字符。

```
numDocuments = numel(textData);
XTrain = cell(1,numDocuments);
YTrain = cell(1,numDocuments);
for i = 1:numel(textData)
    characters = textData{i};
    sequenceLength = numel(characters);

    % Get indices of characters.
    [~,idx] = ismember(characters,uniqueCharacters);

    % Convert characters to vectors.
    X = zeros(numUniqueCharacters,sequenceLength);
    for j = 1:sequenceLength
        X(idx(j),j) = 1;
    end

    % Create vector of categorical responses with end of text character.
    charactersShifted = [cellstr(characters(2:end))' endOfTextCharacter];
    Y = categorical(charactersShifted);

    XTrain{i} = X;
    YTrain{i} = Y;
end
```

查看第一个观测值和相应序列的大小。该序列是一个  $D \times S$  矩阵，其中  $D$  是特征数（唯一字符的数量）， $S$  是序列长度（文本中的字符数量）。

```
textData{1}
```

```
ans =
'From fairest creatures we desire increase, ¶ That thereby beauty's rose might never die, ¶ But as the riper shou
size(XTrain{1})
```

```
ans = 1×2
```

```
62 611
```

查看相应的响应序列。该序列是由响应组成的  $1 \times S$  分类向量。

```
YTrain{1}
```

```
ans = 1×611 categorical array
F   r   o   m   .   f   a   i   r   e   s   t   .   c   r   e   a   t   u   r   e   s   .   w   e
```

## 创建和训练 LSTM 网络

定义 LSTM 架构。指定一个“序列到序列”LSTM 分类网络，其中包含 200 个隐含单元。将训练数据的特征维度（唯一字符的数量）设置为输入大小，将响应中的类别数量设置为全连接层的输出大小。

```
inputSize = size(XTrain{1},1);
numHiddenUnits = 200;
numClasses = numel(categories([YTrain{:}]));

layers = [
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits,'OutputMode','sequence')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

使用 `trainingOptions` 函数指定训练选项。将训练轮数指定为 500，将初始学习率指定为 0.01。要防止梯度爆炸，请将梯度阈值设置为 2。通过将 `'Shuffle'` 选项设置为 `'every-epoch'`，指定在每轮对数据进行乱序处理。要监控训练进度，请将 `'Plots'` 选项设置为 `'training-progress'`。要隐藏详细输出，请将 `'Verbose'` 设置为 `false`。

小批量大小选项指定一次迭代要处理的观测值数量。请指定能够均分数据的小批量大小，以确保函数使用全部观测值进行训练。否则，函数将忽略不能完成一个小批量的观测值。将小批量大小设置为 77。

```
options = trainingOptions('adam',...
    'MaxEpochs',500,...
    'InitialLearnRate',0.01,...
    'GradientThreshold',2,...
    'MiniBatchSize',77,...
    'Shuffle','every-epoch',...
    'Plots','training-progress',...
    'Verbose',false);
```

训练网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



## 生成新文本

使用示例末尾列出的 `generateText` 函数，使用经过训练的网络生成文本。

`generateText` 函数逐字符生成文本，从文本开始字符开始，并使用特殊字符重新构造文本。该函数使用输出预测分数对每个字符进行采样。当网络预测到文本结束字符或生成的文本长度为 500 个字符时，该函数停止预测。

使用经过训练的网络生成文本。

```
generatedText = generateText(net,uniqueCharacters,startOfTextCharacter,newlineCharacter,whitespaceCharacter)
generatedText =
"Look, that your lepperites of such soous toor men,
Where than proud on your sweetest but lever ill lie.
One of Death a deal doth teal hearts come,
And that which gives did mistress one learn
Made mens of tongue that hands hear,
And all they with me, do I fortune to brief;
And every peinted could with this right ampontion sorend
By genilir'd lime thou hours, and wonder sposing,
And night by day you waster'd then new;
For ailing thuse borrowest vein fulse were of here spent,
Since my heart morey "
```

## 文本生成函数

`generateText` 函数逐字符生成文本，从文本开始字符开始，并使用特殊字符重新构造文本。该函数使用输出预测分数对每个字符进行采样。当网络预测到文本结束字符或生成的文本长度为 500 个字符时，该函数停止预测。

```
function generatedText = generateText(net,uniqueCharacters,startOfTextCharacter,newlineCharacter,whitespaceCharacter)
```

通过查找文本开始字符的索引来创建其向量。

```
numUniqueCharacters = numel(uniqueCharacters);
X = zeros(numUniqueCharacters,1);
idx = strfind(uniqueCharacters,startOfTextCharacter);
X(idx) = 1;
```

使用经过训练的 LSTM 网络，使用 `predictAndUpdateState` 和 `datasample` 逐字符生成文本。当网络预测到文本结束字符或生成的文本长度为 500 个字符时，停止预测。`datasample` 函数需要 Statistics and Machine Learning Toolbox™。

对于大型数据集合、长序列或大型网络，在 GPU 上进行预测计算通常比在 CPU 上快。其他情况下，在 CPU 上进行预测计算通常更快。对于单时间步预测，请使用 CPU。要使用 CPU 进行预测，请将 `predictAndUpdateState` 的 'ExecutionEnvironment' 选项设置为 'cpu'。

```
generatedText = "";
vocabulary = string(net.Layers(end).Classes);

maxLength = 500;
while strlength(generatedText) < maxLength
    % Predict the next character scores.
    [net,characterScores] = predictAndUpdateState(net,X,'ExecutionEnvironment','cpu');

    % Sample the next character.
    newCharacter = datasample(vocabulary,1,'Weights',characterScores);

    % Stop predicting at the end of text.
    if newCharacter == endOfTextCharacter
        break
    end

    % Add the character to the generated text.
    generatedText = generatedText + newCharacter;

    % Create a new vector for the next input.
    X() = 0;
    idx = strfind(uniqueCharacters,newCharacter);
    X(idx) = 1;
end
```

通过将特殊字符替换为对应的空白字符和换行符，重新构造生成的文本。

```
generatedText = replace(generatedText,[newlineCharacter whitespaceCharacter],[newline " "]);
```

```
end
```

## 另请参阅

`trainNetwork` | `trainingOptions` | `lstmLayer` | `sequenceInputLayer`

## 相关示例

- “Word-By-Word Text Generation Using Deep Learning” (Text Analytics Toolbox)
- “Pride and Prejudice and MATLAB” (Text Analytics Toolbox)
- “使用深度学习进行时序预测” (第 4-9 页)
- “使用深度学习进行序列分类” (第 4-2 页)
- “使用深度学习进行“序列到序列” 分类” (第 4-33 页)
- “使用深度学习进行“序列到序列” 回归” (第 4-38 页)
- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 《傲慢与偏见》与 MATLAB

此示例说明如何训练深度学习 LSTM 网络来通过字符嵌入生成文本。

要训练深度学习网络以生成文本，请训练“序列到序列”的 LSTM 网络，以预测字符序列中的下一个字符。要训练网络以预测下一个字符，请将响应指定为移位一个时间步的输入序列。

要使用字符嵌入，请将每个训练观测值转换为整数序列，其中的整数对字符词汇表进行索引。在网络中包含一个单词嵌入层，该层学习字符的嵌入并将整数映射到向量。

### 加载训练数据

读取《傲慢与偏见》（简·奥斯汀著）的古腾堡计划电子书的 HTML 代码，并使用 `webread` 和 `htmlTree` 对其进行解析。

```
url = "https://www.gutenberg.org/files/1342/1342-h/1342-h.htm";
code = webread(url);
tree = htmlTree(code);
```

通过查找 `p` 元素来提取段落。使用 CSS 选择器 `:not(.toc)` 指定忽略带有 "toc" 类的段落元素。

```
paragraphs = findElement(tree,'p:not(.toc)');
```

使用 `extractHTMLText` 从段落中提取文本数据，并删除空字符串。

```
textData = extractHTMLText(paragraphs);
textData(textData == "") = [];
```

删除短于 20 个字符的字符串。

```
idx = strlength(textData) < 20;
textData(idx) = [];
```

用文字云可视化文本数据。

```
figure
wordcloud(textData);
title("Pride and Prejudice")
```



## 将文本数据转换为序列

将文本数据转换为预测变量的字符索引序列和响应的分类序列。

分类函数将换行符和空白字符条目视为未定义。要为这些字符创建分类元素，请分别使用特殊字符 "¶" (段落符号, "\x00B6") 和 "·" (间隔号, "\x00B7") 替换它们。为防止出现歧义，您必须选择文本中未出现的特殊字符。这些字符未出现在训练数据中，因此可用于此目的。

```
newlineCharacter = compose("\x00B6");
whitespaceCharacter = compose("\x00B7");
textData = replace(textData,[newline " "],[newlineCharacter whitespaceCharacter]);
```

循环文本数据，并创建表示每个观测值字符的字符索引序列以及响应的字符分类序列。要表示每个观测值的结束，请包含特殊字符 "ETX" (文本结尾, "\x2403")。

```
endOfTextCharacter = compose("\x2403");
numDocuments = numel(textData);
for i = 1:numDocuments
    characters = textData{i};
    X = double(characters);

    % Create vector of categorical responses with end of text character.
    charactersShifted = [cellstr(characters(2:end))' endOfTextCharacter];
    Y = categorical(charactersShifted);

    XTrain{i} = X;
    YTrain{i} = Y;
end
```

在训练过程中，默认情况下，软件将训练数据拆分成小批量并填充序列，使它们具有相同的长度。过多填充会对网络性能产生负面影响。

为了防止训练过程添加过多填充，您可以按序列长度对训练数据进行排序，并选择合适的小批量大小，以使同一个小批量中的序列长度相近。

获取每个观测值的序列长度。

```
numObservations = numel(XTrain);
for i=1:numObservations
    sequence = XTrain{i};
    sequenceLengths(i) = size(sequence,2);
end
```

按序列长度对数据进行排序。

```
[~,idx] = sort(sequenceLengths);
XTrain = XTrain(idx);
YTrain = YTrain(idx);
```

### 创建和训练 LSTM 网络

定义 LSTM 架构。指定一个“序列到序列”LSTM 分类网络，其中包含 400 个隐含单元。将输入大小设置为训练数据的特征维度。对于字符索引序列，特征维度为 1。指定维度为 200 的单词嵌入层，并指定单词数（对应于字符数）为输入数据中的最高字符值。将全连接层的输出大小设置为响应中的类别数。为帮助防止过拟合，在 LSTM 层后而包含一个丢弃层。

单词嵌入层学习字符嵌入并将每个字符映射到一个 200 维向量。

```
inputSize = size(XTrain{1},1);
numClasses = numel(categories([YTrain{:}])); % YTrain is a categorical array
numCharacters = max([textData{:}]); % textData is a character array

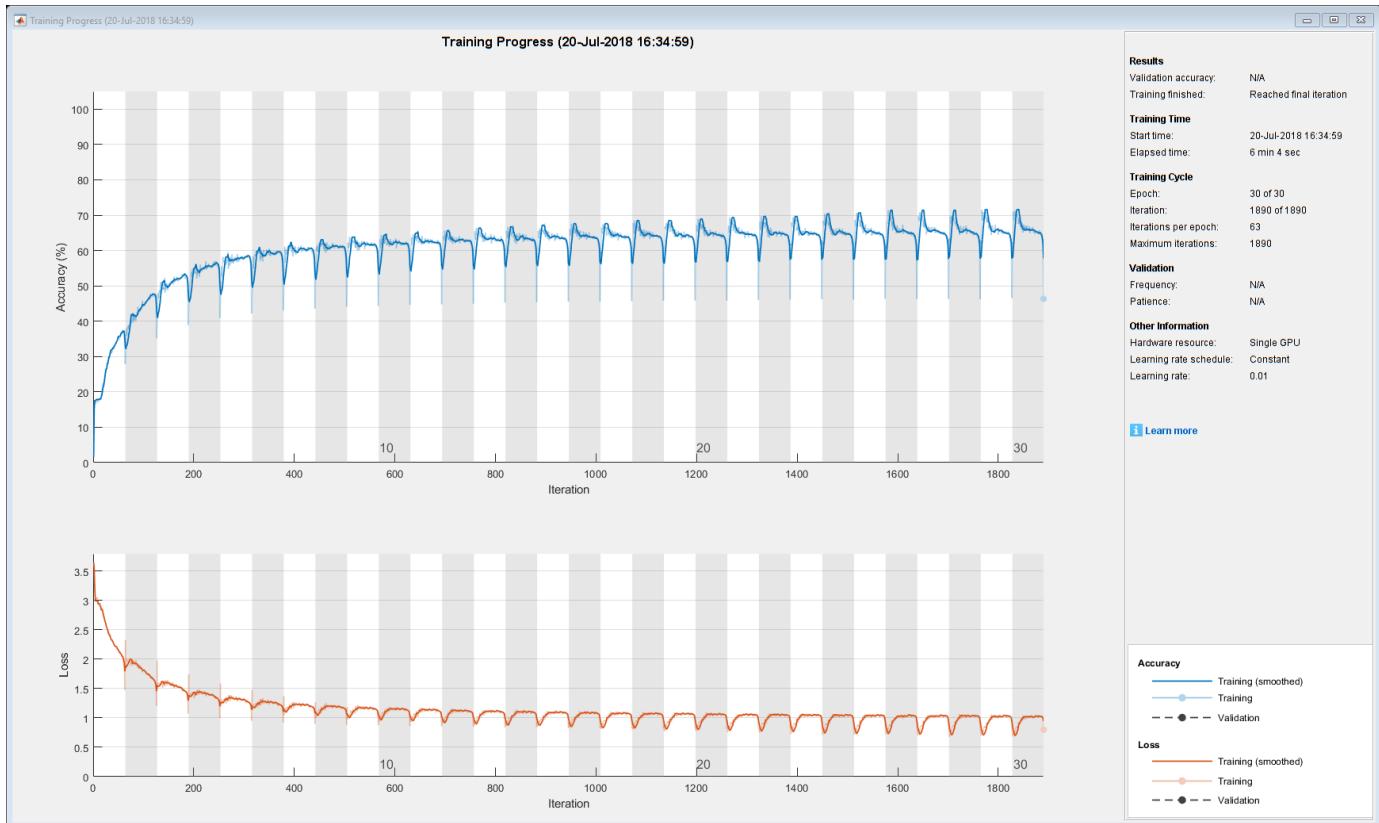
layers = [
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(200,numCharacters)
    lstmLayer(400,'OutputMode','sequence')
    dropoutLayer(0.2);
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

指定训练选项。指定以小批量大小 32 和初始学习率 0.01 进行训练。要防止梯度爆炸，请将梯度阈值设置为 1。要确保数据保持排序，请将 'Shuffle' 设置为 'never'。要监控训练进度，请将 'Plots' 选项设置为 'training-progress'。要隐藏详细输出，请将 'Verbose' 设置为 false。

```
options = trainingOptions('adam',...
    'MiniBatchSize',32,...
    'InitialLearnRate',0.01, ...
    'GradientThreshold',1, ...
    'Shuffle','never', ...
    'Plots','training-progress',...
    'Verbose',false);
```

训练网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



## 生成新文本

根据训练数据中文本的所有首字符的概率分布抽取一个字符来生成文本的第一个字符。接着使用经过训练的 LSTM 网络基于当前已生成的文本序列预测下一序列，以生成其余字符。继续逐个生成字符，直到网络预测到“文本结尾”字符。

根据训练数据中所有首字符的分布抽取第一个字符。

```
initialCharacters = extractBefore(textData,2);
firstCharacter = datasample(initialCharacters,1);
generatedText = firstCharacter;
```

将第一个字符转换为数值索引。

```
X = double(char(firstCharacter));
```

在后续的预测中，会根据网络的预测分数对来抽取下一个字符。预测分数表示下一个字符的概率分布。从网络输出层的类名给出的字符词汇表中抽取字符。从网络的分类层获取词汇表。

```
vocabulary = string(net.Layers(end).ClassNames);
```

使用 `predictAndUpdateState` 逐个字符进行预测。对于每次预测，都输入前一个字符的索引。当网络预测到文本结尾字符或生成的文本长度达到 500 个字符时，停止预测。对于大型数据集合、长序列或大型网络，在 GPU 上进行预测计算通常比在 CPU 上快。其他情况下，在 CPU 上进行预测计算通常更快。对于单时间步预测，请使用 CPU。要使用 CPU 进行预测，请将 `predictAndUpdateState` 的 'ExecutionEnvironment' 选项设置为 'cpu'。

```
maxLength = 500;
while strlength(generatedText) < maxLength
```

```
% Predict the next character scores.
[net,characterScores] = predictAndUpdateState(net,X,'ExecutionEnvironment','cpu');

% Sample the next character.
newCharacter = datasample(vocabulary,1,'Weights',characterScores);

% Stop predicting at the end of text.
if newCharacter == endOfTextCharacter
    break
end

% Add the character to the generated text.
generatedText = generatedText + newCharacter;

% Get the numeric index of the character.
X = double(char(newCharacter));
end
```

通过将特殊字符替换为对应的空白字符和换行符，重新构造生成的文本。

```
generatedText = replace(generatedText,[newlineCharacter whitespaceCharacter],[newline " "])

generatedText =
"I wish Mr. Darcy, upon latter of my sort sincerely fixed in the regard to relanth. We were to join on the Lucase"
```

要生成多篇文本，请在每次生成完成后使用 `resetState` 重置网络状态。

```
net = resetState(net);
```

## 另请参阅

`wordEmbeddingLayer` | `doc2sequence` | `tokenizedDocument` | `lstmLayer` | `trainNetwork` |  
`trainingOptions` | `sequenceInputLayer` | `wordcloud` | `extractHTMLText` | `findElement` |  
`htmlTree`

## 相关示例

- “使用深度学习生成文本”（第 4-72 页）
- “Word-By-Word Text Generation Using Deep Learning”（Text Analytics Toolbox）
- “Create Simple Text Model for Classification”（Text Analytics Toolbox）
- “Analyze Text Data Using Topic Models”（Text Analytics Toolbox）
- “Analyze Text Data Using Multiword Phrases”（Text Analytics Toolbox）
- “Train a Sentiment Classifier”（Text Analytics Toolbox）
- “使用深度学习进行序列分类”（第 4-2 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）

# 使用深度学习进行逐单词文本生成

此示例说明如何训练深度学习 LSTM 网络来逐单词生成文本。

要训练深度学习网络以逐单词生成文本，请训练“序列到序列”的 LSTM 网络，以预测单词序列中的下一个单词。要训练网络以预测下一个单词，请将响应指定为移位一个时间步的输入序列。

此示例从网站上读取文本。它读取并解析 HTML 代码以提取相关文本，然后使用自定义的小批量数据存储 **documentGenerationDatastore** 将文档作为小批量序列数据输入网络。数据存储将文档转换为数值单词索引序列。深度学习网络是包含单词嵌入层的 LSTM 网络。

小批量数据存储是支持批量读取数据的数据存储实现。您可以使用小批量数据存储作为深度学习应用程序的训练数据集、验证数据集、测试数据集以及预测数据集的源。使用小批量数据存储可读取无法放入内存的数据，或者在读取批量数据时执行特定的预处理操作。

您可以通过自定义函数来调整自定义小批量数据存储 **documentGenerationDatastore.m**，使之适合您的数据。有关说明如何创建您自己的自定义小批量数据存储的示例，请参阅“Develop Custom Mini-Batch Datastore”。

## 加载训练数据

加载训练数据。从 Project Gutenberg 读取 Alice's Adventures in Wonderland by Lewis Carroll 中的 HTML 代码。

```
url = "https://www.gutenberg.org/files/11/11-h/11-h.htm";
code = webread(url);
```

## 解析 HTML 代码

HTML 代码包含 `<p>`（段落）元素内的相关文本。通过使用 **htmlTree** 解析 HTML 代码，然后找到元素名为 “p”的所有元素，来提取相关文本。

```
tree = htmlTree(code);
selector = "p";
subtrees = findElement(tree,selector);
```

使用 **extractHTMLText** 从 HTML 子树中提取文本数据，并查看前 10 段。

```
textData = extractHTMLText(subtrees);
textData(1:10)
```

```
ans = 10×1 string array
```

```
""
```

```
""
```

```
""
```

```
""
```

```
""
```

```
""
```

```
""
```

```
""
```

```
"Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or  
"So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and s  
"There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the  
"In another moment down went Alice after it, never once considering how in the world she was to get out again
```

删除空段落并查看更新后的前 10 个段落。

```
textData(textData == "") = [];
textData(1:10)
```

ans = 10×1 string array

"Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once on  
"So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and si  
"There was nothing so very remarkable in that; nor did Alice think it so very much out of the way to hear the  
"In another moment down went Alice after it, never once considering how in the world she was to get out again.  
"The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that  
"Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look abo  
"Well!" thought Alice to herself, 'after such a fall as this, I shall think nothing of tumbling down stairs! How br  
"Down, down, down. Would the fall never come to an end? 'I wonder how many miles I've fallen by this time?' s  
"Presently she began again. 'I wonder if I shall fall right through the earth! How funny it'll seem to come out at  
"Down, down, down. There was nothing else to do, so Alice soon began talking again. 'Dinah'll miss me very mu

用文字云可视化文本数据。

```
figure  
wordcloud(textData);  
title("Alice's Adventures in Wonderland")
```



## 准备要训练的数据

使用 `documentGenerationDatastore` 创建包含训练数据的数据存储。要创建数据存储，请先将自定义小批量数据存储 `documentGenerationDatastore.m` 保存到路径。对于预测变量，此数据存储使用单词编码将文档转换为单词索引序列。每个文档的第一个单词索引对应于“文本开始”标记。“文本开始”标记由字符串 "`startOfText`" 给出。作为响应，数据存储返回移位了一个单词的分类序列。

使用 `tokenizedDocument` 对文本数据进行分词。

```
documents = tokenizedDocument(textData);
```

使用分词后的文档创建文档生成数据存储。

```
ds = documentGenerationDatastore(documents);
```

要减少添加到序列中的填充量，请按序列长度对数据存储中的文档进行排序。

```
ds = sort(ds);
```

### 创建和训练 LSTM 网络

定义 LSTM 网络架构。要将序列数据输入到网络中，请包含一个序列输入层并将输入大小设置为 1。接下来，包含一个维度为 100 且与单词编码具有相同单词数的单词嵌入层。接下来，包含一个 LSTM 层并指定隐藏单元个数为 100。最后，添加一个大小与类数相同的全连接层、一个 softmax 层和一个分类层。类的数量是词汇表中的单词数加上一个针对“文本结束”类的额外类。

```
inputSize = 1;
embeddingDimension = 100;
numWords = numel(ds.Encoding.Vocabulary);
numClasses = numWords + 1;

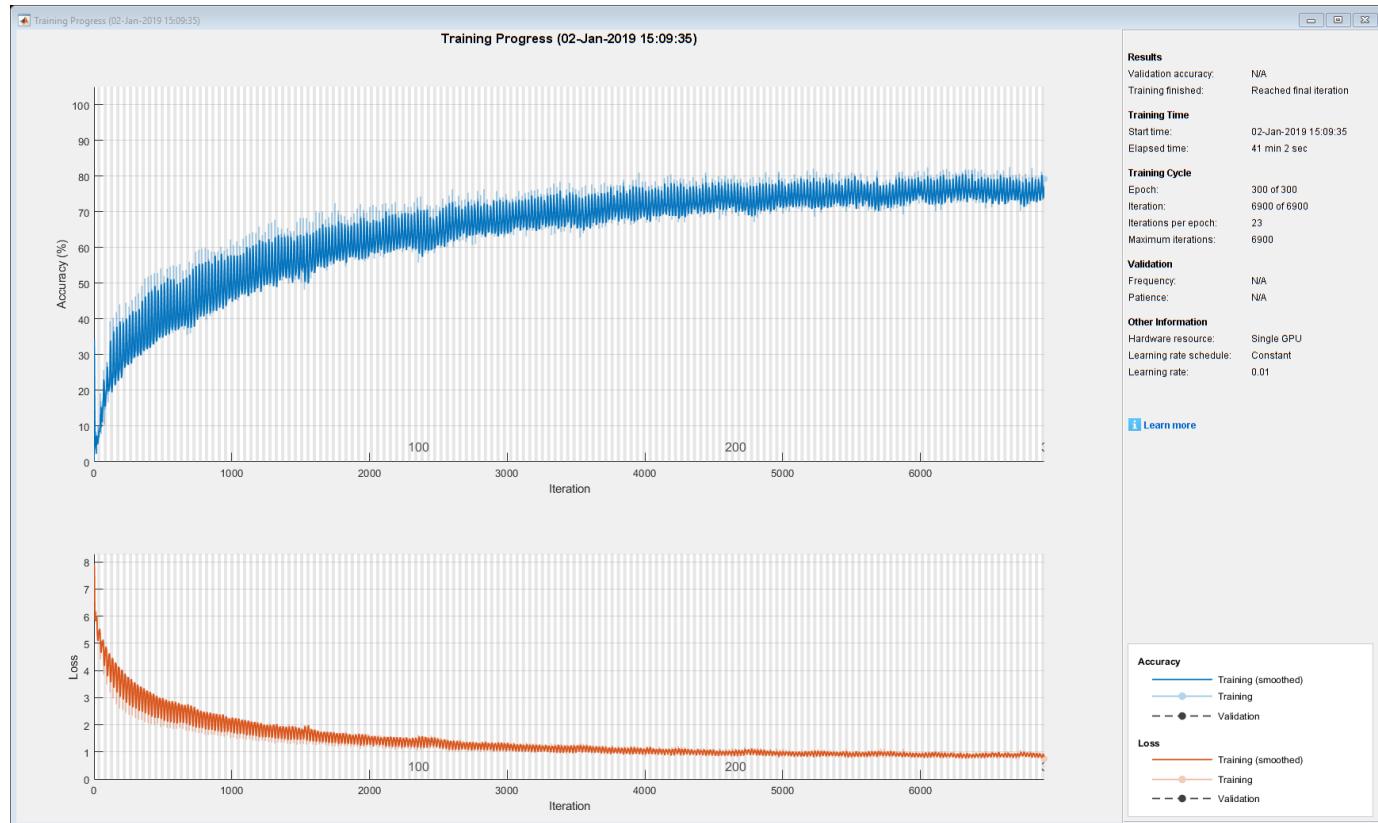
layers = [
    sequenceInputLayer(inputSize)
    wordEmbeddingLayer(embeddingDimension,numWords)
    lstmLayer(100)
    dropoutLayer(0.2)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

指定训练选项。指定求解器为 'adam'。进行 300 轮训练，学习率为 0.01。将小批量大小设置为 32。要保持数据按序列长度排序，请将 'Shuffle' 选项设置为 'never'。要监控训练进度，请将 'Plots' 选项设置为 'training-progress'。要隐藏详细输出，请将 'Verbose' 设置为 `false`。

```
options = trainingOptions('adam',...
    'MaxEpochs',300, ...
    'InitialLearnRate',0.01, ...
    'MiniBatchSize',32, ...
    'Shuffle','never', ...
    'Plots','training-progress',...
    'Verbose',false);
```

使用 `trainNetwork` 训练网络。

```
net = trainNetwork(ds,layers,options);
```



## 生成新文本

根据训练数据中文本的所有首个单词的概率分布抽取一个单词来生成文本的第一个单词。接着使用经过训练的 LSTM 网络基于当前已生成的文本序列预测下一时间步，以生成其余单词。继续逐个生成单词，直到网络预测到“文本结尾”单词。

要使用网络进行第一次预测，请输入表示“文本开始”标记的索引。使用 `word2ind` 函数和文档数据存储所使用的单词编码来查找索引。

```
enc = ds.Encoding;
wordIndex = word2ind(enc,"startOfText")
```

```
wordIndex = 1
```

在后续的预测中，会根据网络的预测分数来抽取下一个单词。预测分数表示下一个单词的概率分布。从网络输出层的类名给出的词汇表中抽取单词。

```
vocabulary = string(net.Layers(end).Classes);
```

使用 `predictAndUpdateState` 逐单词进行预测。对于每次预测，都输入前一个单词的索引。当网络预测到文本结尾单词或生成的文本长度达到 500 个字符时，停止预测。对于大型数据集合、长序列或大型网络，在 GPU 上进行预测计算通常比在 CPU 上快。其他情况下，在 CPU 上进行预测计算通常更快。对于单时间步预测，请使用 CPU。要使用 CPU 进行预测，请将 `predictAndUpdateState` 的 `'ExecutionEnvironment'` 选项设置为 `'cpu'`。

```
generatedText = "";
maxLength = 500;
while strlength(generatedText) < maxLength
```

```
% Predict the next word scores.
[net,wordScores] = predictAndUpdateState(net,wordIndex,'ExecutionEnvironment','cpu');

% Sample the next word.
newWord = datasample(vocabulary,1,'Weights',wordScores);

% Stop predicting at the end of text.
if newWord == "EndOfText"
    break
end

% Add the word to the generated text.
generatedText = generatedText + " " + newWord;

% Find the word index for the next input.
wordIndex = word2ind(enc,newWord);
end
```

生成过程在每个预测之间引入空白字符，这意味着一些标点字符前后会出现不必要的空格。通过删除相应标点字符前后的空格来重新构造生成的文本。

删除特定标点字符前的空格。

```
punctuationCharacters = [".,"","?","!"];
generatedText = replace(generatedText," " + punctuationCharacters,punctuationCharacters);
```

删除特定标点字符后的空格。

```
punctuationCharacters = ["(" ")"];
generatedText = replace(generatedText,punctuationCharacters + " ",punctuationCharacters)

generatedText =
"‘Sure, it's a good Turtle!’ said the Queen in a low, weak voice."
```

要生成多篇文本，请在每次生成完成后使用 `resetState` 重置网络状态。

```
net = resetState(net);
```

## 另请参阅

[wordEmbeddingLayer](#) | [doc2sequence](#) | [tokenizedDocument](#) | [lstmLayer](#) | [trainNetwork](#) | [trainingOptions](#) | [sequenceInputLayer](#) | [wordcloud](#) | [extractHTMLText](#) | [findElement](#) | [htmlTree](#)

## 相关示例

- “使用深度学习生成文本”（第 4-72 页）
- “Create Simple Text Model for Classification”（Text Analytics Toolbox）
- “Analyze Text Data Using Topic Models”（Text Analytics Toolbox）
- “Analyze Text Data Using Multiword Phrases”（Text Analytics Toolbox）
- “Train a Sentiment Classifier”（Text Analytics Toolbox）
- “使用深度学习进行序列分类”（第 4-2 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）



# 深度学习调整和可视化

---

- “使用 GoogLeNet 的 Deep Dream 图像” (第 5-2 页)
- “使用贝叶斯优化进行深度学习” (第 5-8 页)
- “并行训练深度学习网络” (第 5-17 页)
- “监控深度学习训练进度” (第 5-22 页)
- “自定义深度学习网络训练期间的输出” (第 5-26 页)
- “使用类激活映射调查网络预测” (第 5-30 页)
- “可视化卷积神经网络的激活区域” (第 5-35 页)
- “可视化卷积神经网络的特征” (第 5-46 页)

## 使用 GoogLeNet 的 Deep Dream 图像

此示例说明如何使用 `deepDreamImage` 和预训练卷积神经网络 GoogLeNet 生成图像。

Deep Dream 是深度学习中的一种特征可视化方法，它会合成可强烈激活网络层的图像。通过可视化这些图像，您可以突出显示通过网络学习到的图像特征。这些图像有助于了解和诊断网络行为。

您可以通过可视化接近网络末尾的层的特征来生成有趣的图像。

该示例使用 Deep Learning Toolbox™ 和 Deep Learning Toolbox Model for GoogLeNet Network 生成图像。

### 加载预训练网络

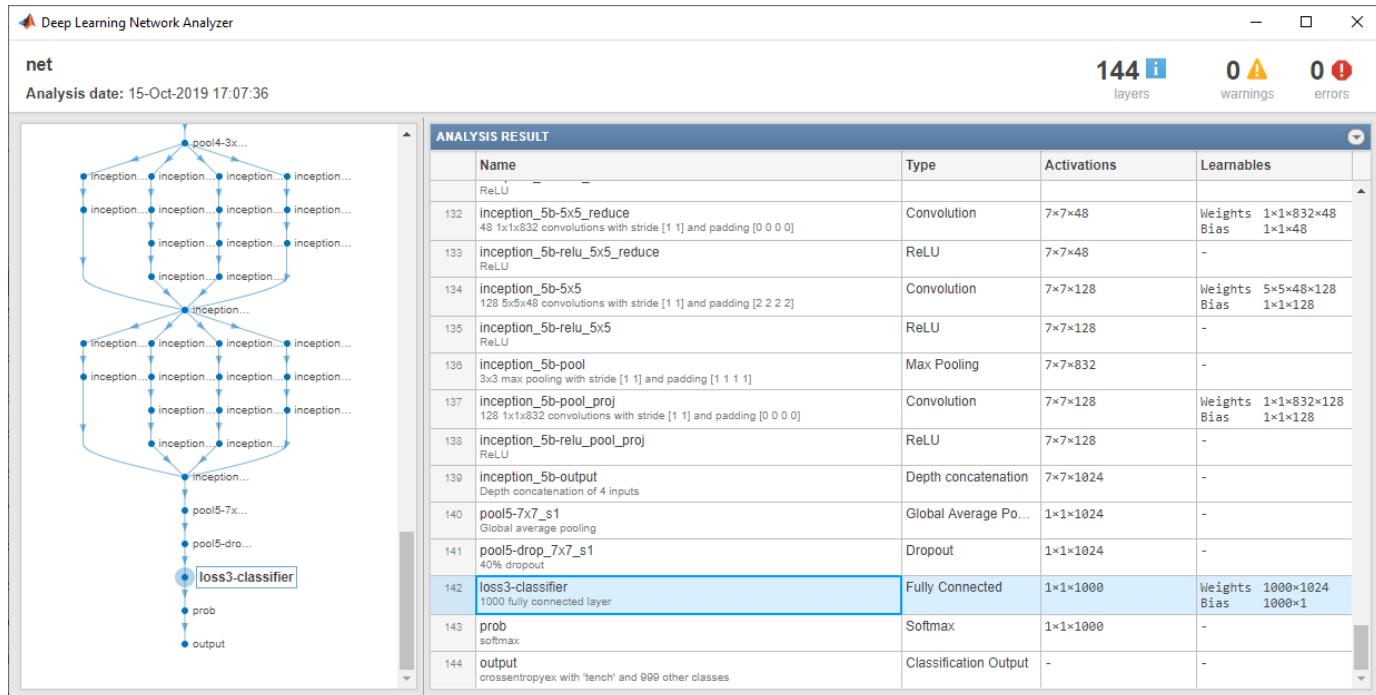
加载预训练的 GoogLeNet 网络。如果未安装 Deep Learning Toolbox Model for GoogLeNet Network 支持包，则软件会提供下载链接。

```
net = googlenet;
```

### 生成图像

要生成与给定类最相似的图像，请选择全连接层。首先，使用 `analyzeNetwork` 查看网络架构，找到此层的层索引。

```
analyzeNetwork(net)
```



然后选择全连接层，在此示例中为 142。

```
layer = 142;
layerName = net.Layers(layer).Name
```

```
layerName =
'loss3-classifier'
```

通过选择多个类，您可以一次生成多个图像。通过将 **channels** 设置为这些类名称的索引，选择要可视化的类。

```
channels = [114 293 341 484 563 950];
```

这些类存储在输出层（最后一层）的 **Classes** 属性中。您可以通过选择 **channels** 中的条目来查看所选类的名称。

```
net.Layers(end).Classes(channels)
```

```
ans = 6×1 categorical
  snail
  tiger
  zebra
  castle
  fountain
  strawberry
```

使用 **deepDreamImage** 生成图像。如果存在兼容的 GPU，此命令会使用 GPU。否则将使用 CPU。使用 GPU 需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。

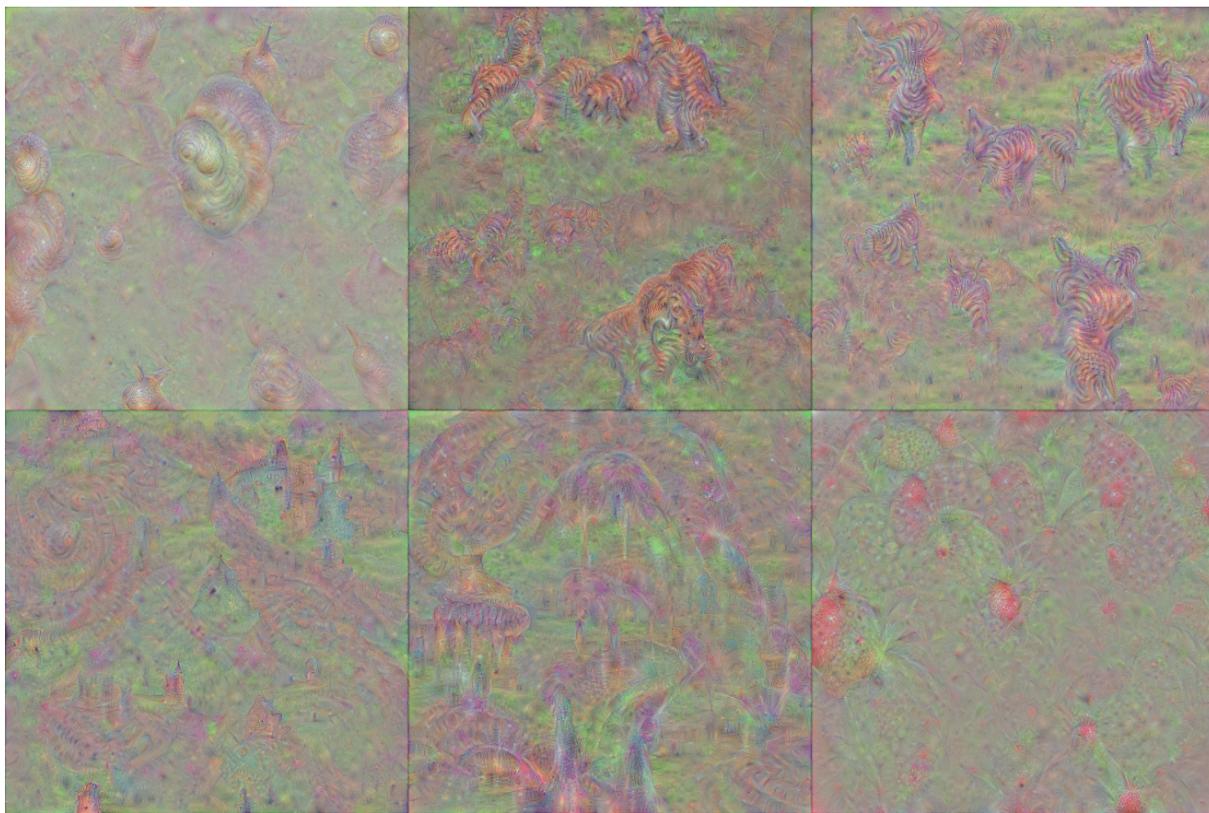
```
I = deepDreamImage(net,layerName,channels);
```

Iteration	Activation	Pyramid Level
	Strength	
1	0.09	1
2	0.67	1
3	4.86	1
4	8.41	1
5	11.27	1
6	14.86	1
7	17.39	1
8	22.84	1
9	27.78	1
10	34.39	1
1	3.99	2
2	11.51	2
3	13.82	2
4	19.87	2
5	20.67	2
6	20.82	2
7	24.01	2
8	27.20	2
9	28.24	2
10	35.93	2
1	34.91	3
2	46.18	3
3	41.03	3
4	48.84	3
5	51.13	3
6	58.65	3

7	58.12	3
8	61.68	3
9	71.53	3
10	76.01	3

使用 `imtile` 一起显示所有图像。

```
figure
I = imtile(I);
imshow(I)
```



### 生成更详细的图像

增加金字塔等级数和每个金字塔等级的迭代次数可以生成更详细的图像，但代价是额外计算。

您可以使用 '`NumIterations`' 选项增加迭代次数。将迭代次数设置为 100。

```
iterations = 100;
```

生成强烈激活 'tiger' 类（通道 293）的详细图像。将 '`Verbose`' 设置为 `false` 以隐藏优化过程的详细信息。

```
channels = 293;
I = deepDreamImage(net,layerName,channels, ...
    'Verbose',false, ...
```

```
'NumIterations',iterations);
```

```
figure  
imshow(I)
```



要生成更大、更详细的输出图像，您可以增加金字塔层级的数量和每个金字塔层级的迭代次数。

将金字塔层级数设置为 4。

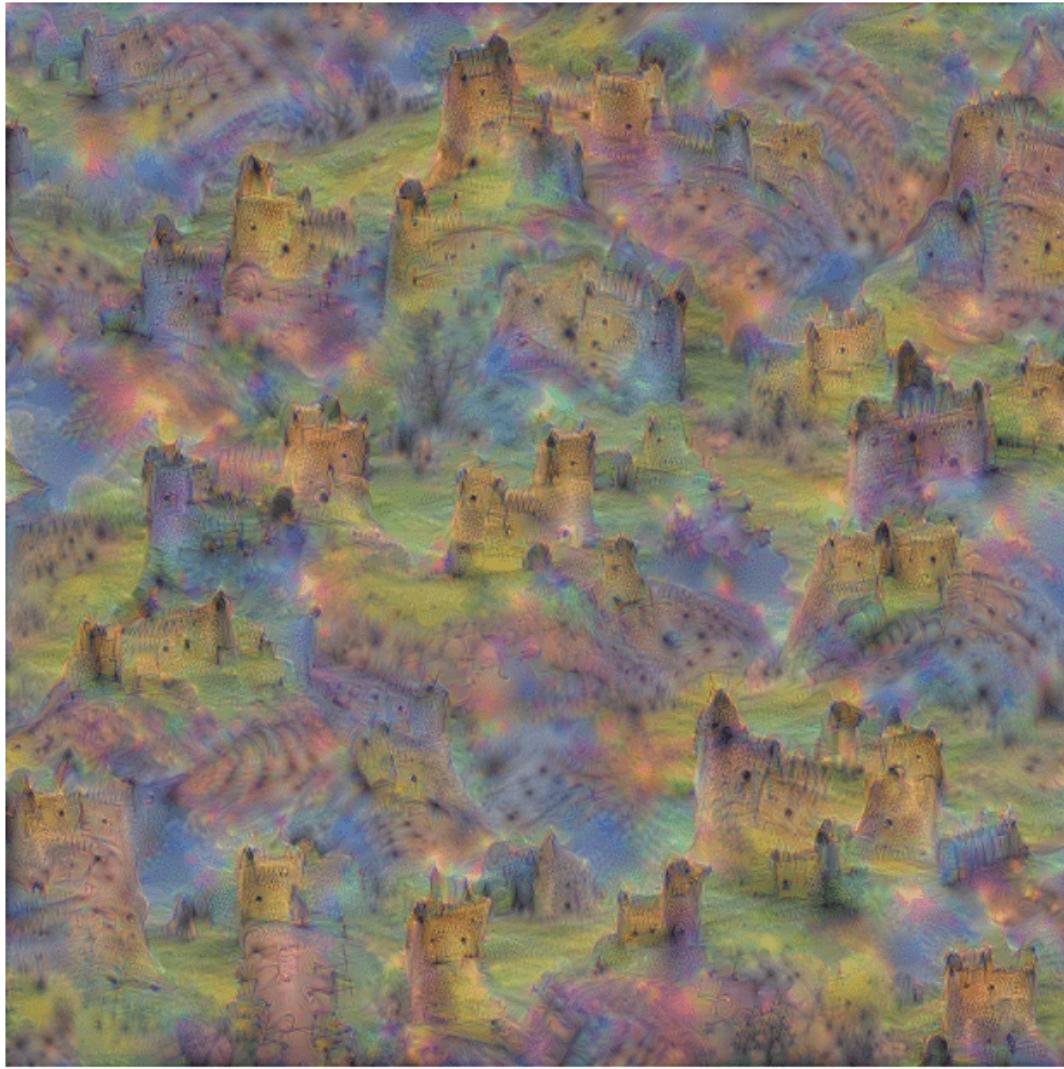
```
levels = 4;
```

生成一个详细图像，该图像强烈激活 'castle' 类（通道 484）。

```
channels = 484;
```

```
I = deepDreamImage(net,layerName,channels,...  
'Verbose',false,...  
'NumIterations',iterations,...  
'PyramidLevels',levels);
```

```
figure  
imshow(I)
```



### 另请参阅

[googlenet](#) | [deepDreamImage](#) | [occlusionSensitivity](#) | [imageLIME](#) | [gradCAM](#)

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “预训练的深度神经网络” (第 1-8 页)
- “可视化卷积神经网络的激活区域” (第 5-35 页)

- “可视化卷积神经网络的特征” (第 5-46 页)
- “Grad-CAM Reveals the Why Behind Deep Learning Decisions”

## 使用贝叶斯优化进行深度学习

此示例说明如何将贝叶斯优化应用于深度学习并找到卷积神经网络的最优网络超参数和训练选项。

要训练深度神经网络，您必须指定神经网络架构以及训练算法的选项。选择和调整这些超参数可能很困难并且需要时间。贝叶斯优化是一种非常适合对分类模型和回归模型的超参数进行优化的算法。您可以使用贝叶斯优化来对不可微分、不连续且计算耗时的函数进行优化。该算法在内部维护目标函数的高斯过程模型，并使用目标函数计算来训练此模型。

以下示例演示如何：

- 下载并准备 CIFAR-10 数据集以进行网络训练。此数据集是一个广泛用于测试图像分类模型的数据集。
- 指定要使用贝叶斯优化进行优化的变量。这些变量是训练算法的选项，以及网络架构本身的参数。
- 定义目标函数，该函数将优化变量的值作为输入，指定网络架构和训练选项，训练和验证网络，并将经过训练的网络保存到磁盘。目标函数在此脚本的末尾定义。
- 通过最大程度地减小对验证集的分类误差来执行贝叶斯优化。
- 从磁盘加载最佳网络并基于测试集对其进行评估。

您也可以使用贝叶斯优化在 Experiment Manager 中找到最佳训练选项。有关详细信息，请参阅“Tune Experiment Hyperparameters by Using Bayesian Optimization”。

### 准备数据

下载 CIFAR-10 数据集 [1]。该数据集包含 60,000 个图像，每个图像的大小为  $32 \times 32$ ，并具有三个颜色通道 (RGB)。整个数据集的大小为 175 MB。下载过程可能需要一些时间，具体取决于您的 Internet 连接情况。

```
datadir = tempdir;
downloadCIFARData(datadir);
```

加载 CIFAR-10 数据集，用作训练图像和标签以及测试图像和标签。要启用网络验证，请将 5000 个测试图像用于验证。

```
[XTrain,YTrain,XTest,YTest] = loadCIFARData(datadir);

idx = randperm(numel(YTest),5000);
XValidation = XTest(:, :, :, idx);
XTest(:, :, :, idx) = [];
YValidation = YTest(idx);
YTest(idx) = [];
```

您可以使用以下代码显示训练图像的样本。

```
figure;
idx = randperm(numel(YTrain),20);
for i = 1:numel(idx)
    subplot(4,5,i);
    imshow(XTrain(:, :, :, idx(i)));
end
```

### 选择要优化的变量

选择要使用贝叶斯优化对哪些变量进行优化，并指定搜索范围。此外，指定变量是否为整数以及是否搜索对数空间中的区间。优化以下变量：

- 网络部分深度。此参数控制网络的深度。网络有三个部分，每个部分都有 **SectionDepth** 个相同的卷积层。因此卷积层的总数为  $3 * \text{SectionDepth}$ 。脚本稍后部分中的目标函数在每个层中采用与  $1/\sqrt{\text{SectionDepth}}$  成比例的卷积滤波器数量。因此，对于不同的部分深度，每次迭代的参数数量和所需的计算量大致相同。
- 初始化学习率。最佳学习率取决于您的数据以及您正在训练的网络。
- 随机梯度下降动量。动量在当前参数更新中包含一个与前一次迭代中的更新成比例的贡献，从而在更新中增加惯性。这会使参数更新更加平滑，并减少随机梯度下降固有的噪声。
- L2 正则化强度。使用正则化以防止过拟合。搜索正则化强度的空间以查找良好的值。数据增强和批量归一化也有助于正则化网络。

```
optimVars = [
    optimizableVariable('SectionDepth',[1 3],'Type','integer')
    optimizableVariable('InitialLearnRate',[1e-2 1],'Transform','log')
    optimizableVariable('Momentum',[0.8 0.98])
    optimizableVariable('L2Regularization',[1e-10 1e-2],'Transform','log')];
```

## 执行贝叶斯优化

使用训练数据和验证数据作为输入，为贝叶斯优化器创建目标函数。目标函数训练一个卷积神经网络并返回对验证集的分类误差。该函数在此脚本的末尾定义。由于 **bayesopt** 使用基于验证集的误差率来选择最佳模型，因此最终网络可能会对验证集过拟合。随后基于独立测试集测试最终选择的模型，以估计泛化误差。

```
ObjFcn = makeObjFcn(XTrain,YTrain,XValidation,YValidation);
```

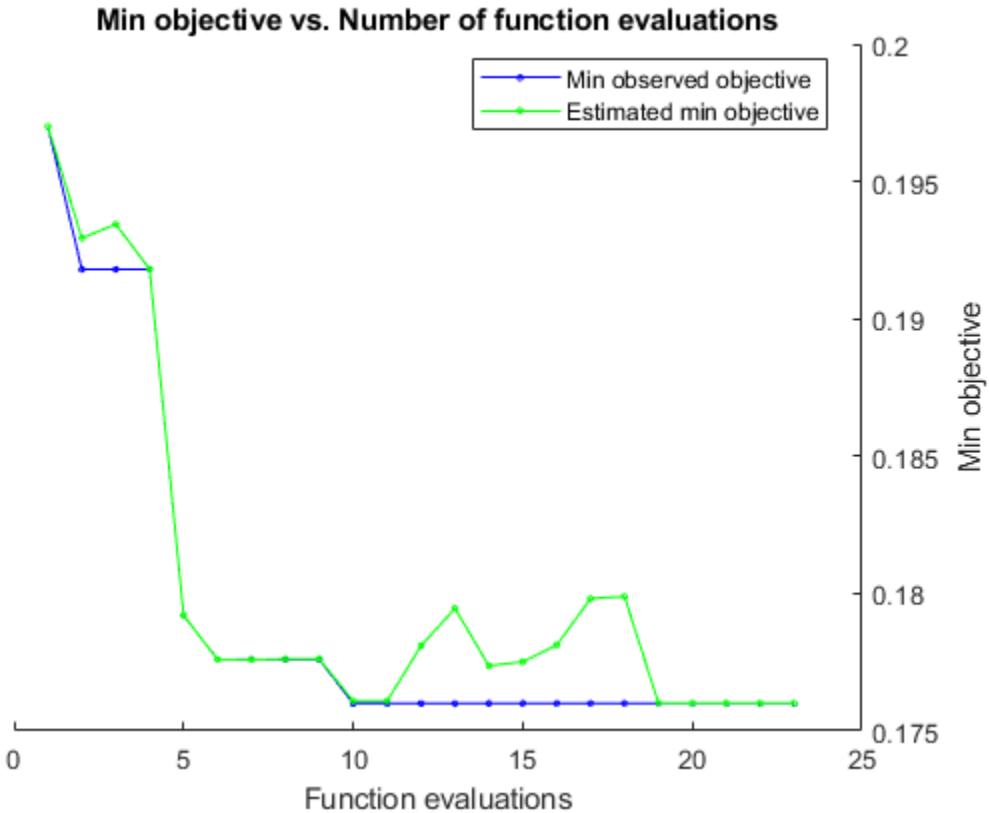
通过最大程度地减小对验证集的分类误差来执行贝叶斯优化。以秒为单位指定总优化时间。为了最好地利用贝叶斯优化的能力，您应该执行至少 30 次对象函数评估。要在多个 GPU 上并行训练网络，请将 '**UseParallel**' 值设置为 **true**。如果您有一个 GPU 并将 '**UseParallel**' 值设置为 **true**，则所有工作进程将共用该 GPU，这不仅无法加速训练，而且会增加 GPU 耗尽内存的几率。

每个网络完成训练后，**bayesopt** 将结果显示到命令行窗口。然后，**bayesopt** 函数返回 **BayesObject.UserDataTrace** 中的文件名。目标函数将经过训练的网络保存到磁盘，并将文件名返回到 **bayesopt**。

```
BayesObject = bayesopt(ObjFcn,optimVars, ...
    'MaxTime',14*60*60, ...
    'IsObjectiveDeterministic',false, ...
    'UseParallel',false);
```

Iter	Eval	Objective	Objective	BestSoFar	BestSoFar	SectionDepth	InitialLearn-	Momentu
	result	runtime	(observed)	(estim.)	Rate		tion	
1	Best	0.197	955.69	0.197	0.197	3	0.61856	0.80624
2	Best	0.1918	790.38	0.1918	0.19293	2	0.074118	0.91031
3	Accept	0.2438	660.29	0.1918	0.19344	1	0.051153	0.90911
4	Accept	0.208	672.81	0.1918	0.1918	1	0.70138	0.81923
5	Best	0.1792	844.07	0.1792	0.17921	2	0.65156	0.93783
6	Best	0.1776	851.49	0.1776	0.17759	2	0.23619	0.91932
7	Accept	0.2232	883.5	0.1776	0.17759	2	0.011147	0.91526

8   Accept   0.2508   822.65   0.1776   0.17762   1   0.023919   0.91048   1.0002e-10
9   Accept   0.1974   1947.6   0.1776   0.17761   3   0.010017   0.97683   5.4603e-10
10   Best   0.176   1938.4   0.176   0.17608   2   0.3526   0.82381   1.4244e-07
11   Accept   0.1914   2874.4   0.176   0.17608   3   0.079847   0.86801   9.7335e-07
12   Accept   0.181   2578   0.176   0.17809   2   0.35141   0.80202   4.5634e-08
13   Accept   0.1838   2410.8   0.176   0.17946   2   0.39508   0.95968   9.3856e-06
14   Accept   0.1786   2490.6   0.176   0.17737   2   0.44857   0.91827   1.0939e-10
15   Accept   0.1776   2668   0.176   0.17751   2   0.95793   0.85503   1.0222e-05
16   Accept   0.1824   3059.8   0.176   0.17812   2   0.41142   0.86931   1.447e-06
17   Accept   0.1894   3091.5   0.176   0.17982   2   0.97051   0.80284   1.5836e-10
18   Accept   0.217   2794.5   0.176   0.17989   1   0.2464   0.84428   4.4938e-06
19   Accept   0.2358   4054.2   0.176   0.17601   3   0.22843   0.9454   0.00098248
20   Accept   0.2216   4411.7   0.176   0.17601   3   0.010847   0.82288   2.4756e-08
=====
Iter   Eval   Objective   Objective   BestSoFar   BestSoFar   SectionDepth   InitialLearn-   Momentum
result   runtime   (observed)   (estim.)   Rate     tion
21   Accept   0.2038   3906.4   0.176   0.17601   2   0.09885   0.81541   0.0021184
22   Accept   0.2492   4103.4   0.176   0.17601   2   0.52313   0.83139   0.0016269
23   Accept   0.1814   4240.5   0.176   0.17601   2   0.29506   0.84061   6.0203e-10




---

Optimization completed.  
 MaxTime of 50400 seconds reached.  
 Total function evaluations: 23  
 Total elapsed time: 53088.5123 seconds  
 Total objective function evaluation time: 53050.7026

Best observed feasible point:

SectionDepth	InitialLearnRate	Momentum	L2Regularization
2	0.3526	0.82381	1.4244e-07

Observed objective function value = 0.176  
 Estimated objective function value = 0.17601  
 Function evaluation time = 1938.4483

Best estimated feasible point (according to models):

SectionDepth	InitialLearnRate	Momentum	L2Regularization
2	0.3526	0.82381	1.4244e-07

Estimated objective function value = 0.17601  
 Estimated function evaluation time = 1898.2641

### 评估最终网络

加载在优化中找到的最佳网络及其验证准确度。

```
bestIdx = BayesObject.IndexOfMinimumTrace(end);
fileName = BayesObject.UserDataTrace{bestIdx};
savedStruct = load(fileName);
valError = savedStruct.valError
```

```
valError = 0.1760
```

预测测试集的标签并计算测试误差。将测试集中每个图像的分类视为具有一定成功概率的独立事件，这意味着分类错误的图像数量遵循二项分布。使用这种方法来计算标准误差 (testErrorSE) 和泛化误差率的约 95% 置信区间 (testError95CI)。此方法通常称为 Wald 方法。`bayesopt` 使用验证集确定最佳网络，而不对网络使用测试集。因此，测试误差可能高于验证误差。

```
[YPredicted,probs] = classify(savedStruct.trainedNet,XTest);
testError = 1 - mean(YPredicted == YTest)
```

```
testError = 0.1910
```

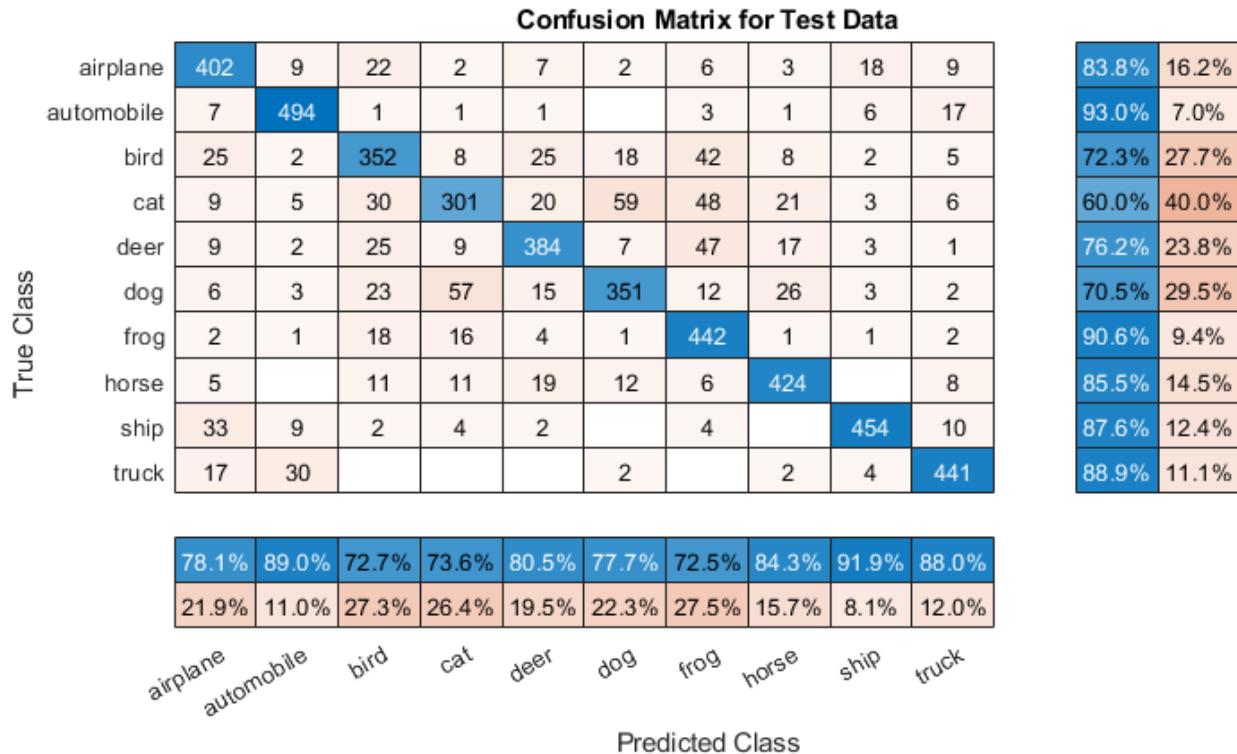
```
NTest = numel(YTest);
testErrorSE = sqrt(testError*(1-testError)/NTest);
testError95CI = [testError - 1.96*testErrorSE, testError + 1.96*testErrorSE]
```

```
testError95CI = 1×2
```

```
0.1801 0.2019
```

绘制测试数据的混淆矩阵。使用列汇总和行汇总显示每个类的准确率和召回率。

```
figure('Units','normalized','Position',[0.2 0.2 0.4 0.4]);
cm = confusionchart(YTest,YPredicted);
cm.Title = 'Confusion Matrix for Test Data';
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
```



您可以使用以下代码显示一些测试图像以及它们的预测类和这些类的概率。

```
figure
idx = randperm(numel(YTest),9);
for i = 1:numel(idx)
    subplot(3,3,i)
    imshow(XTest(:,:,idx(i)));
    prob = num2str(100*max(probs(idx(i,:)),3));
    predClass = char(YPredicted(idx(i)));
    label = [predClass,' ',prob,'%'];
    title(label)
end
```

## 优化的目标函数

定义优化的目标函数。此函数执行以下步骤：

- 1 将优化变量的值作为输入。**bayesopt** 使用某个表中的优化变量的当前值调用目标函数，该表的每个列名等于变量名称。例如，网络部分深度的当前值为 **optVars.SectionDepth**。
- 2 定义网络架构和训练选项。
- 3 训练并验证网络。
- 4 将经过训练的网络、验证误差和训练选项保存到磁盘。
- 5 返回已保存网络的验证误差和文件名。

```
function ObjFcn = makeObjFcn(XTrain,YTrain,XValidation,YValidation)
ObjFcn = @valErrorFun;
    function [valError,cons,fileNames] = valErrorFun(optVars)
```

定义卷积神经网络架构。

- 对卷积层进行填充，使空间输出大小始终与输入大小相同。
- 每次使用最大池化层将空间维度下采样一半时，将滤波器数量增加一倍。这样做可确保每个卷积层所需的计算量大致相同。
- 选择与  $1/\sqrt{\text{SectionDepth}}$  成比例的滤波器数量，使不同深度的网络具有大致相同的参数个数，并且每次迭代所需的计算量大致相同。要增加网络参数的个数和网络整体灵活性，请增大 numF 值。要训练更深的网络，请更改 SectionDepth 变量的范围。
- 使用 convBlock(filterSize,numFilters,numConvLayers) 创建 numConvLayers 卷积层块，其中每个层都有指定的 filterSize 和 numFilters 滤波器，并且每个层都后接一个批量归一化层和一个 ReLU 层。convBlock 函数在此示例的末尾定义。

```

imageSize = [32 32 3];
numClasses = numel(unique(YTrain));
numF = round(16/sqrt(optVars.SectionDepth));
layers = [
    imageInputLayer(imageSize)

    % The spatial input and output sizes of these convolutional
    % layers are 32-by-32, and the following max pooling layer
    % reduces this to 16-by-16.
    convBlock(3,numF,optVars.SectionDepth)
    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    % The spatial input and output sizes of these convolutional
    % layers are 16-by-16, and the following max pooling layer
    % reduces this to 8-by-8.
    convBlock(3,2*numF,optVars.SectionDepth)
    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    % The spatial input and output sizes of these convolutional
    % layers are 8-by-8. The global average pooling layer averages
    % over the 8-by-8 inputs, giving an output of size
    % 1-by-1-by-4*initialNumFilters. With a global average
    % pooling layer, the final classification output is only
    % sensitive to the total amount of each feature present in the
    % input image, but insensitive to the spatial positions of the
    % features.
    convBlock(3,4*numF,optVars.SectionDepth)
    averagePooling2dLayer(8)

    % Add the fully connected layer and the final softmax and
    % classification layers.
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

```

指定网络训练的选项。优化初始学习率、SGD 动量和 L2 正则化强度。

指定验证数据并选择 'ValidationFrequency' 值，使 trainNetwork 每轮训练都验证一次网络。进行固定轮数的训练，并在最后几轮将学习率降低十分之一。这会减少参数更新的噪声，并使网络参数稳定下来，更接近于损失函数的最小值。

```

miniBatchSize = 256;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'InitialLearnRate',optVars.InitialLearnRate, ...
    'Momentum',optVars.Momentum, ...

```

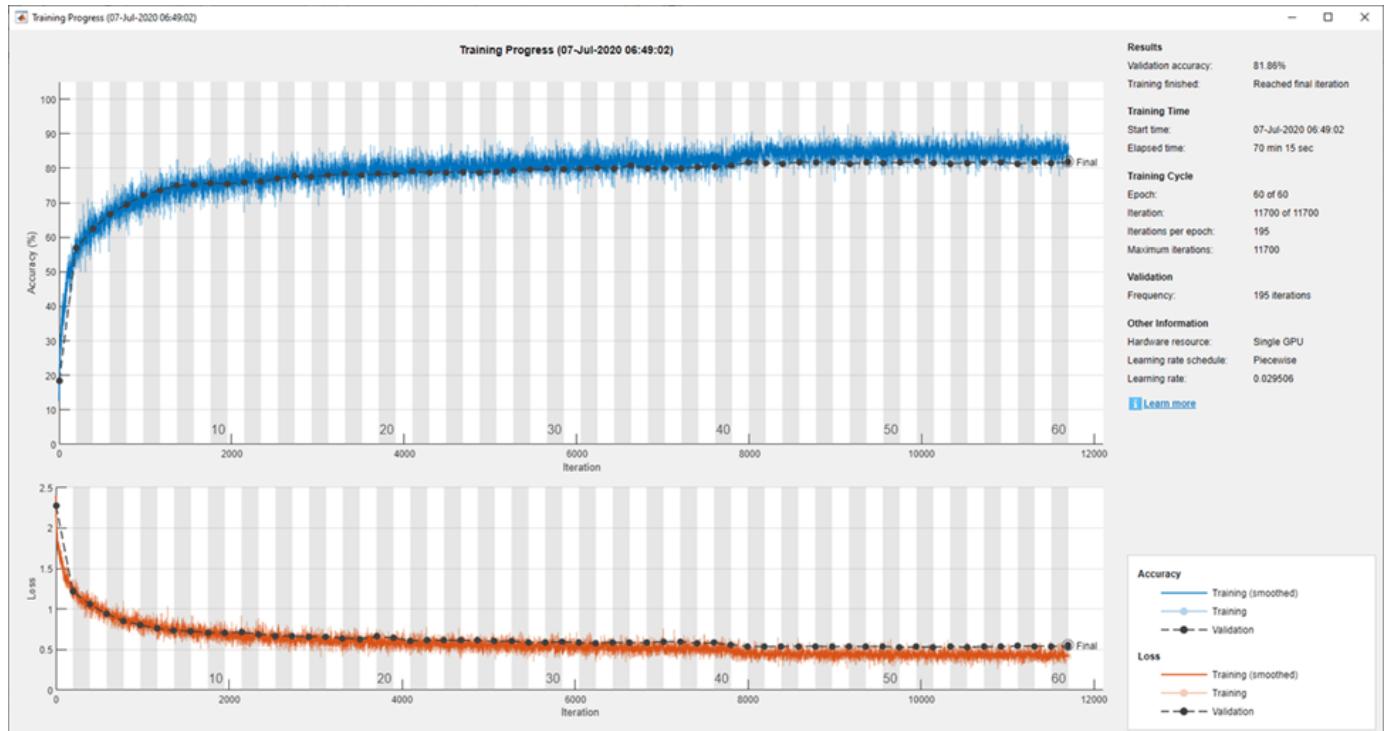
```
'MaxEpochs',60, ...
'LearnRateSchedule','piecewise', ...
'LearnRateDropPeriod',40, ...
'LearnRateDropFactor',0.1, ...
'MiniBatchSize',miniBatchSize, ...
'L2Regularization',optVars.L2Regularization, ...
'Shuffle','every-epoch', ...
'Verbose',false, ...
'Plots','training-progress', ...
'ValidationData',{XValidation,YValidation}, ...
'ValidationFrequency',validationFrequency);
```

使用数据增强沿垂直轴随机翻转训练图像，并在水平方向和垂直方向上将图像随机平移最多四个像素。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
datasource = augmentedImageDatastore(imageSize,XTrain,YTrain,'DataAugmentation',imageAugmenter);
```

训练网络并在训练过程中绘制训练进度。训练结束后关闭所有训练图。

```
trainedNet = trainNetwork(datasource,layers,options);
close(findall(gcf,'Tag','NNET_CNN_TRAININGPLOT_UFIGURE'))
```



基于验证集评估经过训练的网络，计算预测的图像标签，并计算基于验证数据的误差率。

```
YPredicted = classify(trainedNet,XValidation);
valError = 1 - mean(YPredicted == YValidation);
```

创建包含验证误差的文件名，并将网络、验证误差和训练选项保存到磁盘。目标函数返回 `fileName` 作为输出参数，`bayesopt` 返回 `BayesObject.UserDataTrace` 中的所有文件名。额外的必需输出参数 `cons` 指定变量之间的约束。没有变量约束。

```
fileName = num2str(valError) + ".mat";
save(fileName,'trainedNet','valError','options')
cons = [];

end
end
```

`convBlock` 函数创建 `numConvLayers` 卷积层块，其中每个层都有指定的 `filterSize` 和 `numFilters` 滤波器，并且每个层都后接一个批量归一化层和一个 `ReLU` 层。

```
function layers = convBlock(filterSize,numFilters,numConvLayers)
layers = [
    convolution2dLayer(filterSize,numFilters,'Padding','same')
    batchNormalizationLayer
    reluLayer];
layers = repmat(layers,numConvLayers,1);
end
```

## 参考

[1] Krizhevsky, Alex. "Learning multiple layers of features from tiny images." (2009). <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>

## 另请参阅

[试验管理器](#) | [trainNetwork](#) | [trainingOptions](#) | [bayesopt](#)

## 相关示例

- “了解卷积神经网络” (第 1-16 页)
- “指定卷积神经网络的层” (第 1-25 页)
- “设置参数并训练卷积神经网络” (第 1-35 页)
- “预训练的深度神经网络” (第 1-8 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “比较层权重初始化函数” (第 17-22 页)
- “指定自定义权重初始化函数” (第 17-16 页)
- “Tune Experiment Hyperparameters by Using Bayesian Optimization”

# 并行训练深度学习网络

此示例说明如何在本地计算机上运行多个深度学习试验。使用此示例作为模板，您可以修改网络层和训练选项，以满足您的具体应用需要。无论您有一个还是多个 GPU，都可以使用这种方法。如果您只有一个 GPU，网络会在后台逐个进行训练。本示例中的方法使您能够在进行深度学习试验时继续使用 MATLAB®。

您也可以使用 Experiment Manager 以交互方式并行训练多个深度网络。有关详细信息，请参阅 “Use Experiment Manager to Train Networks in Parallel”。

## 准备数据集

在运行该示例之前，您必须能够访问深度学习数据集的本地副本。此示例使用的数据集包含从 0 到 9 的数字的合成图像。在以下代码中，将位置更改为指向您的数据集。

```
datasetLocation = fullfile(matlabroot,'toolbox','nnet',...
    'nnndemos','nndatasets','DigitDataset');
```

如果您要用更多资源运行试验，可以在云中的群集中运行此示例。

- 将数据集上传到 Amazon S3 存储桶中。有关示例，请参阅。
- 创建一个云群集。在 MATLAB 中，您可以直接通过 MATLAB 桌面在云中创建群集。有关详细信息，请参阅 “Create Cloud Cluster” (Parallel Computing Toolbox)。
- 选择您的云群集作为默认群集，在主页选项卡的环境部分中，选择 **Parallel > Select a Default Cluster**。

## 加载数据集

使用 `imageDatastore` 对象加载数据集。将数据集分成训练集、验证集和测试集。

```
imds = imageDatastore(datasetLocation, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

[imdsTrain,imdsValidation,imdsTest] = splitEachLabel(imds,0.8,0.1);
```

要使用增强的图像数据训练网络，请创建 `augmentedImageDatastore` 对象。使用随机平移和水平翻转。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
imageSize = [28 28 1];
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augmentedImdsTrain = augmentedImageDatastore(imageSize,imdsTrain, ...
    'DataAugmentation',imageAugmenter);
```

## 并行训练网络

启动一个工作进程数量与 GPU 数量一样多的并行池。您可以使用 `gpuDeviceCount` (Parallel Computing Toolbox) 函数检查可用 GPU 的数量。MATLAB 为每个工作进程分配一个不同的 GPU。默认情况下，`parpool` 使用您的默认群集配置文件。如果您没有更改默认值，则该配置文件为 `local`。此示例是用一台具有 2 个 GPU 的计算机运行的。

```
numGPUs = gpuDeviceCount("available");
parpool(numGPUs);
```

Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 2).

要在训练期间从工作进程发送训练进度信息，请使用 **parallel.pool.DataQueue** (Parallel Computing Toolbox) 对象。要了解有关如何在训练期间使用数据队列获取反馈的详细信息，请参阅示例。

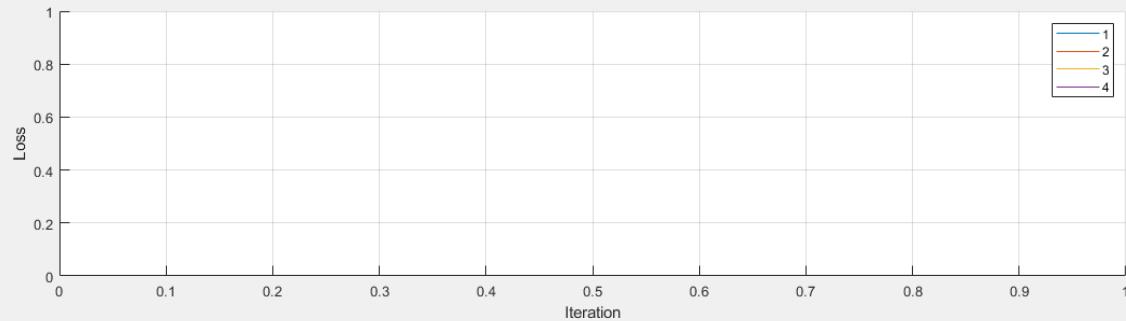
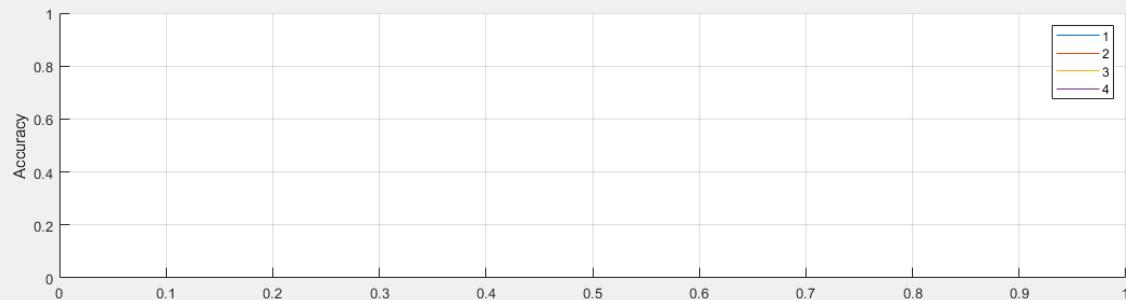
```
dataqueue = parallel.pool.DataQueue;
```

定义网络层和训练选项。为了提高代码可读性，您可以在一个单独的函数中定义它们，该函数返回多个网络架构和训练选项。在本例中，**networkLayersAndOptions** 返回一个网络层元胞数组和一个训练选项数组，二者长度相同。在 MATLAB 中打开此示例，然后点击 **networkLayersAndOptions** 以打开支持函数 **networkLayersAndOptions**。粘贴到您自己的网络层和选项中。该文件包含示例训练选项，说明如何使用输出函数向数据队列发送信息。

```
[layersCell,options] = networkLayersAndOptions(augmentedImdsTrain,imdsValidation,dataqueue);
```

准备训练进度图，并设置回调函数以便在每个工作进程向队列发送数据后更新这些图。**preparePlots** 和 **updatePlots** 是此示例的支持函数。

```
handles = preparePlots(numel(layersCell));
```



```
afterEach(dataqueue,@(data) updatePlots(handles,data));
```

要在并行工作进程中保存计算结果，请使用 **future** 对象。为每次训练的结果预分配一个 **future** 对象数组。

```
trainingFuture(1:numel(layersCell)) = parallel.FevalFuture;
```

使用 **for** 循环遍历网络层和选项，并使用 **parfeval** (Parallel Computing Toolbox) 在并行工作进程上训练网络。要从 **trainNetwork** 请求两个输出参数，请指定 2 作为 **parfeval** 的第二个输入参数。

```
for i=1:numel(layersCell)
    trainingFuture(i) = parfeval(@trainNetwork,2,augmentedImdsTrain,layersCell{i},options(i));
end
```

**parfeval** 不会阻止 MATLAB，因此您可以在计算的同时继续工作。

要从 **future** 对象中获取结果，请使用 **fetchOutputs** 函数。对于本示例，获取经过训练的网络及其训练信息。**fetchOutputs** 会阻止 MATLAB，直到结果可用为止。此步骤可能需要几分钟。

```
[network,trainingInfo] = fetchOutputs(trainingFuture);
```



使用 **save** 函数将结果保存到磁盘。要稍后再次加载结果，请使用 **load** 函数。使用 **sprintf** 和 **datetime**，按照当前日期和时间命名文件。

```
filename = sprintf('experiment-%s',datetime('now','Format','yyyyMMdd"T"HHmmss'));
save(filename,'network','trainingInfo');
```

## 绘制结果

在网络完成训练后，使用 **trainingInfo** 中的信息绘制其训练进度。

使用子图为每个网络分发不同绘图。对于本示例，使用第一行子图绘制训练准确度对轮次编号及验证准确度的图。

```
figure('Units','normalized','Position',[0.1 0.1 0.6 0.6]);
title('Training Progress Plots');
```

```

for i=1:numel(layersCell)
    subplot(2,numel(layersCell),i);
    hold on; grid on;
    ylim([0 100]);
    iterationsPerEpoch = floor(augmentedImdsTrain.NumObservations/options(i).MiniBatchSize);
    epoch = (1:numel(trainingInfo(i).TrainingAccuracy))/iterationsPerEpoch;
    plot(epoch,trainingInfo(i).TrainingAccuracy);
    plot(epoch,trainingInfo(i).ValidationAccuracy,'k','MarkerSize',10);
end
subplot(2,numel(layersCell),1), ylabel('Accuracy');

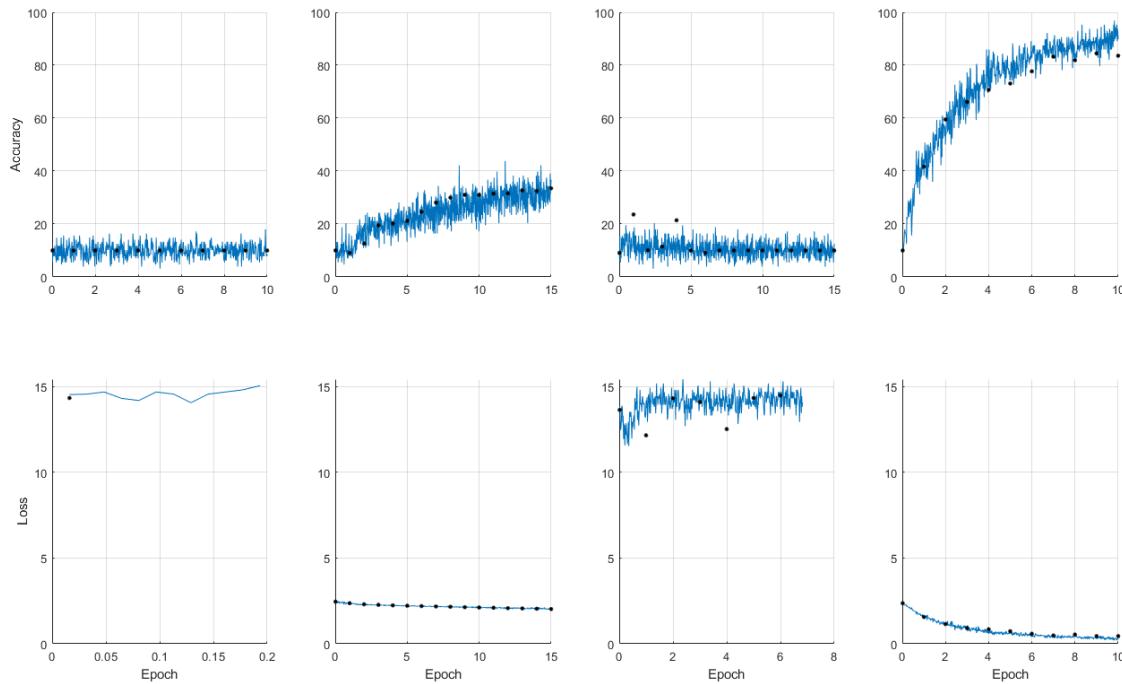
```

然后，使用第二行子图绘制训练损失对轮次编号及验证损失的图。

```

for i=1:numel(layersCell)
    subplot(2,numel(layersCell),numel(layersCell) + i);
    hold on; grid on;
    ylim([0 max([trainingInfo.TrainingLoss])]);
    iterationsPerEpoch = floor(augmentedImdsTrain.NumObservations/options(i).MiniBatchSize);
    epoch = (1:numel(trainingInfo(i).TrainingAccuracy))/iterationsPerEpoch;
    plot(epoch,trainingInfo(i).TrainingLoss);
    plot(epoch,trainingInfo(i).ValidationLoss,'k','MarkerSize',10);
    xlabel('Epoch');
end
subplot(2,numel(layersCell),numel(layersCell)+1), ylabel('Loss');

```



在选择网络后，您可以使用 `classify` 并获得其基于测试数据 `imdsTest` 的准确度。

## 另请参阅

[试验管理器](#) | [augmentedImage datastore](#) | [image datastore](#) | [parfeval](#) (Parallel Computing Toolbox) | | [trainNetwork](#) | [trainingOptions](#)

## 相关示例

- 
- 
- “Use Experiment Manager to Train Networks in Parallel”

## 详细信息

- “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud”

## 监控深度学习训练进度

在训练深度学习网络时，监控训练进度通常很有用。通过在训练过程中绘制各种指标，您可以了解训练的进度情况。例如，您可以确定网络准确度是否改善以及改善速度，还可以确定网络是否开始过拟合训练数据。

在 `trainingOptions` 中将 'training-progress' 指定为 'Plots' 值并开始网络训练时，`trainNetwork` 会创建一个图窗并在每次迭代时显示训练指标。每次迭代都是对梯度的一次估计和对网络参数的一次更新。如果在 `trainingOptions` 中指定验证数据，则每次 `trainNetwork` 验证网络时，该图窗都会显示验证指标。该图窗绘制以下内容：

- **训练准确度** - 针对每个小批量的分类准确度。
- **经过平滑处理的训练准确度** - 经过平滑处理的训练准确度，通过将平滑算法应用于训练准确度来获得。它的噪声低于未平滑的准确度，因此更易于揭示趋势。
- **验证准确度** - 针对整个验证集的分类准确度（使用 `trainingOptions` 指定）。
- **训练损失、经过平滑处理的训练损失和验证损失** - 分别指每个小批量的损失、其经过平滑处理的版本以及验证集的损失。如果网络的最终层是一个 `classificationLayer`，则损失函数是交叉熵损失。有关分类和回归问题的损失函数的详细信息，请参阅“输出层”（第 1-32 页）。

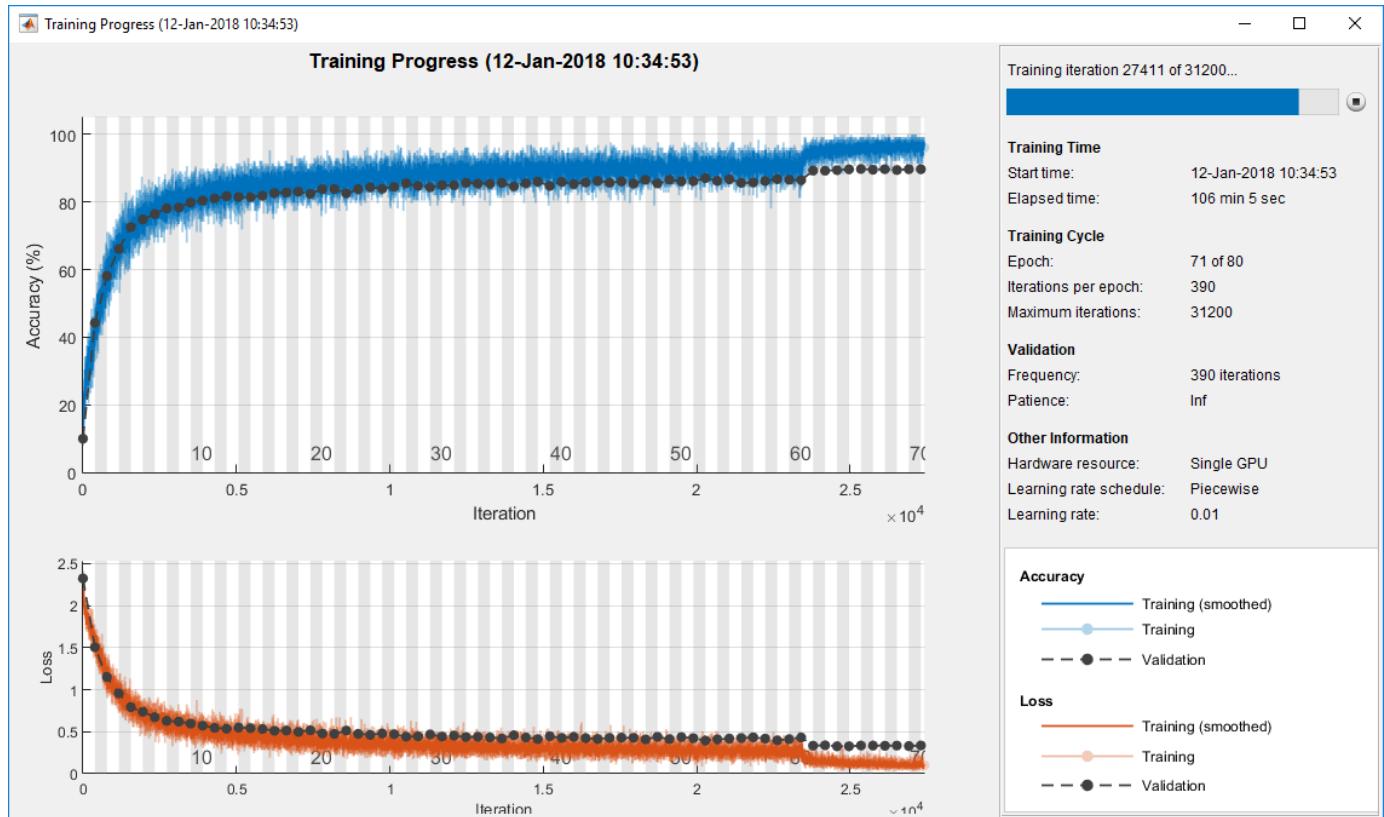
对于回归网络，该图窗绘制均方根误差 (RMSE) 而不是准确度。

图窗使用交替底色来标记**每一轮训练**。一轮训练是对整个训练数据集的一次完整遍历。

在训练过程中，您可以通过点击右上角的停止按钮停止训练并返回网络的当前状态。例如，您可能希望在网络准确度达到稳定水平并且准确度明显不再提高时停止训练。点击停止按钮后，可能需要一段时间才能完成训练。训练完成后，`trainNetwork` 将返回经过训练的网络。

训练结束后，**查看结果**，其中显示最终验证准确度和训练结束的原因。最终验证指标在绘图中标注为 **Final**。如果您的网络包含批量归一化层，则最终验证指标可以与训练过程中评估出的验证指标不同。这是因为在训练完成后，用于批量归一化的均值和方差统计量可能会有所不同。例如，如果 '`BatchNormalizationStatistics`' 训练选项为 '`population`'，则在训练后，软件通过再次使训练数据穿过来完成批量归一化统计，并使用得到的均值和方差。如果 '`BatchNormalizationStatistics`' 训练选项为 '`moving`'，则软件在训练过程中使用运行估计来逼近统计量，并使用统计量的最新值。

在右侧，查看有关训练时间和设置的信息。要了解有关训练选项的详细信息，请参阅“设置参数并训练卷积神经网络”（第 1-35 页）。



### 在训练过程中绘制训练进度

训练网络并在训练过程中绘制训练进度。

加载训练数据，其中包含 5000 个数字图像。留出 1000 个图像用于网络验证。

```
[XTrain,YTrain] = digitTrain4DArrayData;
```

```
idx = randperm(size(XTrain,4),1000);
XValidation = XTrain(:,:,1:idx);
XTrain(:,:,1:idx) = [];
YValidation = YTrain(idx);
YTrain(idx) = [];
```

构建网络以对数字图像数据进行分类。

```
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer
```

```

maxPooling2dLayer(2,'Stride',2)

convolution2dLayer(3,32,'Padding','same')
batchNormalizationLayer
reluLayer

fullyConnectedLayer(10)
softmaxLayer
classificationLayer];

```

指定网络训练的选项。要在训练过程中按固定时间间隔验证网络，请指定验证数据。选择 'ValidationFrequency' 值，以使网络大致在每轮训练都被验证一次。要在训练过程中绘制训练进度，请将 'training-progress' 指定为 'Plots' 值。

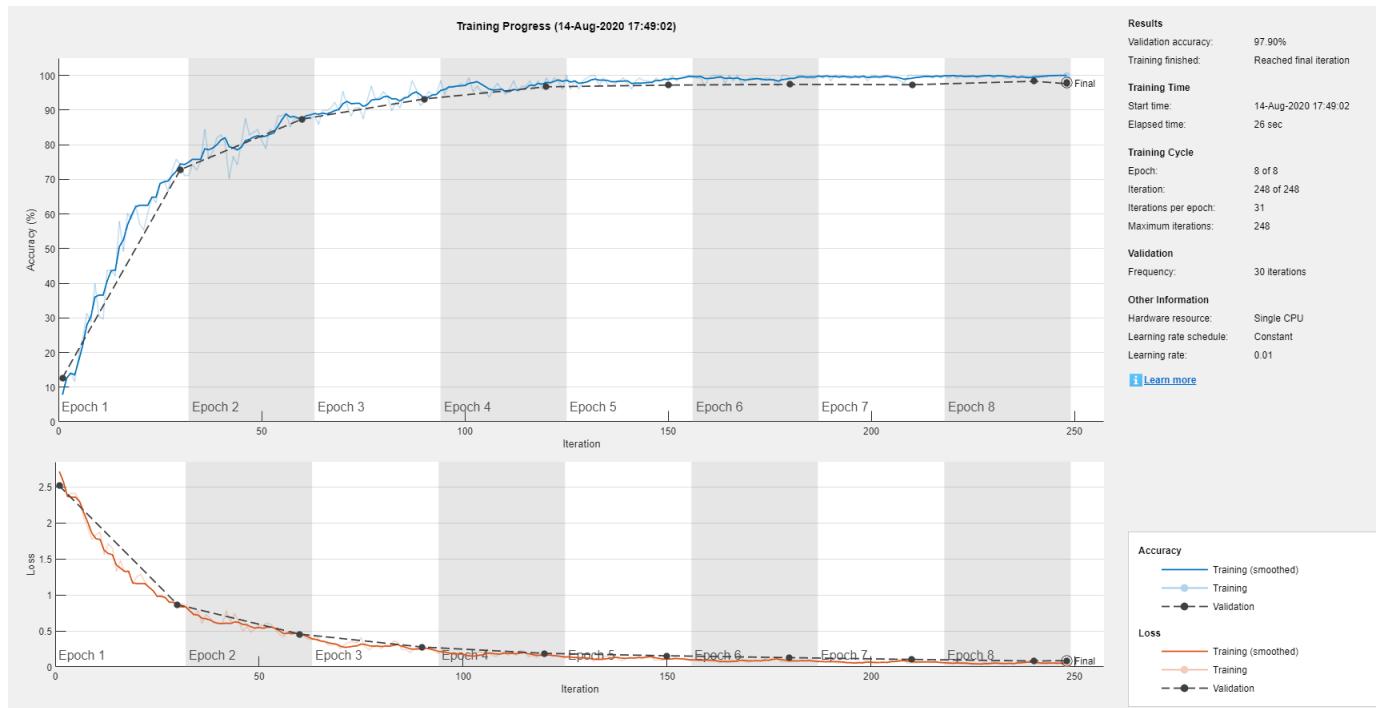
```

options = trainingOptions('sgdm', ...
    'MaxEpochs',8, ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

```

训练网络。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



## 另请参阅

[trainNetwork](#) | [trainingOptions](#)

## 相关示例

- “了解卷积神经网络”（第 1-16 页）
- “指定卷积神经网络的层”（第 1-25 页）
- “设置参数并训练卷积神经网络”（第 1-35 页）
- “预训练的深度神经网络”（第 1-8 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）

## 自定义深度学习网络训练期间的输出

此示例说明如何定义在深度学习神经网络训练期间的每次迭代都运行的输出函数。如果您使用 `trainingOptions` 的 '`OutputFcn`' 名称-值对组参数指定输出函数，则 `trainNetwork` 将分别在训练开始前、每次训练迭代后以及训练结束后各调用这些函数一次。每次调用输出函数时，`trainNetwork` 都会传递一个包含当前迭代编号、损失和准确度等信息的结构体。您可以使用输出函数显示或绘制进度信息，或者停止训练。要提前停止训练，请让输出函数返回 `true`。如果任何输出函数返回 `true`，则训练结束，并且 `trainNetwork` 返回最新网络。

要在验证集的损失不再降低时停止训练，只需分别使用 `trainingOptions` 的 '`ValidationData`' 和 '`ValidationPatience`' 名称-值对组参数指定验证数据和验证容忍度。验证容忍度是指在网络训练停止之前验证集的损失可以大于或等于先前最小损失的次数。您可以使用输出函数添加其他停止条件。此示例说明如何创建在验证数据的分类准确度不再提高时停止训练的输出函数。输出函数在脚本末尾定义。

加载训练数据，其中包含 5000 个数字图像。留出 1000 个图像用于网络验证。

```
[XTrain,YTrain] = digitTrain4DArrayData;
```

```
idx = randperm(size(XTrain,4),1000);
XValidation = XTrain(:, :, :, idx);
XTrain(:, :, :, idx) = [];
YValidation = YTrain(idx);
YTrain(idx) = [];
```

构建网络以对数字图像数据进行分类。

```
layers = [
    imageInputLayer([28 28 1])

    convolution2dLayer(3,8,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,16,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2,'Stride',2)

    convolution2dLayer(3,32,'Padding','same')
    batchNormalizationLayer
    reluLayer

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

指定网络训练的选项。要在训练过程中按固定时间间隔验证网络，请指定验证数据。选择 '`ValidationFrequency`' 值，以便每轮训练都验证一次网络。

要在验证集的分类准确度不再提高时停止训练，请将 `stopIfAccuracyNotImproving` 指定为输出函数。`stopIfAccuracyNotImproving` 的第二个输入参数是在网络训练停止之前验证集的准确度可以小于或等于先前最高准确度的次数。为最大训练轮数指定一个较大的值。训练不应到达最后一轮，它应该会自动停止。

```

miniBatchSize = 128;
validationFrequency = floor(numel(YTrain)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'InitialLearnRate',0.01, ...
    'MaxEpochs',100, ...
    'MiniBatchSize',miniBatchSize, ...
    'VerboseFrequency',validationFrequency, ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',validationFrequency, ...
    'Plots','training-progress', ...
    'OutputFcn',@(info)stopIfAccuracyNotImproving(info,3));

```

训练网络。当验证准确度停止升高时，训练停止。

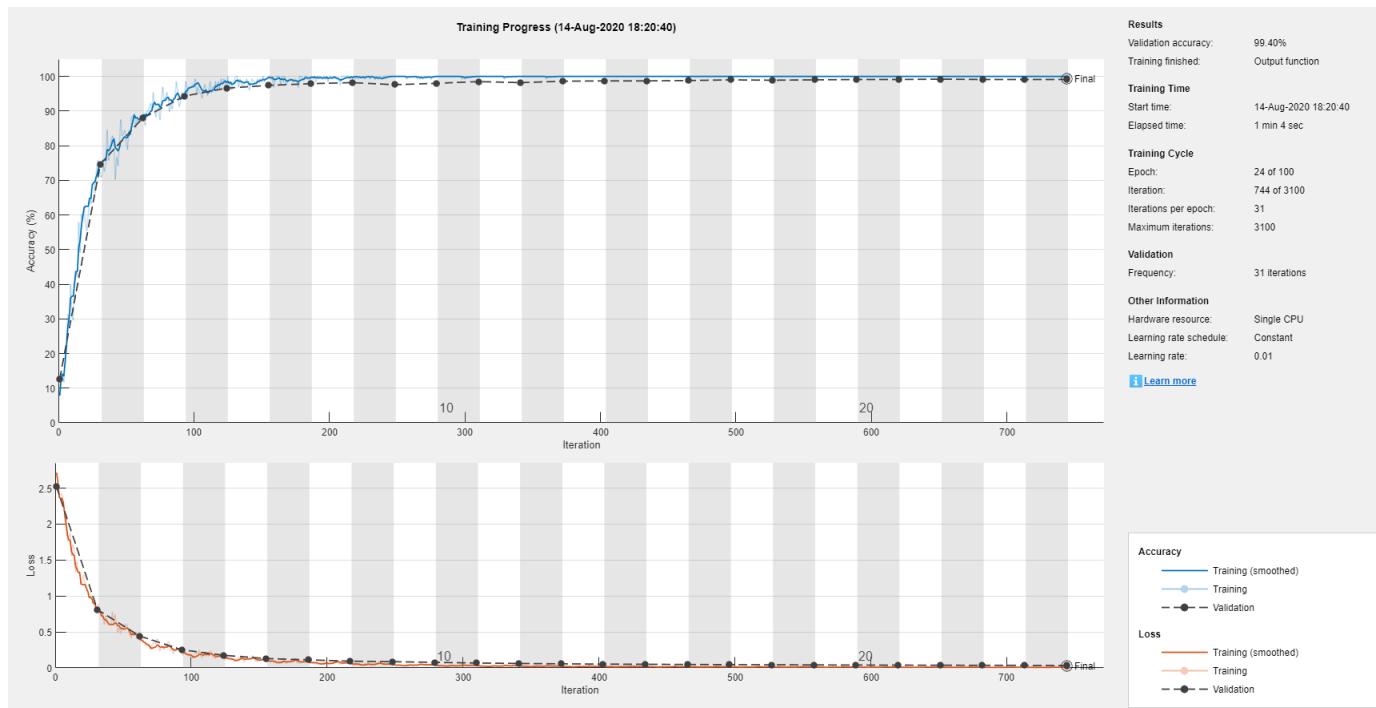
```
net = trainNetwork(XTrain,YTrain,layers,options);
```

Training on single CPU.

Initializing input data normalization.

Epoch	Iteration	Time Elapsed	Mini-batch Accuracy	Validation Accuracy	Mini-batch Loss	Validation Loss	Base Learning Rate
		(hh:mm:ss)					
1	1	00:00:03	7.81%	12.70%	2.7155	2.5169	0.0100
1	31	00:00:06	71.09%	74.70%	0.8805	0.8120	0.0100
2	62	00:00:08	87.50%	87.90%	0.3866	0.4448	0.0100
3	93	00:00:11	94.53%	94.30%	0.2178	0.2529	0.0100
4	124	00:00:13	96.09%	96.60%	0.1433	0.1759	0.0100
5	155	00:00:15	100.00%	97.40%	0.0994	0.1306	0.0100
6	186	00:00:18	99.22%	97.90%	0.0786	0.1126	0.0100
7	217	00:00:20	99.22%	98.20%	0.0552	0.0938	0.0100
8	248	00:00:23	100.00%	97.60%	0.0429	0.0871	0.0100
9	279	00:00:26	100.00%	98.00%	0.0338	0.0777	0.0100
10	310	00:00:28	100.00%	98.50%	0.0271	0.0681	0.0100
11	341	00:00:31	100.00%	98.20%	0.0237	0.0623	0.0100
12	372	00:00:33	100.00%	98.60%	0.0212	0.0570	0.0100
13	403	00:00:36	100.00%	98.70%	0.0186	0.0533	0.0100
14	434	00:00:38	100.00%	98.70%	0.0163	0.0507	0.0100
15	465	00:00:41	100.00%	98.80%	0.0143	0.0483	0.0100
16	496	00:00:43	100.00%	99.00%	0.0127	0.0457	0.0100
17	527	00:00:46	100.00%	98.90%	0.0113	0.0435	0.0100
18	558	00:00:48	100.00%	99.00%	0.0102	0.0416	0.0100

	19	589	00:00:51	100.00%	99.10%	0.0093	0.0400	0.0100	
	20	620	00:00:53	100.00%	99.10%	0.0086	0.0387	0.0100	
	21	651	00:00:56	100.00%	99.20%	0.0081	0.0375	0.0100	
	22	682	00:00:58	100.00%	99.10%	0.0076	0.0364	0.0100	
	23	713	00:01:01	100.00%	99.10%	0.0073	0.0355	0.0100	
	24	744	00:01:03	100.00%	99.10%	0.0069	0.0346	0.0100	
	=====								



### 定义输出函数

定义输出函数 `stopIfAccuracyNotImproving(info,N)`，该函数在验证数据的最优分类准确度连续 N 次网络验证都没有提高时停止网络训练。此标准类似于使用验证损失的内置停止条件，只不过它适用于分类准确度而不是损失。

```
function stop = stopIfAccuracyNotImproving(info,N)
```

```
stop = false;
```

```
% Keep track of the best validation accuracy and the number of validations for which
% there has not been an improvement of the accuracy.
```

```
persistent bestValAccuracy
persistent valLag
```

```
% Clear the variables when training starts.
```

```
if info.State == "start"
```

```
bestValAccuracy = 0;
```

```
valLag = 0;
```

```
elseif ~isempty(info.ValidationLoss)

% Compare the current validation accuracy to the best accuracy so far,
% and either set the best accuracy to the current accuracy, or increase
% the number of validations for which there has not been an improvement.
if info.ValidationAccuracy > bestValAccuracy
    valLag = 0;
    bestValAccuracy = info.ValidationAccuracy;
else
    valLag = valLag + 1;
end

% If the validation lag is at least N, that is, the validation accuracy
% has not improved for at least N validations, then return true and
% stop training.
if valLag >= N
    stop = true;
end

end
end
```

## 另请参阅

[trainNetwork](#) | [trainingOptions](#)

## 相关示例

- “了解卷积神经网络”（第 1-16 页）
- “设置参数并训练卷积神经网络”（第 1-35 页）
- “在 MATLAB 中进行深度学习”（第 1-2 页）
- “比较层权重初始化函数”（第 17-22 页）
- “指定自定义权重初始化函数”（第 17-16 页）

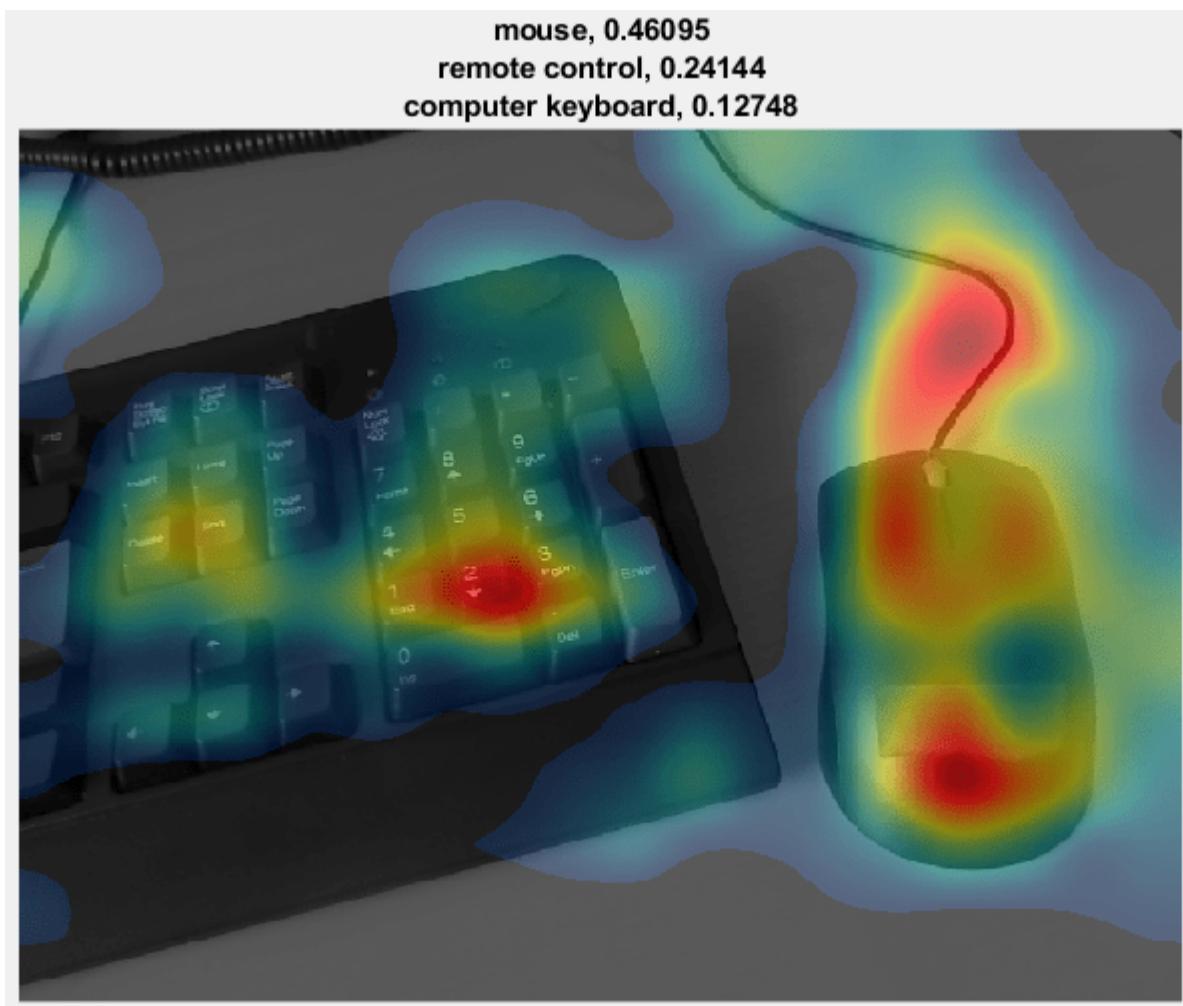
## 使用类激活映射调查网络预测

此示例说明如何使用类激活映射 (CAM) 来调查和解释用于图像分类的深度卷积神经网络的预测。

深度学习网络通常被认为是“黑匣子”，人们无法搞清楚网络到底学到了什么或网络输入的哪一部分与网络预测有关。当这些模型失败并给出不正确的预测时，往往错得十分离谱，而且不会给出任何警告或解释。类激活映射 [1] 是一种可用于直观解释卷积神经网络预测的方法。不正确的、看似不合理的预测通常会有合理的解释。使用类激活映射，您可以检查输入图像的特定部分是否让网络产生“混淆”并导致它作出错误的预测。

您可以使用类激活映射来识别训练集中的偏置，并提高模型准确度。如果您发现网络基于错误的特征进行预测，则您可以通过收集更好的数据来使网络更加稳健。例如，假设您训练网络来区分猫和狗的图像。该网络在训练集上具有很高的准确度，但在现实世界的示例中表现不佳。通过对训练示例使用类激活映射，您发现网络不是基于图像中的猫和狗而是基于背景在进行预测。然后您意识到您的所有猫图片都有红色背景，您的所有狗图片都有绿色背景，这是网络在训练中学习到的背景颜色。然后您可以收集没有这种偏置的新数据。

以下示例类激活映射显示输入图像的哪些区域对预测类 `mouse` 贡献最大。红色区域贡献最大。



## 加载预训练的网络和网络摄像头

加载预训练的卷积神经网络进行图像分类。SqueezeNet、GoogLeNet、ResNet-18 和 MobileNet-v2 是相对较快的网络。SqueezeNet 是最快的网络，它的类激活映射的分辨率是其他网络映射的四倍。您无法对网络末端有多个全连接层的网络（如 AlexNet、VGG-16 和 VGG-19）使用类激活映射。

```
netName = "squeezenet";
net = eval(netName);
```

创建一个 `webcam` 对象并连接到您的网络摄像头。

```
camera = webcam;
```

提取网络的图像输入大小和输出类。在本示例末尾定义的 `activationLayerName` 辅助函数返回从中提取激活的层的名称。该层是网络的最后一个卷积层后的 ReLU 层。

```
inputSize = net.Layers(1).InputSize(1:2);
classes = net.Layers(end).Classes;
layerName = activationLayerName(netName);
```

## 显示类激活映射

创建一个图窗并以循环方式执行类激活映射。要终止循环的执行，请关闭图窗。

```
h = figure('Units','normalized','Position',[0.05 0.05 0.9 0.8],'Visible','on');

while ishandle(h)
```

使用网络摄像头拍摄快照。调整图像大小，使其最短边的长度（本例中为图像高度）等于网络的图像输入大小。在您调整大小时，保留图像的纵横比。您也可以将图像大小调整为更大或更小。放大图像会提高最终类激活映射的分辨率，但会导致整体预测不太准确。

计算网络最后一个卷积层后的 ReLU 层中调整大小后的图像的激活区域。

```
im = snapshot(camera);
imResized = imresize(im,[inputSize(1), NaN]);
imageActivations = activations(net,imResized,layerName);
```

特定类的类激活映射是最后一个卷积层之后的 ReLU 层的激活映射，由每个激活对该类的最终得分的贡献程度来加权。这些权重等于网络的最终全连接层对该类的权重。SqueezeNet 没有最终全连接层。最后一个卷积层后的 ReLU 层的输出已经是类激活映射。

您可以为任何输出类生成一个类激活映射。例如，如果网络分类不正确，您可以比较真实类和预测类的类激活映射。对于本示例，我们为得分最高的预测类生成类激活映射。

```
scores = squeeze(mean(imageActivations,[1 2]));

if netName ~= "squeezenet"
    fcWeights = net.Layers(end-2).Weights;
    fcBias = net.Layers(end-2).Bias;
    scores = fcWeights*scores + fcBias;

    [~,classIds] = maxk(scores,3);

    weightVector = shiftdim(fcWeights(classIds(1),:,-1));
    classActivationMap = sum(imageActivations.*weightVector,3);
else
```

```
[~,classIds] = maxk(scores,3);
classActivationMap = imageActivations(:, :, classIds(1));
end
```

计算顶层类标签和最终归一化类分数。

```
scores = exp(scores)/sum(exp(scores));
maxScores = scores(classIds);
labels = classes(classIds);
```

绘制类激活映射。在第一个子图中显示原始图像。在第二个子图中，使用在本示例末尾定义的 **CAMshow** 辅助函数，在原始图像的暗灰度版本之上显示类激活映射。显示前三个预测标签及其预测分数。

```
subplot(1,2,1)
imshow(im)

subplot(1,2,2)
CAMshow(im,classActivationMap)
title(string(labels) + "," + string(maxScores));

drawnow

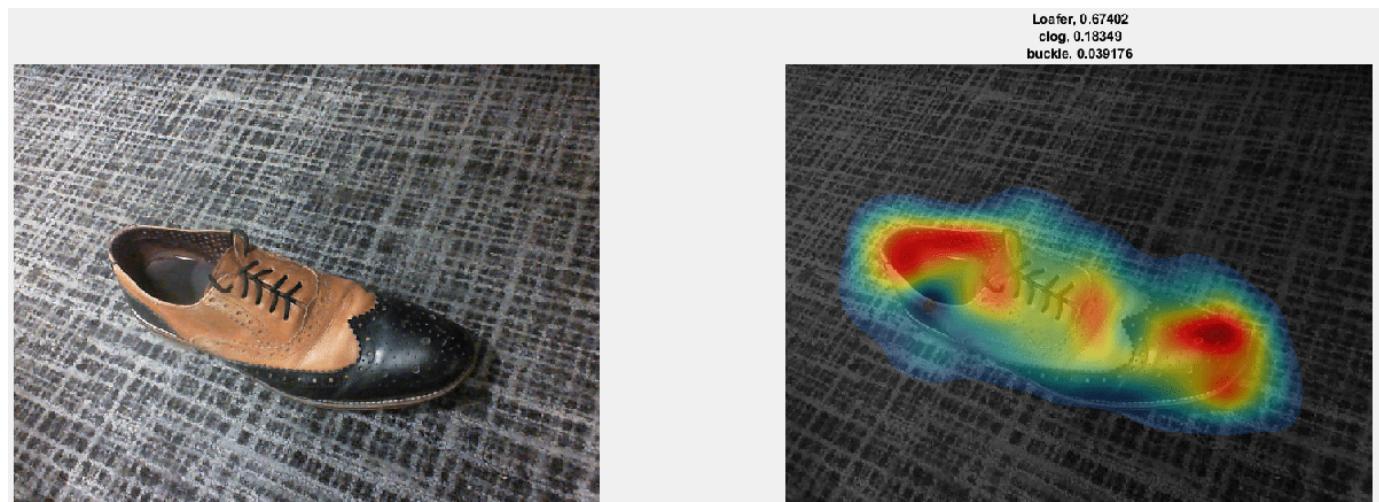
end
```

清除网络摄像头对象。

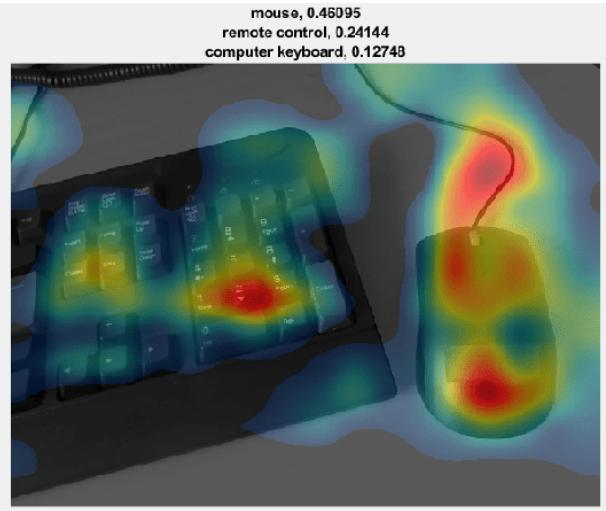
```
clear camera
```

### 映射示例

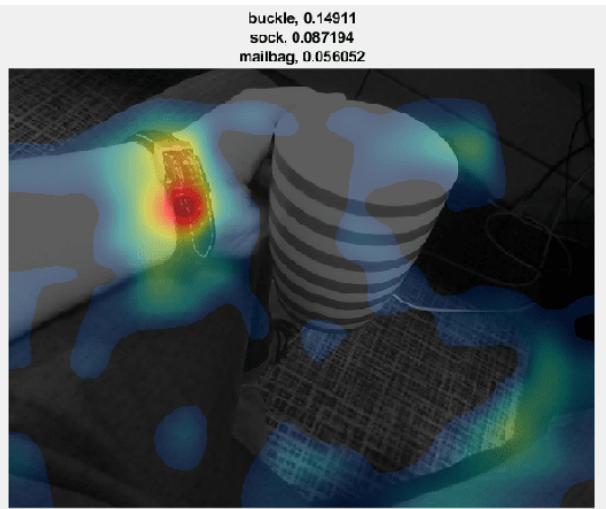
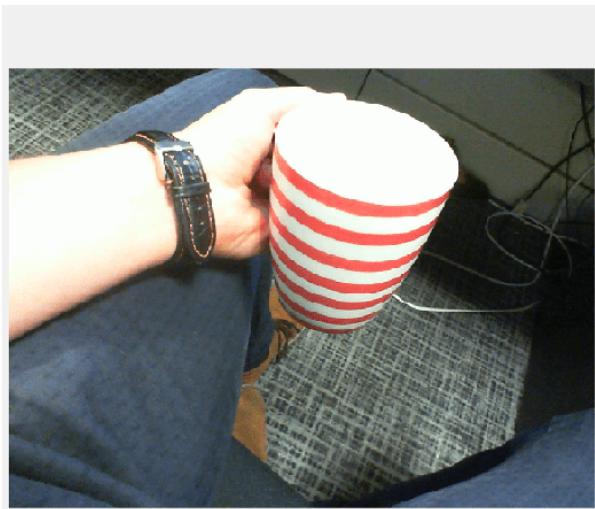
网络将下列图像中的物品正确识别为休闲鞋（一种鞋）。右侧图像中的类激活映射显示输入图像的每个区域对预测的类 **Loafer** 的贡献。红色区域贡献最大。网络基于整只鞋对其分类，但最强的输入来自红色区域 - 即鞋尖和鞋口。



网络将下面图像分类为鼠标。正如类激活映射所示，预测不仅基于图像中的鼠标，还基于键盘。由于训练集可能包含许多鼠标与键盘一起出现的图像，网络预测包含键盘的图像更可能包含鼠标。



网络将下面的咖啡杯图像归类为扣环。如类激活映射所示，网络对图像进行了错误分类，因为图像包含太多容混淆物品。网络检测并重点关注手表腕带，而不是咖啡杯。



## 辅助函数

**CAMshow(im,CAM)** 在图像 im 的暗灰色版本上叠加类激活映射 CAM。该函数将类激活映射调整为 im 的大小，对其进行归一化，为其设置下限阈值，并使用 jet 颜色图将其可视化。

```

function CAMshow(im,CAM)
imSize = size(im);
CAM = imresize(CAM,imSize(1:2));
CAM = normalizeImage(CAM);
CAM(CAM<0.2) = 0;
cmap = jet(255).*linspace(0,1,255)';
CAM = ind2rgb(uint8(CAM*cmap)*255;

combinedImage = double(rgb2gray(im))/2 + CAM;
combinedImage = normalizeImage(combinedImage)*255;
imshow(uint8(combinedImage));
end

```

```
function N = normalizeImage(I)
minimum = min(I(:));
maximum = max(I(:));
N = (I-minimum)/(maximum-minimum);
end

function layerName = activationLayerName(netName)

if netName == "squeezeNet"
    layerName = 'relu_conv10';
elseif netName == "googlenet"
    layerName = 'inception_5b-output';
elseif netName == "resnet18"
    layerName = 'res5b_relu';
elseif netName == "mobilenetv2"
    layerName = 'out_relu';
end

end
```

### 参考

[1] Zhou, Bolei, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. "Learning deep features for discriminative localization." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2921-2929. 2016.

### 另请参阅

[activations](#) | [squeezeNet](#) | [occlusionSensitivity](#) | [gradCAM](#) | [imageLIME](#)

### 相关示例

- “预训练的深度神经网络” (第 1-8 页)
- “Grad-CAM Reveals the Why Behind Deep Learning Decisions”
- “Understand Network Predictions Using LIME”
- “可视化卷积神经网络的激活区域” (第 5-35 页)
- “Understand Network Predictions Using Occlusion”
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 可视化卷积神经网络的激活区域

此示例说明如何将图像馈送到卷积神经网络并显示网络的不同层的激活区域。通过将激活区域与原始图像进行比较，检查激活区域并发现网络学习的特征。发现较浅层中的通道学习颜色和边缘等简单特征，而较深层中的通道学习眼睛等复杂特征。以这种方式识别特征可以帮助您了解网络学习的内容。

该示例需要 Deep Learning Toolbox™ 和 Image Processing Toolbox™。

### 加载预训练的网络和数据

加载一个预训练的 SqueezeNet 网络。

```
net = squeezenet;
```

读取并显示图像。保存图像大小，以便稍后使用。

```
im = imread('face.jpg');  
imshow(im)
```

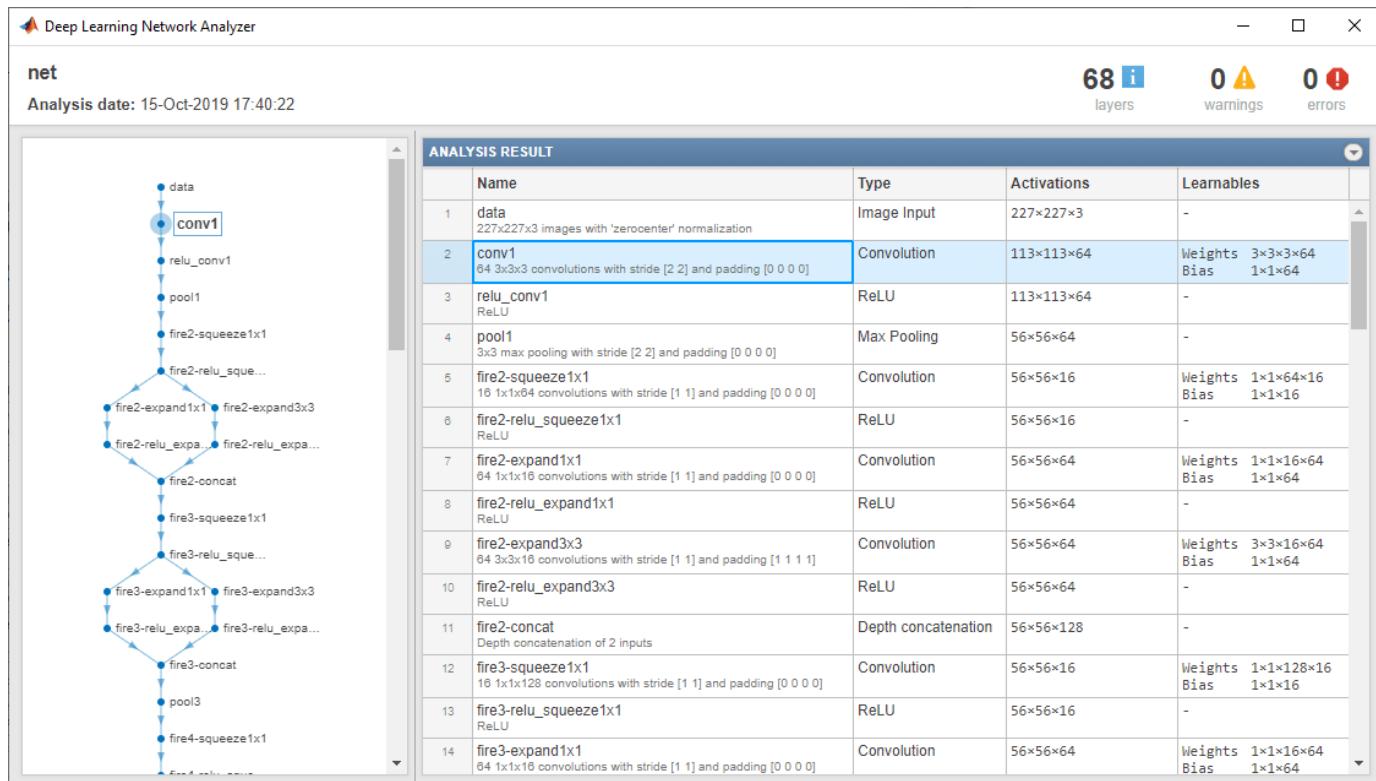


```
imgSize = size(im);  
imgSize = imgSize(1:2);
```

## 查看网络架构

分析该网络，了解您可以查看哪些层。卷积层使用可学习的参数执行卷积。网络学习识别有用的特征，通常每个通道对应一个特征。观察到第一个卷积层有 64 个通道。

**analyzeNetwork(net)**



图像输入层指定输入大小。您可以在将图像通过网络之前调整图像大小，但网络也可以处理较大的图像。如果您为网络提供较大的图像，则激活区域也会变大。但是，由于网络是基于大小为 227×227 的图像进行训练的，因此无法识别超过该大小的对象或特征。

## 显示第一个卷积层的激活区域

观察卷积层中的哪些区域在图像上激活，并将其与原始图像中的相应区域进行比较，以研究特征。卷积神经网络的每层由许多称为通道的二维数组组成。用图像对网络进行一轮训练，并检查 **conv1** 层的输出激活区域。

```
act1 = activations(net,im,'conv1');
```

激活区域以三维数组的形式返回，其中第三个维度对 **conv1** 层上的通道进行索引。要使用 **imtile** 函数显示这些激活区域，请将数组重构为四维。**imtile** 的输入中的第三个维度表示图像颜色。将第三个维度的大小设置为 1，因为激活区域没有颜色。第四个维度对通道进行索引。

```
sz = size(act1);
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);
```

现在您可以显示激活区域。每次激活可能采用任何值，因此使用 **mat2gray** 归一化输出。缩放所有激活值，以使最小激活值为 0，最大激活值为 1。在 8×8 网格上显示 64 个图像，层中的每个通道对应一个图像。

```
I = imtile(mat2gray(act1),'GridSize',[8 8]);
imshow(I)
```



### 调查特定通道中的激活区域

激活区域网格中的每个图块都是 `conv1` 层中某个通道的输出。白色像素表示强的正激活区域，黑色像素表示强的负激活区域。主要为灰色的通道未对输入图像进行强烈激活。通道激活区域中的像素位置对应于原始图像中的相同位置。通道中某个位置的白色像素表示该通道在该位置强烈激活。

调整通道 22 的激活区域大小以使其与原始图像具有相同的大小，并显示激活区域。

```
act1ch22 = act1(:,:,22);
act1ch22 = mat2gray(act1ch22);
```

```
act1ch22 = imresize(act1ch22,imgSize);
```

```
I = imtile({im,act1ch22});  
imshow(I)
```



您可以看到此通道在红色像素上激活，因为通道中的偏白的像素对应于原始图像中的红色区域。

### 查找最强的激活通道

您还可以通过编程方式调查具有较大激活区域的通道来尝试查找感兴趣的通道。使用 `max` 函数查找激活区域最大的通道，调整大小并显示激活区域。

```
[maxValue,maxValueIndex] = max(max(max(act1)));  
act1chMax = act1(:,:,maxValueIndex);  
act1chMax = mat2gray(act1chMax);  
act1chMax = imresize(act1chMax,imgSize);  
  
I = imtile({im,act1chMax});  
imshow(I)
```



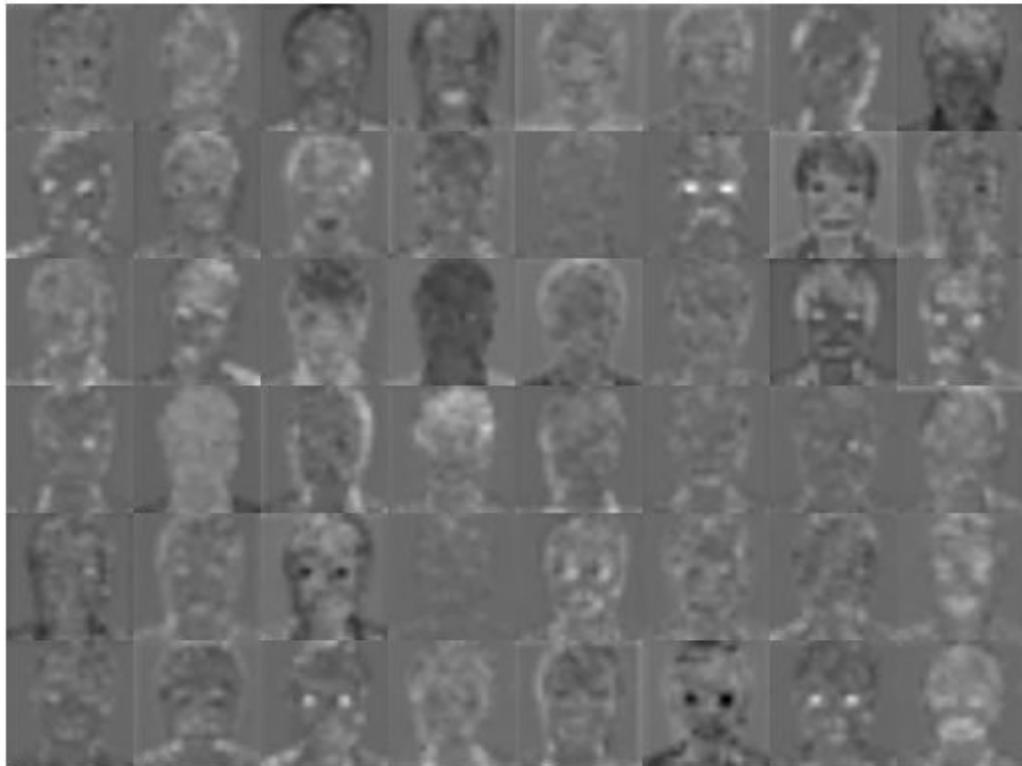
与原始图像进行比较，注意此通道在边缘激活。它在浅色左侧/深色右侧边缘上正激活，在深色左侧/浅色右侧边缘上负激活。

### 调查更深的层

大多数卷积神经网络在第一个卷积层中学习检测颜色和边缘等特征。在更深的卷积层中，网络学习检测更复杂的特征。较深的层通过组合较浅层的特征来构建其特征。以调查 `conv1` 层的方式调查 `fire6-squeeze1x1` 层。计算、重构并在网格中显示激活区域。

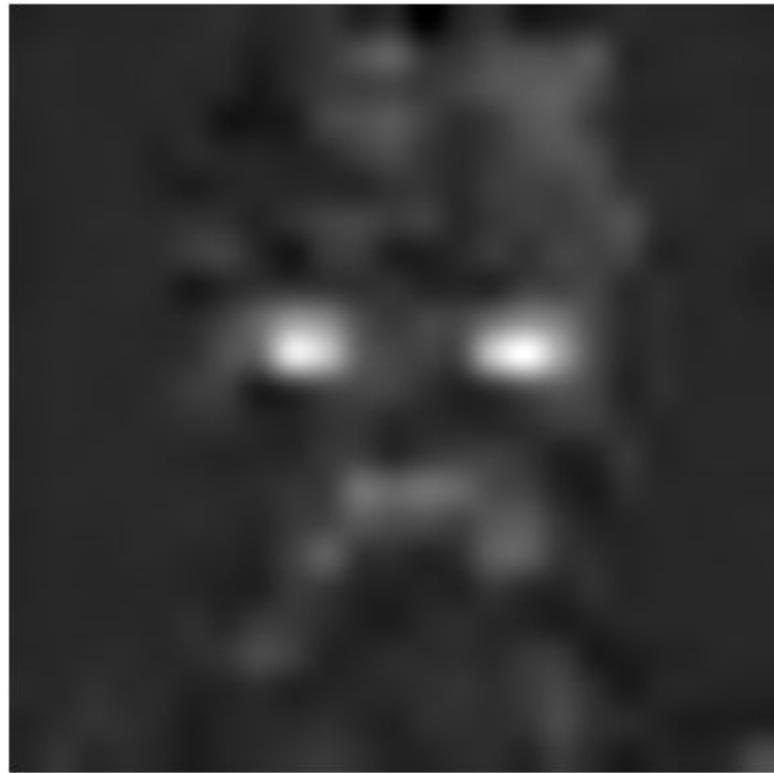
```
act6 = activations(net,im,'fire6-squeeze1x1');
sz = size(act6);
act6 = reshape(act6,[sz(1) sz(2) 1 sz(3)]);

I = imtile(imresize(mat2gray(act6),[64 64]),'GridSize',[6 8]);
imshow(I)
```



图像太多，无法详细调查，因此不妨关注一些更有趣的图像。显示 fire6-squeeze1x1 层中最强的激活区域。

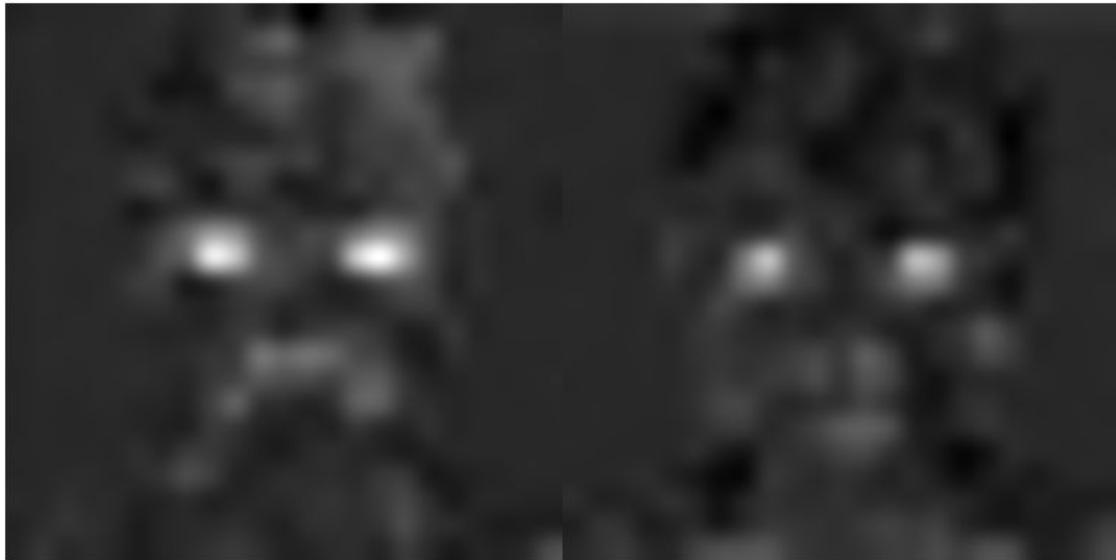
```
[maxValue6,maxValueIndex6] = max(max(max(act6)));
act6chMax = act6(:,:,maxValueIndex6);
imshow(imresize(mat2gray(act6chMax),imgSize))
```



在本例中，最大激活通道在详细特征方面不像其他一些通道那样令人感兴趣，并且表现出强力的负（深色）激活以及正（浅色）激活。此通道可能专注于面部。

在所有通道的网格中，可能有通道针对眼睛激活。进一步调查通道 14 和 47。

```
I = imtile(imresize(mat2gray(act6(:,:, [14 47])), imgSize));  
imshow(I)
```



许多通道包含同时存在浅色和深色的激活区域。它们分别是正激活区域和负激活区域。但是，由于 fire6-squeeze1x1 层后面是修正线性单元 (ReLU)，因此只使用正激活区域。要只调查正激活区域，请重复分析以可视化 fire6-relu\_squeeze1x1 层的激活区域。

```
act6relu = activations(net,im,'fire6-relu_squeeze1x1');
sz = size(act6relu);
act6relu = reshape(act6relu,[sz(1) sz(2) 1 sz(3)]);

I = imtile(imresize(mat2gray(act6relu(:,:,14:47)),imgSize));
imshow(I)
```



与 fire6-squeeze1x1 层的激活区域相比，fire6-relu\_squeeze1x1 层的激活区域清楚地定位到具有强面部特征的图像区域。

#### 测试通道是否识别眼睛

检查 fire6-relu\_squeeze1x1 层的通道 14 和 47 是否针对眼睛激活。将两眼一睁一闭的新图像输入到网络中，并将得到的激活区域与原始图像的激活区域进行比较。

读取并显示两眼一睁一闭的图像，并计算 fire6-relu\_squeeze1x1 层的激活区域。

```
imClosed = imread('face-eye-closed.jpg');
imshow(imClosed)
```



```
act6Closed = activations(net,imClosed,'fire6-relu_squeeze1x1');
sz = size(act6Closed);
act6Closed = reshape(act6Closed,[sz(1),sz(2),1,sz(3)]);
```

在一个图窗中绘制图像和激活区域。

```
channelsClosed = repmat(imresize(mat2gray(act6Closed(:,:,14:47))),imgSize),[1 1 3]);
channelsOpen = repmat(imresize(mat2gray(act6relu(:,:,14:47))),imgSize),[1 1 3]);
I = imtile(cat(4,im,channelsOpen*255,imClosed,channelsClosed*255));
imshow(I)
title('Input Image, Channel 14, Channel 47');
```



您可以从激活区域中看到，通道 14 和 47 都针对单个眼睛激活，并且在某种程度上也针对嘴周围的区域激活。

网络从未被告知要学习眼睛的特征，但它已学习到眼睛是区分图像类的一个有用特征。以前的机器学习方法通常手动设计特定于问题的特征，但这些深度卷积网络可以为自己学习有用的特征。例如，学习识别眼睛可以帮助网络区分猎豹和豹纹地毯。

## 另请参阅

[squeezenet](#) | [activations](#) | [deepDreamImage](#)

## 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “预训练的深度神经网络” (第 1-8 页)
- “使用 GoogLeNet 的 Deep Dream 图像” (第 5-2 页)
- “可视化卷积神经网络的特征” (第 5-46 页)

## 可视化卷积神经网络的特征

此示例说明如何可视化卷积神经网络学习的特征。

卷积神经网络使用特征对图像进行分类。网络在训练过程中自行学习这些特征。网络在训练过程中学到的内容有时不明确。但是，您可以使用 `deepDreamImage` 函数将学习的特征可视化。

卷积层输出一个激活三维体，其中沿第三个维度的切片对应于应用到层输入的单个滤波器。网络末尾的全连接层输出的通道对应于较浅层学习的特征的高级组合。

您可以使用 `deepDreamImage` 生成可强烈激活网络层特定通道的图像，从而将所学习的特征可视化。

该示例需要 Deep Learning Toolbox™，以及 Deep Learning Toolbox Model for GoogLeNet Network 支持包。

### 加载预训练网络

加载预训练的 GoogLeNet 网络。

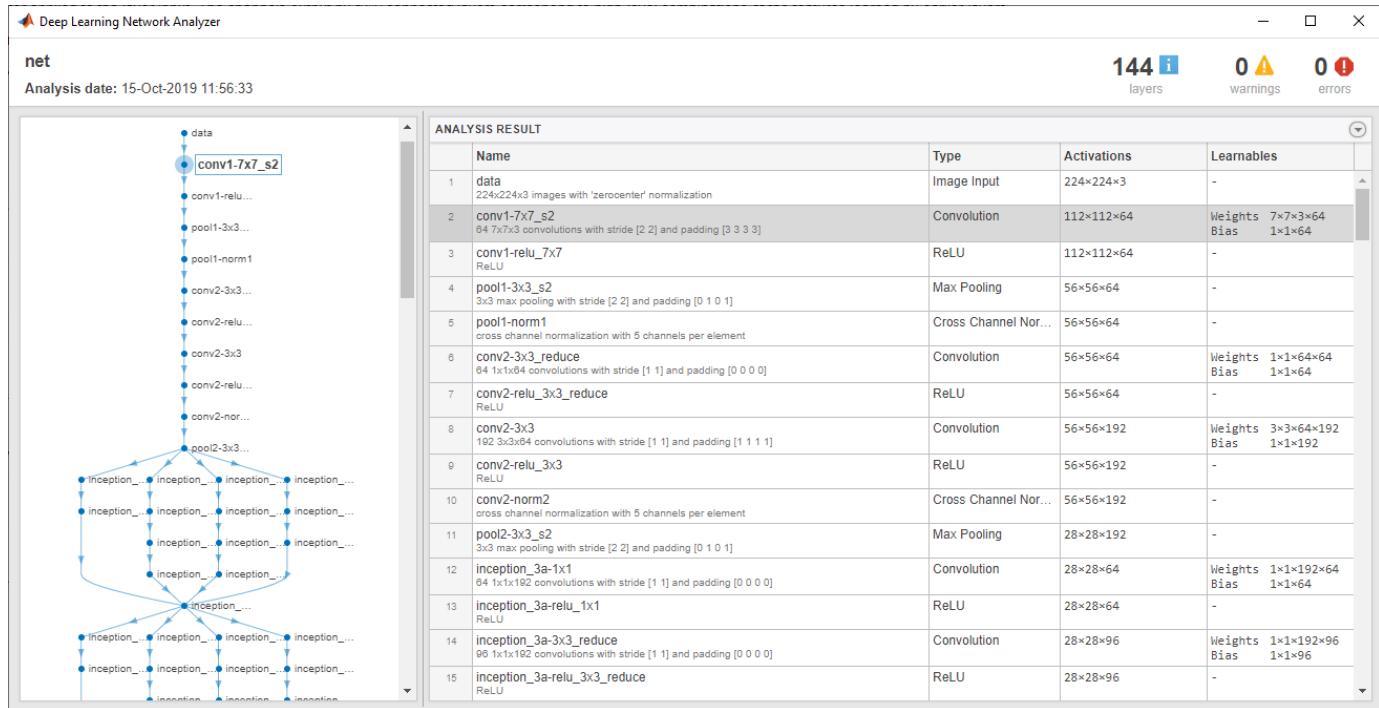
```
net = googlenet;
```

### 可视化较浅的卷积层

GoogLeNet 网络中有多个卷积层。靠近网络开头的卷积层具有较小的感受野，用于学习较小的低级特征。靠近网络末端的层具有较大的感受野，用于学习较大的特征。

使用 `analyzeNetwork` 属性，查看网络架构并找到卷积层。

```
analyzeNetwork(net)
```



### 卷积层 1 上的特征

将 `layer` 设置为第一个卷积层。该层是网络中的第二层，名为 '`conv1-7x7_s2`'。

```
layer = 2;
name = net.Layers(layer).Name

name =
'conv1-7x7_s2'
```

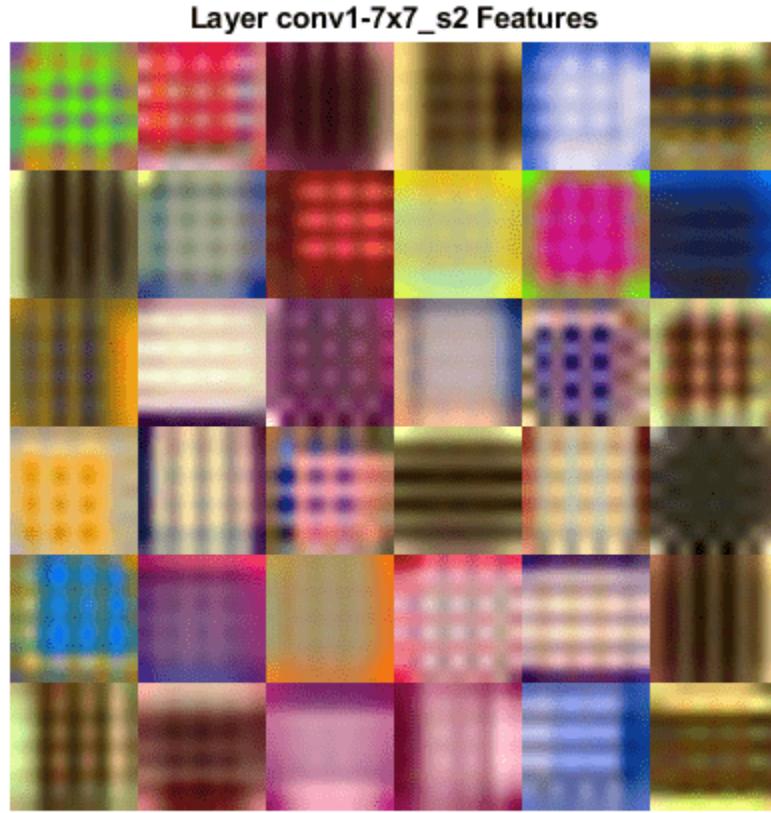
通过将 `channels` 设置为索引 1:36 的向量，使用 `deepDreamImage` 可可视化该层学习的前 36 个特征。将 '`PyramidLevels`' 设置为 1，以避免图像缩放。要将图像显示在一起，可以使用 `imtile`。

默认情况下，`deepDreamImage` 使用兼容的 GPU（如果可用）。否则将使用 CPU。使用 GPU 需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅 “GPU Support by Release”（Parallel Computing Toolbox）。

```
channels = 1:36;
I = deepDreamImage(net,name,channels, ...
    'PyramidLevels',1);
```

Iteration	Activation	Pyramid Level
	Strength	
1	0.26	1
2	6.99	1
3	14.24	1
4	21.49	1
5	28.74	1
6	35.99	1
7	43.24	1
8	50.50	1
9	57.75	1
10	65.00	1

```
figure
I = imtile(I,'ThumbnailSize',[64 64]);
imshow(I)
title(['Layer ',name,' Features'],'Interpreter','none')
```



这些图像主要包含边缘和颜色，指示层 'conv1-7x7\_s2' 中的滤波器是边缘检测器和颜色滤波器。

### 卷积层 2 上的特征

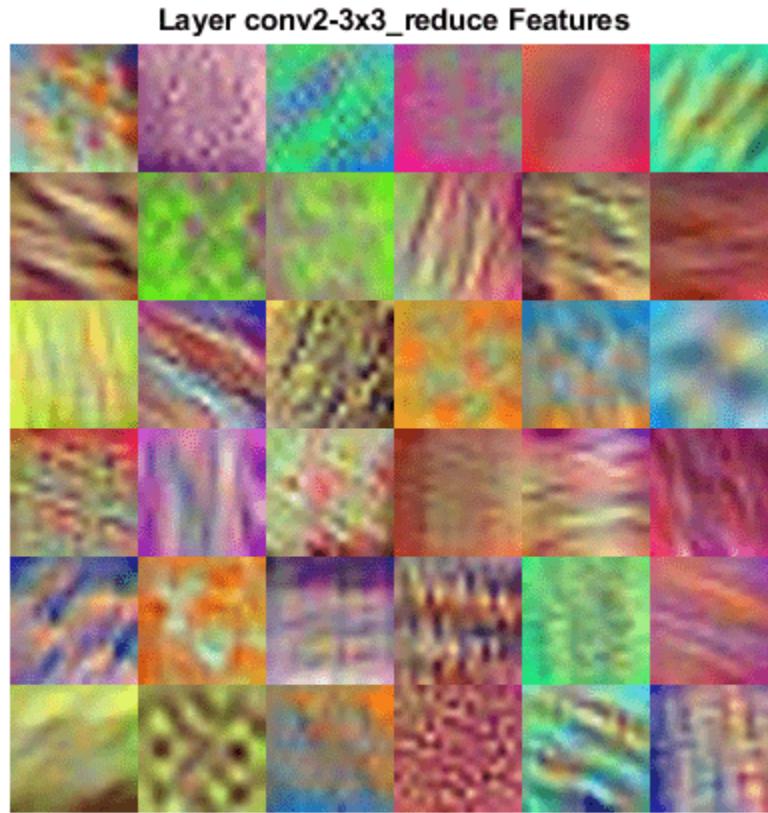
第二个卷积层名为 'conv2-3x3\_reduce'，对应于层 6。通过将 channels 设置为索引 1:36 的向量，可视化该层学习的前 36 个特征。

要在优化过程中隐藏详细输出，请在调用 `deepDreamImage` 时将 'Verbose' 设置为 'false'。

```
layer = 6;
name = net.Layers(layer).Name

name =
'conv2-3x3_reduce'

channels = 1:36;
I = deepDreamImage(net,name,channels, ...
    'Verbose',false, ...
    'PyramidLevels',1);
figure
I = imtile(I,'ThumbnailSize',[64 64]);
imshow(I)
name = net.Layers(layer).Name;
title(['Layer ',name,' Features'],'Interpreter','none')
```



该层的滤波器将检测比第一个卷积层更复杂的模式。

### 可视化较深的卷积层

较深的层学习较浅层学习的特征的高级组合。

增加金字塔等级数和每个金字塔等级的迭代次数可以生成更详细的图像，但代价是额外计算。您可以使用 'NumIterations' 选项增加迭代次数，并使用 'PyramidLevels' 选项增加金字塔层级数。

```

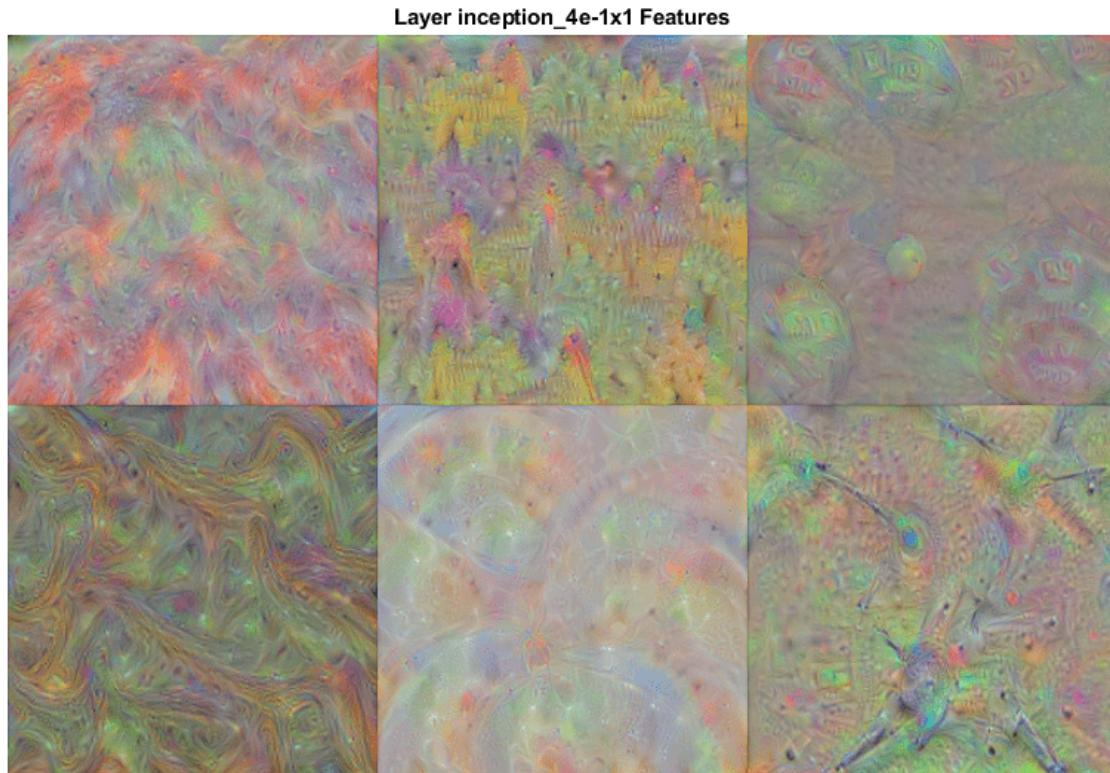
layer = 97;
name = net.Layers(layer).Name

name =
'inception_4e-1x1'

channels = 1:6;
I = deepDreamImage(net,name,channels, ...
    'Verbose',false, ...
    "NumIterations",20, ...
    'PyramidLevels',2);
figure
I = imtile(I,'ThumbnailSize',[250 250]);
imshow(I)

```

```
name = net.Layers(layer).Name;
title(['Layer ',name,' Features'],'Interpreter','none')
```



请注意，越深入网络的层会产生越详细的滤波器，这些滤波器已学习了复杂的模式和纹理。

### 可视化全连接层

要生成最接近每个类的图像，请选择全连接层，并将 **channels** 设置为类的索引。

选择全连接层（层 142）。

```
layer = 142;
name = net.Layers(layer).Name
```

```
name =
'loss3-classifier'
```

通过将 **channels** 设置为这些类名称的索引，选择要可视化的类。

```
channels = [114 293 341 484 563 950];
```

这些类存储在输出层（最后一层）的 **Classes** 属性中。您可以通过选择 **channels** 中的条目来查看所选类的名称。

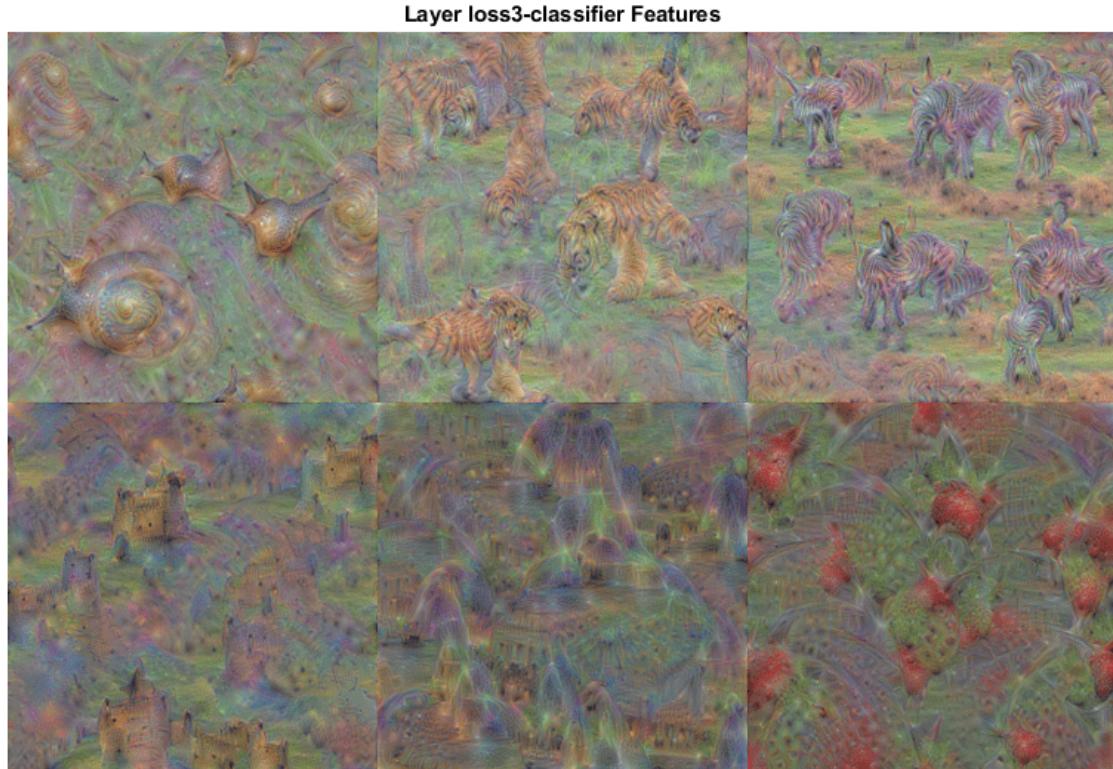
```
net.Layers(end).Classes(channels)
```

```
ans = 6×1 categorical
snail
```

```
tiger
zebra
castle
fountain
strawberry
```

生成强烈激活这些类的详细图像。在调用 `deepDreamImage` 时将 `'NumIterations'` 设置为 100，以生成更详细的图像。从全连接层生成的图像对应于图像类。

```
I = deepDreamImage(net,name,channels, ...
    'Verbose',false, ...
    'NumIterations',100, ...
    'PyramidLevels',2);
figure
I = imtile(I,'ThumbnailSize',[250 250]);
imshow(I)
name = net.Layers(layer).Name;
title(['Layer ',name,' Features'])
```



生成的图像强烈激活所选类。为“zebra”类生成的图像包含明显的斑马条纹，而为“castle”类生成的图像包含塔楼和窗口。

## 另请参阅

[googlenet](#) | [deepDreamImage](#) | [occlusionSensitivity](#) | [gradCAM](#) | [imageLIME](#)

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “使用 GoogLeNet 的 Deep Dream 图像” (第 5-2 页)
- “Grad-CAM Reveals the Why Behind Deep Learning Decisions”
- “可视化卷积神经网络的激活区域” (第 5-35 页)
- “预训练的深度神经网络” (第 1-8 页)

# 管理深度学习试验

---



# 通过并行计算和云进行深度学习

---

- “使用自动并行支持功能在云中训练网络” (第 7-2 页)
- “使用 parfeval 训练多个深度学习网络” (第 7-6 页)
- “将深度学习批处理作业发送到群集” (第 7-13 页)
- “使用自动多 GPU 支持训练网络” (第 7-16 页)
- “使用 parfor 训练多个深度学习网络” (第 7-20 页)
- “将深度学习数据上传到云” (第 7-27 页)

## 使用自动并行支持功能在云中训练网络

此示例说明如何利用 MATLAB 对并行训练的自动支持来训练卷积神经网络。深度学习训练通常需要几小时或几天。借助并行计算，您可以在本地或云群集中使用多个图形处理单元 (GPU) 来加快训练速度。如果您可以使用具有多个 GPU 的计算机，则可以在数据的本地副本上完成此示例。如果要使用更多资源，则可以将深度学习训练扩展到云。要了解有关并行训练选项的详细信息，请参阅“Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud”。此示例逐步指导您如何利用 MATLAB 的自动并行支持功能在云群集中训练深度学习网络。

### 要求

您需要配置群集并将数据上传到云，才能运行该示例。在 MATLAB 中，您可以直接通过 MATLAB 桌面在云中创建群集。在主页选项卡上，在 Parallel 菜单中，选择 **Create and Manage Clusters**。在 Cluster Profile Manager 中，点击 **Create Cloud Cluster**。您也可以使用 MathWorks 云中心来创建和访问计算群集。有关详细信息，请参阅云中心快速入门。然后，将您的数据上传到 Amazon S3 存储桶并直接从 MATLAB 访问它。此示例使用已存储在 Amazon S3 中的 CIFAR-10 数据集的副本。有关说明，请参阅“将深度学习数据上传到云”（第 7-27 页）。

### 设置并行池

在群集中启动并行池，并将工作进程数设置为群集中的 GPU 数量。如果指定的工作进程数大于 GPU 数量，则其余的工作进程将处于空闲状态。此示例假设您所使用的群集设置为默认群集配置文件。在 MATLAB 主页选项卡上的 **Parallel > Select a Default Cluster** 中，检查默认群集配置文件。

```
numberOfWorkers = 8;
parpool(numberOfWorkers);
```

Starting parallel pool (parpool) using the 'MyClusterInTheCloud' profile ...
connected to 8 workers.

### 从云中加载数据集

使用 **imageDatastore** 从云中加载训练数据集和测试数据集。在本示例中，您使用存储在 Amazon S3 中的 CIFAR-10 数据集的副本。为确保工作进程能够访问云中的数据存储，请确保已正确设置 AWS 凭据的环境变量。请参阅“将深度学习数据上传到云”（第 7-27 页）。

```
imdsTrain = imageDatastore('s3://cifar10cloud/cifar10/train',...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

imdsTest = imageDatastore('s3://cifar10cloud/cifar10/test',...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

通过创建 **augmentedImageDatastore** 对象，用增强的图像数据训练网络。使用随机平移和水平翻转。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
imageSize = [32 32 3];
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augmentedImdsTrain = augmentedImageDatastore(imageSize,imdsTrain, ...
    'DataAugmentation',imageAugmenter, ...
    'OutputSizeMode','randcrop');
```

## 定义网络架构和训练选项

为 CIFAR-10 数据集定义一个网络架构。为了简化代码，使用对输入进行卷积的卷积块。池化层对空间维度进行下采样。

```
blockDepth = 4; % blockDepth controls the depth of a convolutional block
netWidth = 32; % netWidth controls the number of filters in a convolutional block

layers = [
    imageInputLayer(imageSize)

    convolutionalBlock(netWidth,blockDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(2*netWidth,blockDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(4*netWidth,blockDepth)
    averagePooling2dLayer(8)

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer
];
```

定义训练选项。通过将执行环境设置为 `parallel`，使用当前群集并行训练网络。当您使用多个 GPU 时，就增加了可用的计算资源。根据 GPU 的数量扩大小批量大小，以保持每个 GPU 上的工作负载不变。根据小批量大小缩放学习率。使用学习率调度，以随着训练的进行降低学习率。打开训练进度图可在训练过程中获得可视化的反馈数据。

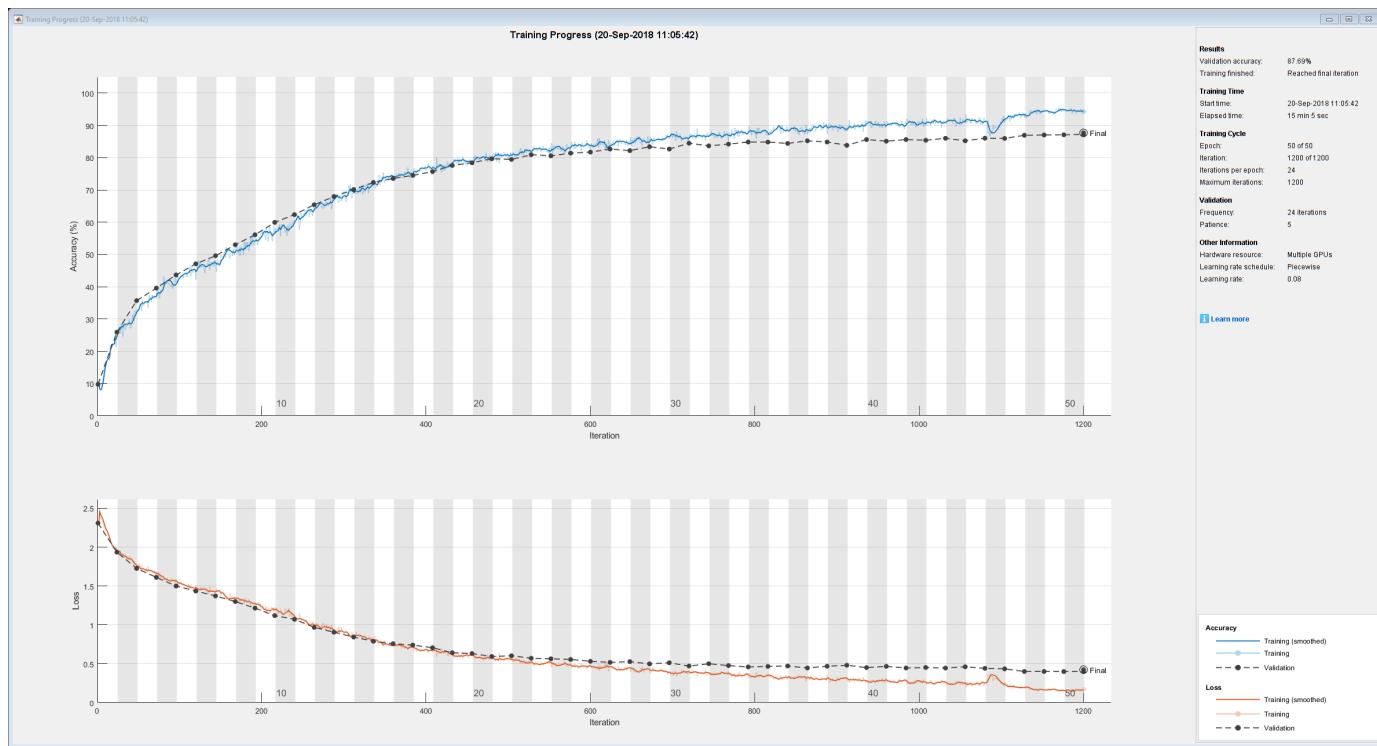
```
miniBatchSize = 256 * numberOfWorkers;
initialLearnRate = 1e-1 * miniBatchSize/256;

options = trainingOptions('sgdm', ...
    'ExecutionEnvironment','parallel', ... % Turn on automatic parallel support.
    'InitialLearnRate',initialLearnRate, ... % Set the initial learning rate.
    'MiniBatchSize',miniBatchSize, ... % Set the MiniBatchSize.
    'Verbose',false, ... % Do not send command line output.
    'Plots','training-progress', ... % Turn on the training progress plot.
    'L2Regularization',1e-10, ...
    'MaxEpochs',50, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsTest, ...
    'ValidationFrequency',floor(numel(imdsTrain.Files)/miniBatchSize), ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',45);
```

## 训练网络及其分类使用

在群集中训练网络。在训练过程中，绘图将会显示进度。

```
net = trainNetwork(augmentedImdsTrain,layers,options)
```



```
net =
SeriesNetwork with properties:

    Layers: [43×1 nnet.cnn.layer.Layer]
```

通过使用经过训练的网络对本地计算机上的测试图像进行分类，确定网络的准确度。然后将预测的标签与实际标签进行比较。

```
YPredicted = classify(net,imdsTest);
accuracy = sum(YPredicted == imdsTest.Labels)/numel(imdsTest.Labels)
```

### 定义辅助函数

定义一个函数，以便在网络架构中创建卷积块。

```
function layers = convolutionalBlock(numFilters,numConvLayers)
    layers = [
        convolution2dLayer(3,numFilters,'Padding','same')
        batchNormalizationLayer
        reluLayer
    ];
    layers = repmat(layers,numConvLayers,1);
end
```

**另请参阅**  
[trainNetwork](#) | [trainingOptions](#) | [imageDatastore](#)

## 相关示例

- “将深度学习数据上传到云” (第 7-27 页)
- “使用 parfor 训练多个深度学习网络” (第 7-20 页)

## 使用 `parfeval` 训练多个深度学习网络

此示例说明如何使用 `parfeval` 对深度学习网络的网络架构深度执行参数扫描，并在训练期间检索数据。

深度学习训练通常需要几小时或几天，搜寻良好的架构可能很困难。借助并行计算，您可以加快搜寻良好模型的速度并实现自动化。如果您可以使用具有多个图形处理单元 (GPU) 的计算机，则可以使用本地并行池在数据集的本地副本上完成此示例。如果要使用更多资源，可以将深度学习训练扩展到云。此示例说明如何使用 `parfeval` 在云群集中对网络架构的深度执行参数扫描。使用 `parfeval` 可以在后台进行训练而不会阻止 MATLAB，并提供可在结果令人满意时提前停止训练的选项。您可以修改脚本，以对其他任何参数执行参数扫描。此外，此示例还说明如何在计算期间使用 `DataQueue` 从工作进程获取反馈。

### 要求

您需要配置群集并将数据上传到云，才能运行此示例。在 MATLAB 中，您可以直接通过 MATLAB 桌面在云中创建群集。在主页选项卡上，在 Parallel 菜单中，选择 **Create and Manage Clusters**。在 Cluster Profile Manager 中，点击 **Create Cloud Cluster**。您也可以使用 MathWorks 云中心来创建和访问计算群集。有关详细信息，请参阅云中心快速入门。对于本示例，请确保在 MATLAB 主页选项卡的 **Parallel > Select a Default Cluster** 中将您的群集设置为默认群集。然后，将您的数据上传到 Amazon S3 存储桶并直接从 MATLAB 中使用它。此示例使用已存储在 Amazon S3 中的 CIFAR-10 数据集的副本。有关说明，请参阅“将深度学习数据上传到云”（第 7-27 页）。

### 从云中加载数据集

使用 `imageDatastore` 从云中加载训练数据集和测试数据集。将训练数据集拆分为训练数据集和验证数据集两部分，并保留测试数据集以测试基于参数扫描得到的最佳网络。在本示例中，您使用存储在 Amazon S3 中的 CIFAR-10 数据集的副本。为确保工作进程能够访问云中的数据存储，请确保已正确设置 AWS 凭据的环境变量。请参阅“将深度学习数据上传到云”（第 7-27 页）。

```
imds = imageDatastore('s3://cifar10cloud/cifar10/train', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

imdsTest = imageDatastore('s3://cifar10cloud/cifar10/test', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

[imdsTrain,imdsValidation] = splitEachLabel(imds,0.9);
```

通过创建 `augmentedImageDatastore` 对象，用增强的图像数据训练网络。使用随机平移和水平翻转。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
imageSize = [32 32 3];
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augmentedImdsTrain = augmentedImageDatastore(imageSize,imdsTrain, ...
    'DataAugmentation',imageAugmenter, ...
    'OutputSizeMode','randcrop');
```

### 同时训练多个网络

定义训练选项。设置小批量大小并根据小批量大小线性缩放初始学习率。设置验证频率，使 `trainNetwork` 每轮训练都验证一次网络。

```

miniBatchSize = 128;
initialLearnRate = 1e-1 * miniBatchSize/256;
validationFrequency = floor(numel(imdsTrain.Labels)/miniBatchSize);
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ... % Set the mini-batch size
    'Verbose',false, ... % Do not send command line output.
    'InitialLearnRate',initialLearnRate, ... % Set the scaled learning rate.
    'L2Regularization',1e-10, ...
    'MaxEpochs',30, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency', validationFrequency);

```

指定要对其执行参数扫描的网络架构的深度。使用 `parfeval` 同时对多个网络执行并行参数扫描训练。在扫描中使用循环来迭代不同的网络架构。在脚本末尾创建辅助函数 `createNetworkArchitecture`，它接受输入参数来控制网络的深度并为 CIFAR-10 创建一个架构。使用 `parfeval` 将由 `trainNetwork` 执行的计算量分散到群集中的工作进程。`parfeval` 将返回一个 `future` 变量，以便在计算完成后存储经过训练的网络和训练信息。

```

netDepths = 1:4;
for idx = 1:numel(netDepths)
    networksFuture(idx) = parfeval(@trainNetwork,2, ...
        augmentedImdsTrain,createNetworkArchitecture(netDepths(idx)),options);
end

```

Starting parallel pool (parpool) using the 'MyCluster' profile ...  
Connected to the parallel pool (number of workers: 4).

`parfeval` 不会阻止 MATLAB，这意味着您可以继续执行命令。在本例中，通过对 `networksFuture` 使用 `fetchOutputs` 获取经过训练的网络及其训练信息。`fetchOutputs` 函数会等待直至 `future` 变量完成执行。

```
[trainedNetworks,trainingInfo] = fetchOutputs(networksFuture);
```

通过访问 `trainingInfo` 结构体获得网络的最终验证准确度。

```
accuracies = [trainingInfo.FinalValidationAccuracy]
```

```
accuracies = 1×4
```

```
72.5600 77.2600 79.4000 78.6800
```

选择准确度最好的网络。根据测试数据集测试网络性能。

```

[~, I] = max(accuracies);
bestNetwork = trainedNetworks(I(1));
YPredicted = classify(bestNetwork,imdsTest);
accuracy = sum(YPredicted == imdsTest.Labels)/numel(imdsTest.Labels)

accuracy = 0.7840

```

计算测试数据的混淆矩阵。

```

figure('Units','normalized','Position',[0.2 0.2 0.4 0.4]);
confusionchart(imdsTest.Labels,YPredicted,'RowSummary','row-normalized','ColumnSummary','column-normalized')

```

True Class	airplane	19	19	8	14	2	11	12	23	31	
	automobile	8	922	2	1		4	2	6	55	86.1% 13.9%
bird	93	4	618	28	67	40	111	24	3	12	92.2% 7.8%
cat	36	7	41	571	55	104	116	43	9	18	61.8% 38.2%
deer	15	3	36	25	754	7	118	35	5	2	57.1% 42.9%
dog	11	3	37	102	48	649	61	73	6	10	75.4% 24.6%
frog	9	1	24	21	14	7	914	3	3	4	64.9% 35.1%
horse	28	6	11	15	51	28	22	830	1	8	91.4% 8.6%
ship	96	26	7	6	4	1	8	3	827	22	83.0% 17.0%
truck	17	62	2	7			6	2	10	894	82.7% 17.3%
											89.4% 10.6%
	73.3%	87.6%	77.5%	72.8%	74.9%	77.4%	66.7%	80.8%	92.6%	84.7%	
	26.7%	12.4%	22.5%	27.2%	25.1%	22.6%	33.3%	19.2%	7.4%	15.3%	
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck	Predicted Class

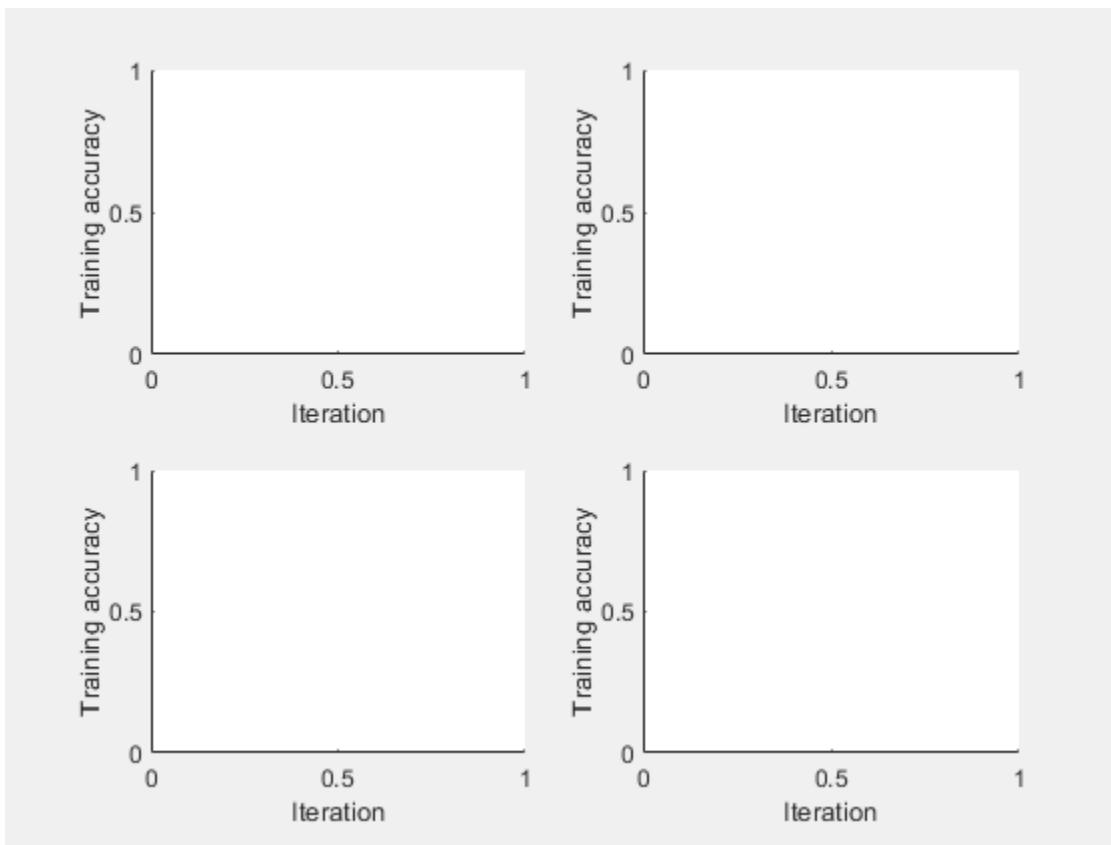
### 在训练过程中发送反馈数据

准备并初始化显示每个工作进程中的训练进度的绘图。使用 `animatedLine` 可以方便地显示变化的数据。

```

f = figure;
f.Visible = true;
for i=1:4
    subplot(2,2,i)
    xlabel('Iteration');
    ylabel('Training accuracy');
    lines(i) = animatedline;
end

```



使用 **DataQueue** 将工作进程中的训练进度数据发送给客户端，然后对数据绘图。使用 **afterEach** 在每次工作进程发送训练进度反馈时更新绘图。参数 **opts** 包含有关工作进程、训练迭代和训练准确度的信息。

```
D = parallel.pool.DataQueue;
 afterEach(D, @(opts) updatePlot(lines, opts{:}));
```

指定要对其执行参数扫描的网络架构的深度，并使用 **parfeval** 执行并行参数扫描。通过将脚本作为附加文件添加到当前池中，使工作进程能访问此脚本中的任何辅助函数。在训练选项中定义一个输出函数，用于将工作进程中的训练进度发送到客户端。训练选项依赖于工作进程的索引，这些选项必须包含在 **for** 循环中。

```
netDepths = 1:4;
addAttachedFiles(gcp,mfilename);
for idx = 1:numel(netDepths)

    miniBatchSize = 128;
    initialLearnRate = 1e-1 * miniBatchSize/256; % Scale the learning rate according to the mini-batch size.
    validationFrequency = floor(numel(imdsTrain.Labels)/miniBatchSize);

    options = trainingOptions('sgdm',...
        'OutputFcn',@(state) sendTrainingProgress(D,idx,state), ... % Set an output function to send intermediate results.
        'MiniBatchSize',miniBatchSize, ... % Set the corresponding MiniBatchSize in the sweep.
        'Verbose',false, ... % Do not send command line output.
        'InitialLearnRate',initialLearnRate, ... % Set the scaled learning rate.
        'L2Regularization',1e-10, ...
        'MaxEpochs',30, ...
        'Shuffle','every-epoch', ...
```

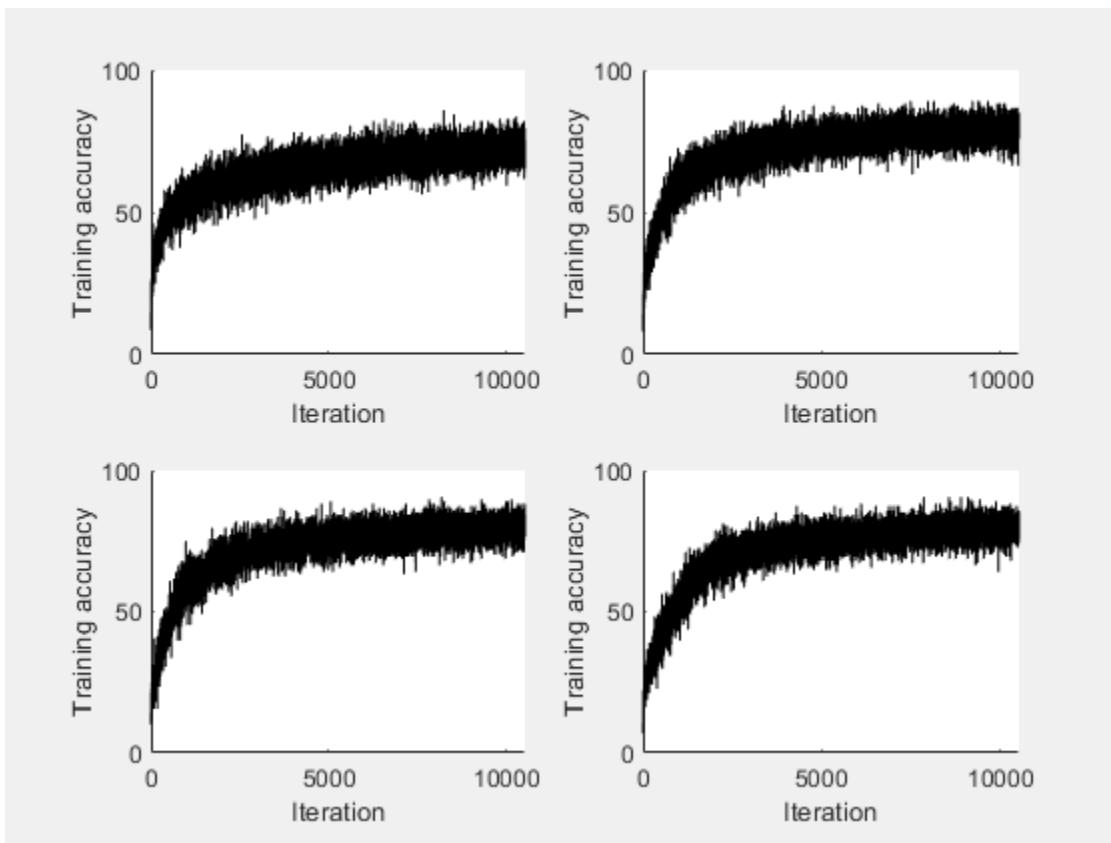
```
'ValidationData',imdsValidation, ...
'ValidationFrequency', validationFrequency);

networksFuture{idx} = parfeval(@trainNetwork,2, ...
    augmentedImdsTrain,createNetworkArchitecture(netDepths{idx}),options);
end
```

**parfeval** 对群集中的工作进程调用 **trainNetwork**。计算在后台进行，因此您可以继续在 MATLAB 中工作。如果您要停止某项 **parfeval** 计算，可以对其对应的 **future** 变量调用 **cancel**。例如，如果您观察到网络性能不佳，您可以取消其 **future** 变量的执行。如果您执行了此操作，则下一个排队的 **future** 变量将开始其计算过程。

在本例中，通过对 **future** 变量调用 **fetchOutputs**，获取经过训练的网络及其训练信息。

```
[trainedNetworks,trainingInfo] = fetchOutputs(networksFuture);
```



获取每个网络的最终验证准确度。

```
accuracies = [trainingInfo.FinalValidationAccuracy]
```

```
accuracies = 1×4
```

```
72.9200 77.4800 76.9200 77.0400
```

## 辅助函数

使用一个函数为 CIFAR-10 数据集定义网络架构，并使用输入参数调整网络深度。为了简化代码，使用对输入进行卷积的卷积块。池化层对空间维度进行下采样。

```
function layers = createNetworkArchitecture(netDepth)
imageSize = [32 32 3];
netWidth = round(16/sqrt(netDepth)); % netWidth controls the number of filters in a convolutional block

layers = [
    imageInputLayer(imageSize)

    convolutionalBlock(netWidth,netDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(2*netWidth,netDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(4*netWidth,netDepth)
    averagePooling2dLayer(8)

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer
];
end
```

定义一个函数，以便在网络架构中创建卷积块。

```
function layers = convolutionalBlock(numFilters,numConvLayers)
layers = [
    convolution2dLayer(3,numFilters,'Padding','same')
    batchNormalizationLayer
    reluLayer
];
layers = repmat(layers,numConvLayers,1);
end
```

定义一个函数，以通过 DataQueue 将训练进度发送到客户端。

```
function sendTrainingProgress(D,idx,info)
if info.State == "iteration"
    send(D,{idx,info.Iteration,info.TrainingAccuracy});
end
end
```

定义一个更新函数，以在工作进程发送中间结果时更新绘图。

```
function updatePlot(lines,idx,iter,acc)
addpoints(lines(idx),iter,acc);
drawnow limitrate nocallbacks
end
```

## 另请参阅

[parfeval](#) | [afterEach](#) | [trainNetwork](#) | [trainingOptions](#) | [imageDatastore](#)

## 相关示例

- “使用自动并行支持功能在云中训练网络” (第 7-2 页)
- “使用 parfor 训练多个深度学习网络” (第 7-20 页)
- “将深度学习数据上传到云” (第 7-27 页)

# 将深度学习批处理作业发送到群集

此示例说明如何将深度学习训练批处理作业发送到群集，以便您可以在训练过程中继续工作或者关闭 MATLAB。

训练深度神经网络通常需要几个小时或几天的时间。为了高效利用时间，您可以以批处理作业的形式训练神经网络，并在得出结果后从群集中获取结果。您可以在进行计算时继续在 MATLAB 中工作，或者关闭 MATLAB，稍后使用 Job Monitor 获得结果。此示例以批量作业的形式发送“使用 `parfor` 训练多个深度学习网络”（第 7-20 页）中的并行参数扫描。在作业完成后，您可以获取经过训练的网络并比较其准确度。

## 要求

您需要配置群集并将数据上传到云，才能运行此示例。在 MATLAB 中，您可以直接通过 MATLAB 桌面在云中创建群集。在主页选项卡上，在 Parallel 菜单中，选择 **Create and Manage Clusters**。在 Cluster Profile Manager 中，点击 **Create Cloud Cluster**。您也可以使用 MathWorks 云中心来创建和访问计算群集。有关详细信息，请参阅云中心快速入门。对于本示例，请确保在 MATLAB 主页选项卡的 **Parallel > Select a Default Cluster** 中将您的群集设置为默认群集。然后，将您的数据上传到 Amazon S3 存储桶并直接从 MATLAB 中使用它。此示例使用已存储在 Amazon S3 中的 CIFAR-10 数据集的副本。有关说明，请参阅“将深度学习数据上传到云”（第 7-27 页）。

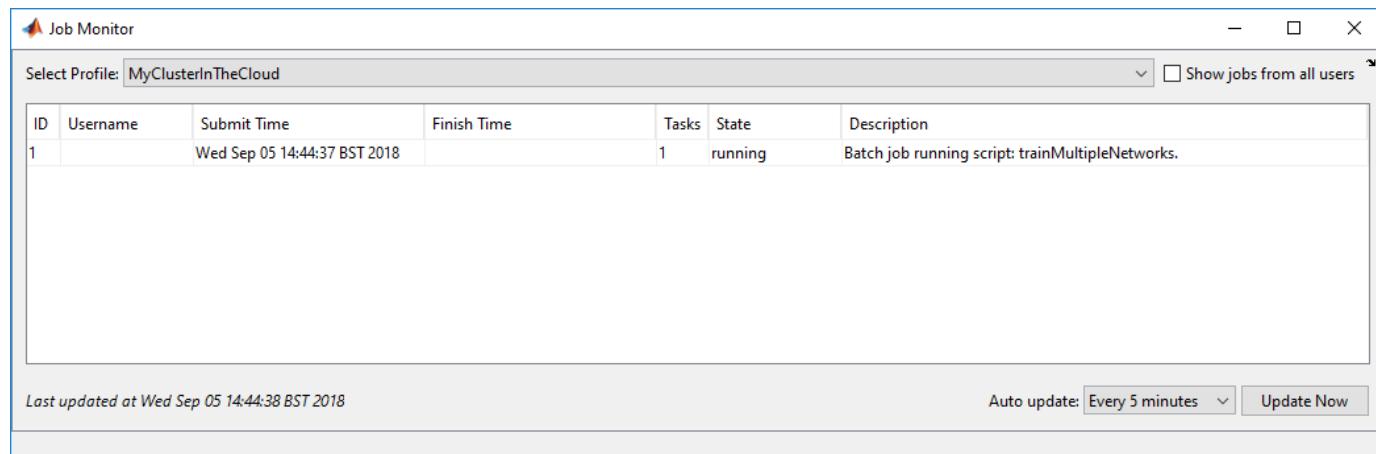
## 提交批处理作业

使用 `batch` 函数将脚本作为批处理作业发送到群集。群集会分配一个工作进程来执行脚本的内容。如果额外的工作进程有助于脚本更好地运行并行代码（例如它支持自动并行处理或 `parfor` 循环），则您需要显式请求工作进程。`batch` 为运行脚本的客户端使用一个工作进程。您可以使用 'Pool' 名称-值对组参数指定更多工作进程。

在本例中，将 `trainMultipleNetworks` 脚本发送到群集。该脚本包含“使用 `parfor` 训练多个深度学习网络”（第 7-20 页）中的并行参数扫描。由于脚本包含 `parfor` 循环，使用 Pool 名称-值对组参数指定 4 个额外的工作进程。

```
totalNumberOfWorkers = 5;
job1 = batch('trainMultipleNetworks', ...
    'Pool',totalNumberOfWorkers-1);
```

您可以通过检查 Job Monitor 来查看群集中作业的当前状态。在主页选项卡的环境部分中，选择 **Parallel > Monitor Jobs** 以打开 Job Monitor。



您可以向群集提交更多作业。如果群集因运行其他作业而不可用，则您提交的任何新作业会处于排队状态，直到群集变为可用。

### 通过编程方式获取结果

将作业提交到群集后，您可以在进行计算时继续在 MATLAB 中工作。如果代码的其余部分依赖于作业的完成，请使用 `wait` 命令阻止 MATLAB。在本例中，我们等待作业完成。

```
wait(job1);
```

作业完成后，使用 `load` 函数获取结果。在本例中，从提交的脚本中的并行参数扫描获取经过训练的网络及其准确度。

```
load(job1,'accuracies');
```

```
accuracies
```

```
accuracies = 4×1
```

```
0.8312
```

```
0.8276
```

```
0.8288
```

```
0.8258
```

```
load(job1,'trainedNetworks');
```

```
trainedNetworks
```

```
trainedNetworks = 4×1 cell array
```

```
{1×1 SeriesNetwork}
```

```
{1×1 SeriesNetwork}
```

```
{1×1 SeriesNetwork}
```

```
{1×1 SeriesNetwork}
```

要加载批处理作业中的所有变量，请使用不带参数的 `load` 函数。

```
load(job1);
```

如果关闭 MATLAB，您仍可以在计算进行过程中或计算完成后恢复群集中的作业以获取结果。在关闭 MATLAB 之前，请记下作业 ID，稍后使用 `findJob` 函数检索该作业。

要检索作业，首先使用 `parcluster` 函数为群集创建一个群集对象。然后，向 `findJob` 提供作业 ID。在本例中，作业 ID 是 1。

```
c = parcluster('MyClusterInTheCloud');
job = findJob(c,'ID',1);
```

完成后，删除作业。该作业将从 Job Monitor 中删除。

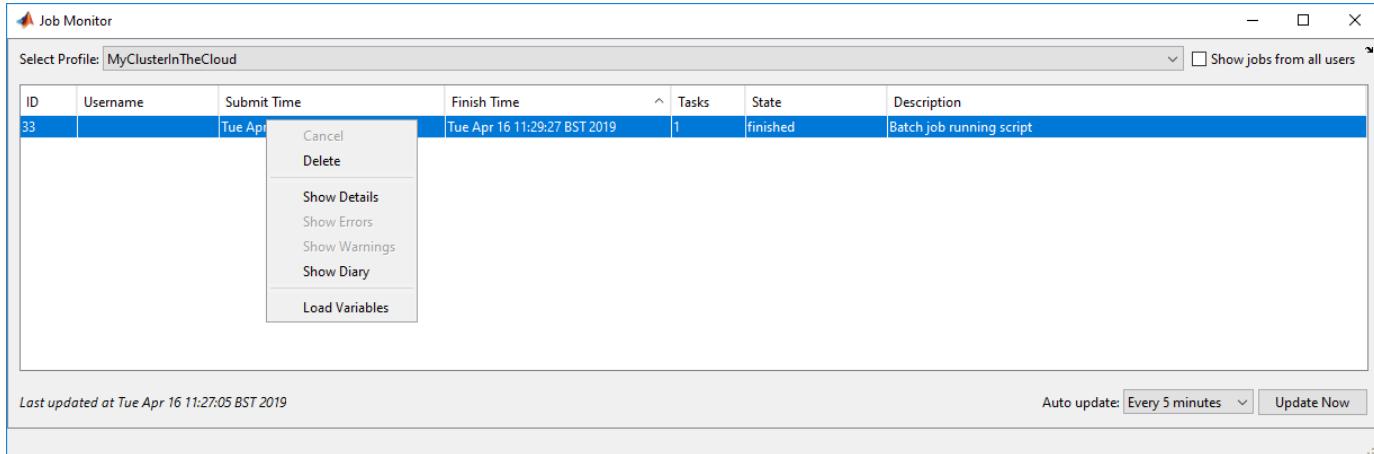
```
delete(job1);
```

### 使用 Job Monitor 获取结果

在提交批处理作业后，所有计算都在群集中进行，您可以安全地关闭 MATLAB。您可以在另一个 MATLAB 会话中使用 Job Monitor 来检查作业的状态。

在一个作业完成后，您可以从 Job Monitor 中检索结果。在主页选项卡的环境部分中，选择 **Parallel > Monitor Jobs** 以打开 Job Monitor。然后右键点击一个作业以显示上下文菜单。从该菜单中，您可以：

- 点击 **Show Details** 将作业加载到工作区中
- 点击 **Load Variables** 加载作业中的所有变量
- 在完成后，点击 **Delete** 删除作业



**另请参阅**  
batch

## 相关示例

- “使用 parfor 训练多个深度学习网络”（第 7-20 页）
- “将深度学习数据上传到云”（第 7-27 页）

## 详细信息

- “Batch Processing” (Parallel Computing Toolbox)

## 使用自动多 GPU 支持训练网络

此示例说明如何使用自动并行支持在本地计算机上使用多个 GPU 进行深度学习训练。训练深度学习网络通常需要几个小时或几天的时间。借助并行计算，您可以使用多个 GPU 加快训练速度。要了解有关并行训练选项的详细信息，请参阅“Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud”。

### 要求

在运行此示例之前，您必须将 CIFAR-10 数据集下载到本地计算机。使用以下代码将该数据集下载到您的当前目录。如果您已有 CIFAR-10 的本地副本，则可以略过本节。

```
directory = pwd;
[locationCifar10Train,locationCifar10Test] = downloadCIFARToFolders(directory);

Downloading CIFAR-10 data set...done.
Copying CIFAR-10 to folders...done.
```

### 加载数据集

使用 `imageDatastore` 对象加载训练和测试数据集。在以下代码中，确保数据存储的位置指向本地计算机中的 CIFAR-10。

```
imdsTrain = imageDatastore(locationCifar10Train, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

imdsTest = imageDatastore(locationCifar10Test, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

要使用增强的图像数据训练网络，请创建 `augmentedImageDatastore` 对象。使用随机平移和水平翻转。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
imageSize = [32 32 3];
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augmentedImdsTrain = augmentedImageDatastore(imageSize,imdsTrain, ...
    'DataAugmentation',imageAugmenter);
```

### 定义网络架构和训练选项

为 CIFAR-10 数据集定义一个网络架构。为了简化代码，使用对输入进行卷积的卷积块。池化层对空间维度进行下采样。

```
blockDepth = 4; % blockDepth controls the depth of a convolutional block.
netWidth = 32; % netWidth controls the number of filters in a convolutional block.

layers = [
    imageInputLayer(imageSize)

    convolutionalBlock(netWidth,blockDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(2*netWidth,blockDepth)
    maxPooling2dLayer(2,'Stride',2)
```

```

convolutionalBlock(4*netWidth,blockDepth)
averagePooling2dLayer(8)

fullyConnectedLayer(10)
softmaxLayer
classificationLayer
];

```

定义训练选项。通过将执行环境设置为 'multi-gpu' 使用多个 GPU 并行训练网络。当您使用多个 GPU 时，就增加了可用的计算资源。根据 GPU 的数量扩大小批量大小，以保持每个 GPU 上的工作负载不变。在此示例中，GPU 的数量是两个。根据小批量大小缩放学习率。使用学习率调度，以随着训练的进行降低学习率。打开训练进度图可在训练过程中获得可视化的反馈数据。

```

numGPUs = 2;
miniBatchSize = 256*numGPUs;
initialLearnRate = 1e-1*miniBatchSize/256;

options = trainingOptions('sgdm', ...
    'ExecutionEnvironment','multi-gpu', ... % Turn on automatic multi-gpu support.
    'InitialLearnRate',initialLearnRate, ... % Set the initial learning rate.
    'MiniBatchSize',miniBatchSize, ... % Set the MiniBatchSize.
    'Verbose',false, ... % Do not send command line output.
    'Plots','training-progress', ... % Turn on the training progress plot.
    'L2Regularization',1e-10, ...
    'MaxEpochs',60, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsTest, ...
    'ValidationFrequency',floor(numel(imdsTrain.Files)/miniBatchSize), ...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',50);

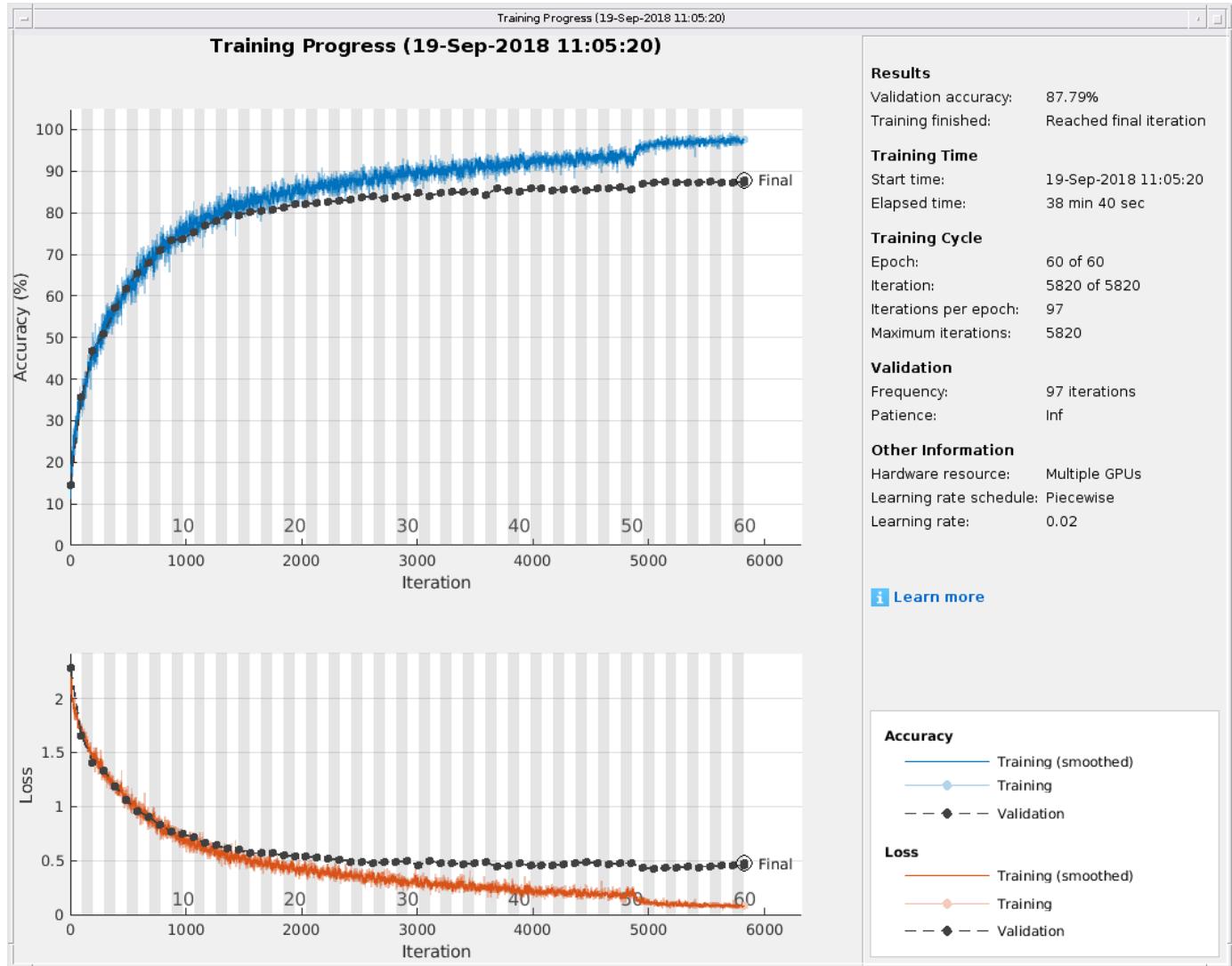
```

## 训练网络及其分类使用

训练网络。在训练过程中，绘图将会显示进度。

```
net = trainNetwork(augmentedImdsTrain,layers,options)
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 2).
```



```
net =
SeriesNetwork with properties:

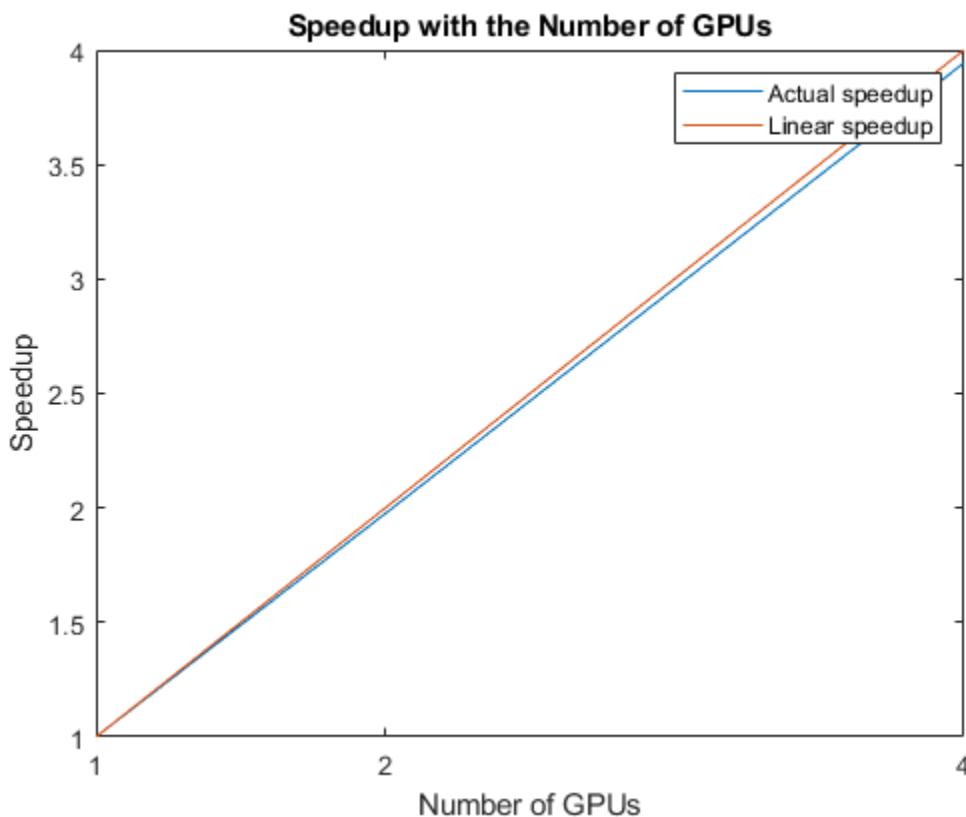
    Layers: [43×1 nnet.cnn.layer.Layer]
```

通过使用经过训练的网络对本地计算机上的测试图像进行分类，确定网络的准确度。然后将预测的标签与实际标签进行比较。

```
YPredicted = classify(net,imdsTest);
accuracy = sum(YPredicted == imdsTest.Labels)/numel(imdsTest.Labels)

accuracy = 0.8779
```

自动多 GPU 支持可以利用多个 GPU 来加快网络训练速度。下图显示在具有四个 NVIDIA© TITAN Xp GPU 的 Linux 计算机上，GPU 的数量对整个训练时间的加速情况。



### 定义辅助函数

定义一个函数，以便在网络架构中创建卷积块。

```
function layers = convolutionalBlock(numFilters,numConvLayers)
    layers = [
        convolution2dLayer(3,numFilters,'Padding','same')
        batchNormalizationLayer
        reluLayer];

    layers = repmat(layers,numConvLayers,1);
end
```

### 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [imageDatastore](#)

### 相关示例

- “Scale Up Deep Learning in Parallel, on GPUs, and in the Cloud”

## 使用 `parfor` 训练多个深度学习网络

此示例说明如何使用 `parfor` 循环对训练选项执行参数扫描。

深度学习训练通常需要几小时或几天，搜寻良好的训练选项可能很困难。借助并行计算，您可以加快搜寻良好模型的速度并实现自动化。如果您可以使用具有多个图形处理单元 (GPU) 的计算机，则可以使用本地 `parpool` 在数据集的本地副本上完成此示例。如果要使用更多资源，可以将深度学习训练扩展到云。此示例说明如何使用 `parfor` 循环在云群集中对训练选项 `MiniBatchSize` 执行参数扫描。您可以修改此脚本，以对其他任何训练选项执行参数扫描。此外，此示例还说明如何在计算期间使用 `DataQueue` 从工作进程获取反馈。您还可以将脚本作为批处理作业发送给群集，这样您可以继续工作或者关闭 MATLAB，稍后再获取结果。有关详细信息，请参阅“将深度学习批处理作业发送到群集”（第 7-13 页）。

### 要求

您需要配置群集并将数据上传到云，才能运行此示例。在 MATLAB 中，您可以直接通过 MATLAB 桌面在云中创建群集。在 **主页** 选项卡上，在 **Parallel** 菜单中，选择 **Create and Manage Clusters**。在 Cluster Profile Manager 中，点击 **Create Cloud Cluster**。您也可以使用 MathWorks 云中心来创建和访问计算群集。有关详细信息，请参阅云中心快速入门。对于本示例，请确保在 MATLAB **主页** 选项卡的 **Parallel > Select a Default Cluster** 中将您的群集设置为默认群集。然后，将您的数据上传到 Amazon S3 存储桶并直接从 MATLAB 中使用它。此示例使用已存储在 Amazon S3 中的 CIFAR-10 数据集的副本。有关说明，请参阅“将深度学习数据上传到云”（第 7-27 页）。

### 从云中加载数据集

使用  `imageDatastore` 从云中加载训练数据集和测试数据集。将训练数据集拆分为训练数据集和验证数据集两部分，并保留测试数据集以测试基于参数扫描得到的最佳网络。在本示例中，您使用存储在 Amazon S3 中的 CIFAR-10 数据集的副本。为确保工作进程能够访问云中的数据存储，请确保已正确设置 AWS 凭据的环境变量。请参阅“将深度学习数据上传到云”（第 7-27 页）。

```
imds = imageDatastore('s3://cifar10cloud/cifar10/train', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

imdsTest = imageDatastore('s3://cifar10cloud/cifar10/test', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');

[imdsTrain,imdsValidation] = splitEachLabel(imds,0.9);
```

通过创建  `augmentedImageDatastore` 对象，用增强的图像数据训练网络。使用随机平移和水平翻转。数据增强有助于防止网络过拟合和记忆训练图像的具体细节。

```
imageSize = [32 32 3];
pixelRange = [-4 4];
imageAugmenter = imageDataAugmenter( ...
    'RandXReflection',true, ...
    'RandXTranslation',pixelRange, ...
    'RandYTranslation',pixelRange);
augmentedImdsTrain = augmentedImageDatastore(imageSize,imdsTrain, ...
    'DataAugmentation',imageAugmenter, ...
    'OutputSizeMode','randcrop');
```

### 定义网络架构

为 CIFAR-10 数据集定义一个网络架构。为了简化代码，使用对输入进行卷积的卷积块。池化层对空间维度进行下采样。

```

imageSize = [32 32 3];
netDepth = 2; % netDepth controls the depth of a convolutional block
netWidth = 16; % netWidth controls the number of filters in a convolutional block

layers = [
    imageInputLayer(imageSize)

    convolutionalBlock(netWidth,netDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(2*netWidth,netDepth)
    maxPooling2dLayer(2,'Stride',2)
    convolutionalBlock(4*netWidth,netDepth)
    averagePooling2dLayer(8)

    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer
];

```

### 同时训练多个网络

指定要对其执行参数扫描的小批量大小。为得到的网络和准确度分配变量。

```

miniBatchSizes = [64 128 256 512];
numMiniBatchSizes = numel(miniBatchSizes);
trainedNetworks = cell(numMiniBatchSizes,1);
accuracies = zeros(numMiniBatchSizes,1);

```

在 **parfor** 循环内基于不同小批量大小对多个网络执行并行参数扫描训练。群集中的工作进程同时训练这些网络，并在训练完成后将训练过的网络和准确度发送回去。如果您要检查训练是否正常工作，请在训练选项中将 **Verbose** 设置为 **true**。请注意，各工作进程独立进行计算，因此命令行输出与迭代的顺序不同。

```

parfor idx = 1:numMiniBatchSizes

miniBatchSize = miniBatchSizes(idx);
initialLearnRate = 1e-1 * miniBatchSize/256; % Scale the learning rate according to the mini-batch size.

% Define the training options. Set the mini-batch size.
options = trainingOptions('sgdm', ...
    'MiniBatchSize',miniBatchSize, ... % Set the corresponding MiniBatchSize in the sweep.
    'Verbose',false, ... % Do not send command line output.
    'InitialLearnRate',initialLearnRate, ... % Set the scaled learning rate.
    'L2Regularization',1e-10, ...
    'MaxEpochs',30, ...
    'Shuffle','every-epoch', ...
    'ValidationData',imdsValidation, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropFactor',0.1, ...
    'LearnRateDropPeriod',25);

% Train the network in a worker in the cluster.
net = trainNetwork(augmentedImdsTrain,layers,options);

% To obtain the accuracy of this network, use the trained network to
% classify the validation images on the worker and compare the predicted labels to the
% actual labels.
YPredicted = classify(net,imdsValidation);

```

```

accuracies(idx) = sum(YPredicted == imdsValidation.Labels)/numel(imdsValidation.Labels);

% Send the trained network back to the client.
trainedNetworks{idx} = net;
end

```

Starting parallel pool (parpool) using the 'MyClusterInTheCloud' profile ...  
Connected to the parallel pool (number of workers: 4).

在 `parfor` 完成后, `trainedNetworks` 将包含工作进程训练得出的网络。显示经过训练的网络及其准确度。

#### `trainedNetworks`

```

trainedNetworks = 4×1 cell array
{1×1 SeriesNetwork}
{1×1 SeriesNetwork}
{1×1 SeriesNetwork}
{1×1 SeriesNetwork}

```

#### `accuracies`

```

accuracies = 4×1

0.8188
0.8232
0.8162
0.8050

```

选择准确度最好的网络。根据测试数据集测试网络性能。

```

[~, I] = max(accuracies);
bestNetwork = trainedNetworks{I(1)};
YPredicted = classify(bestNetwork,imdsTest);
accuracy = sum(YPredicted == imdsTest.Labels)/numel(imdsTest.Labels)

accuracy = 0.8173

```

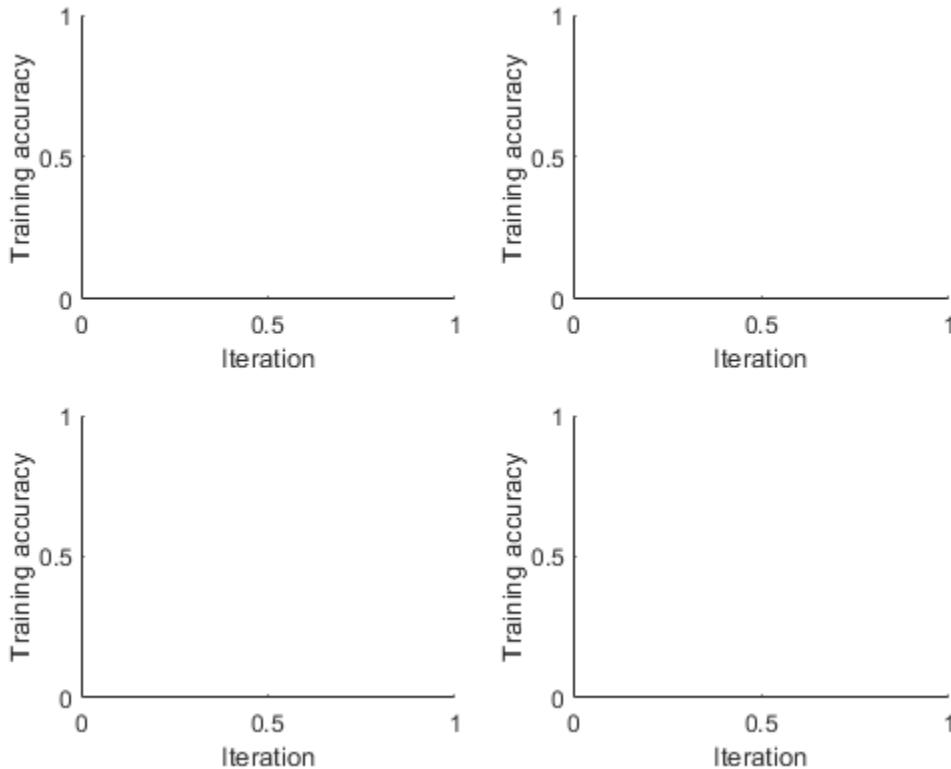
#### 在训练过程中发送反馈数据

准备并初始化显示每个工作进程中的训练进度的绘图。使用 `animatedLine` 可以方便地显示变化的数据。

```

f = figure;
f.Visible = true;
for i=1:4
    subplot(2,2,i)
    xlabel('Iteration');
    ylabel('Training accuracy');
    lines(i) = animatedline;
end

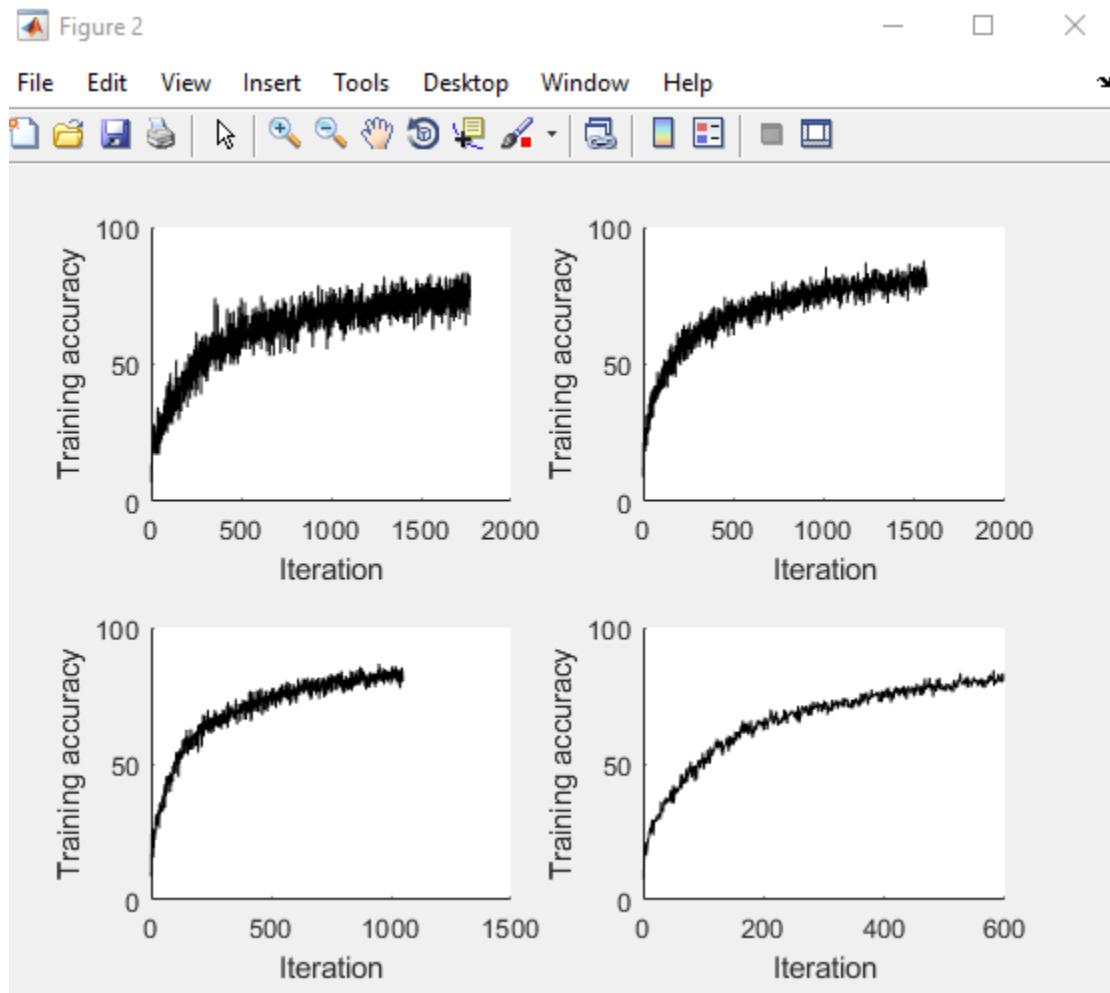
```



使用 **DataQueue** 将工作进程中的训练进度数据发送给客户端，然后对数据绘图。使用 **afterEach** 在每次工作进程发送训练进度反馈时更新绘图。参数 **opts** 包含有关工作进程、训练迭代和训练准确度的信息。

```
D = parallel.pool.DataQueue;
afterEach(D, @(opts) updatePlot(lines, opts{:}));
```

在 **parfor** 循环内基于不同小批量大小对多个网络执行并行参数扫描训练。请注意，在训练选项中使用 **OutputFcn** 可在每次迭代时将训练进度发送到客户端。下图显示了执行以下代码时四个不同工作进程的训练进度。



```

parfor idx = 1:numel(miniBatchSizes)

    miniBatchSize = miniBatchSizes(idx);
    initialLearnRate = 1e-1 * miniBatchSize/256; % Scale the learning rate according to the miniBatchSize.

    % Define the training options. Set an output function to send data back
    % to the client each iteration.
    options = trainingOptions('sgdm', ...
        'MiniBatchSize',miniBatchSize, ... % Set the corresponding MiniBatchSize in the sweep.
        'Verbose',false, ... % Do not send command line output.
        'InitialLearnRate',initialLearnRate, ... % Set the scaled learning rate.
        'OutputFcn',@(state) sendTrainingProgress(D, idx, state), ... % Set an output function to send intermediate re...
        'L2Regularization',1e-10, ...
        'MaxEpochs',30, ...
        'Shuffle','every-epoch', ...
        'ValidationData',imdsValidation, ...
        'LearnRateSchedule','piecewise', ...
        'LearnRateDropFactor',0.1, ...
        'LearnRateDropPeriod',25);

    % Train the network in a worker in the cluster. The workers send
    % training progress information during training to the client.

```

```

net = trainNetwork(augmentedImdsTrain,layers,options);

% To obtain the accuracy of this network, use the trained network to
% classify the validation images on the worker and compare the predicted labels to the
% actual labels.
YPredicted = classify(net,imdsValidation);
accuracies(idx) = sum(YPredicted == imdsValidation.Labels)/numel(imdsValidation.Labels);

% Send the trained network back to the client.
trainedNetworks{idx} = net;
end

```

Analyzing and transferring files to the workers ...done.

在 `parfor` 完成后，`trainedNetworks` 将包含工作进程训练得出的网络。显示经过训练的网络及其准确度。

#### **trainedNetworks**

```

trainedNetworks = 4×1 cell array
    {1×1 SeriesNetwork}
    {1×1 SeriesNetwork}
    {1×1 SeriesNetwork}
    {1×1 SeriesNetwork}

```

#### **accuracies**

```

accuracies = 4×1

0.8214
0.8172
0.8132
0.8084

```

选择准确度最好的网络。根据测试数据集测试网络性能。

```

[~, I] = max(accuracies);
bestNetwork = trainedNetworks{I(1)};
YPredicted = classify(bestNetwork,imdsTest);
accuracy = sum(YPredicted == imdsTest.Labels)/numel(imdsTest.Labels)

accuracy = 0.8187

```

#### **辅助函数**

定义一个函数，以便在网络架构中创建卷积块。

```

function layers = convolutionalBlock(numFilters,numConvLayers)
layers = [
    convolution2dLayer(3,numFilters,'Padding','same')
    batchNormalizationLayer
    reluLayer
];

layers = repmat(layers,numConvLayers,1);
end

```

定义一个函数，以通过 `DataQueue` 将训练进度发送到客户端。

```
function sendTrainingProgress(D,idx,info)
if info.State == "iteration"
    send(D,{idx,info.Iteration,info.TrainingAccuracy});
end
end
```

定义一个更新函数，以在工作进程发送中间结果时更新绘图。

```
function updatePlot(lines,idx,iter,acc)
addpoints(lines(idx),iter,acc);
drawnow limitrate nocallbacks
end
```

### 另请参阅

[trainNetwork](#) | [parallel.pool.DataQueue](#) | [imageDatastore](#)

### 相关示例

- “将深度学习数据上传到云” (第 7-27 页)
- “将深度学习批处理作业发送到群集” (第 7-13 页)

### 详细信息

- “Parallel for-Loops (parfor)” (Parallel Computing Toolbox)

# 将深度学习数据上传到云

此示例说明如何将数据上传到 Amazon S3 存储桶。

您需要将数据上传到云，才能在云中执行深度学习训练。此示例说明如何将 CIFAR-10 数据集下载到计算机上，然后将数据上传到 Amazon S3 存储桶，便于以后在 MATLAB 中使用。CIFAR-10 数据集是一个带标签的图像数据集，常用于对图像分类算法进行基准测试。在运行此示例之前，您需要 Amazon Web Services (AWS) 帐户的访问权限。将数据集上传到 Amazon S3 后，您可以尝试“通过并行计算和云进行深度学习”中的任何示例。

## 将 CIFAR-10 下载到本地计算机

指定用于保存下载的数据集的本地目录。以下代码在当前目录中创建一个文件夹，其中包含数据集中的所有图像。

```
directory = pwd;
[trainDirectory,testDirectory] = downloadCIFARToFolders(directory);
```

Downloading CIFAR-10 data set...done.  
Copying CIFAR-10 to folders...done.

## 将本地数据集上传到 Amazon S3 存储桶

要在云中处理数据，您可以将数据上传到 Amazon S3，然后使用数据存储从群集中的工作进程访问 S3 中的数据。以下步骤说明如何将 CIFAR-10 数据集从本地计算机上传到 Amazon S3 存储桶。

- 1.为提高与 Amazon S3 之间传输文件的效率，请从 <https://aws.amazon.com/cli/> 下载并安装 AWS 命令行界面工具。
- 2.将您的 AWS 访问密钥 ID、秘密访问密钥和存储桶区域指定为系统环境变量。请与 AWS 帐户管理员联系来获取密钥。

例如，在 Linux、macOS 或 Unix 上，指定以下变量：

```
export AWS_ACCESS_KEY_ID="YOUR_AWS_ACCESS_KEY_ID"
export AWS_SECRET_ACCESS_KEY="YOUR_AWS_SECRET_ACCESS_KEY"
export AWS_DEFAULT_REGION="us-east-1"
```

在 Windows 上，指定以下变量：

```
set AWS_ACCESS_KEY_ID="YOUR_AWS_ACCESS_KEY_ID"
set AWS_SECRET_ACCESS_KEY="YOUR_AWS_SECRET_ACCESS_KEY"
set AWS_DEFAULT_REGION="us-east-1"
```

要永久指定这些环境变量，请在您的用户或系统环境中设置它们。

- 3.使用 AWS S3 网页或类似以下的命令为您的数据创建存储桶：

```
aws s3 mb s3://mynewbucket
```

- 4.使用类似以下的命令上传您的数据：

```
aws s3 cp mylocaldatapath s3://mynewbucket --recursive
```

例如：

```
aws s3 cp path/to/CIFAR10/in/the/local/machine s3://MyExampleCloudData/cifar10/ --recursive
```

5. 通过在 MATLAB 中完成以下步骤，将您的 AWS 凭据复制到群集工作进程：

- a. 在 **主页** 选项卡上的 **环境** 部分中，选择 **Parallel > Create and Manage Clusters**。
- b. 在 Cluster Profile Manager 的 **Cluster Profile** 窗格中，选择您的云群集配置文件。
- c. 在 **Properties** 选项卡中，选择 **EnvironmentVariables** 属性，根据需要滚动以查找该属性。
- d. 在窗口右下角，点击 **Edit**。
- e. 点击 **EnvironmentVariables** 右侧的框，然后键入以下三个变量，每个变量各占一行：  
`AWS_ACCESS_KEY_ID`、`AWS_SECRET_ACCESS_KEY` 和 `AWS_DEFAULT_REGION`。
- f. 在窗口右下角，点击 **Done**。

有关如何创建云群集的信息，请参阅“Create Cloud Cluster”（Parallel Computing Toolbox）。

### 在 MATLAB 中使用数据集

将数据存储到 Amazon S3 后，可以使用数据存储从群集工作进程访问这些数据。只需创建一个指向 S3 存储桶 URL 的数据存储。以下示例代码显示如何使用 `imageDatastore` 访问 S3 存储桶。将 '`s3://MyExampleCloudData/cifar10/train`' 替换为您的 S3 存储桶的 URL。

```
imds = imageDatastore('s3://MyExampleCloudData/cifar10/train', ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

借助现在存储在 Amazon S3 中的 CIFAR-10 数据集，您可以尝试“通过并行计算和云进行深度学习”中的任一示例，这些示例说明了如何在不同用例中使用 CIFAR-10。

### 另请参阅

`imageDatastore`

### 相关示例

- “使用 `parfor` 训练多个深度学习网络”（第 7-20 页）

# 计算机视觉示例

---

## 使用 YOLO v2 深度学习进行目标检测

此示例说明如何训练 you only look once (YOLO) v2 目标检测器。

深度学习是一种功能强大的机器学习方法，可用于训练稳健的目标检测器。目标检测有多种方法，包括 Faster R-CNN 和 you only look once (YOLO) v2。此示例使用 `trainYOLOv2ObjectDetector` 函数训练 YOLO v2 车辆检测器。有关详细信息，请参阅 “Getting Started with YOLO v2” (Computer Vision Toolbox)。

### 下载预训练的检测器

下载预训练的检测器，避免在训练上花费时间。如果要训练检测器，请将 `doTraining` 变量设置为 `true`。

```
doTraining = false;
if ~doTraining && ~exist('yolov2ResNet50VehicleExample_19b.mat','file')
    disp('Downloading pretrained detector (98 MB)...');
    pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/yolov2ResNet50VehicleExample_19b.mat';
    websave('yolov2ResNet50VehicleExample_19b.mat',pretrainedURL);
end
```

### 加载数据集

此示例使用包含 295 个图像的小型车辆数据集。其中许多图像来自加州理工学院的 Caltech Cars 1999 和 2001 数据集，可在加州理工学院计算视觉网站上获得，该网站由 Pietro Perona 创建并经许可使用。每个图像包含一个或两个带标签的车辆实例。小型数据集适用于探查 YOLO v2 训练过程，但在实践中，需要更多带标签的图像来训练稳健的检测器。解压缩车辆图像并加载车辆真实值数据。

```
unzip vehicleDatasetImages.zip
data = load('vehicleDatasetGroundTruth.mat');
vehicleDataset = data.vehicleDataset;
```

车辆数据存储在一个包含两列的表中，其中第一列包含图像文件路径，第二列包含车辆边界框。

```
% Display first few rows of the data set.
vehicleDataset(1:4,:)

ans=4×2 table
  imageFilename        vehicle
  _____
  {'vehicleImages/image_00001.jpg'}  {1×4 double}
  {'vehicleImages/image_00002.jpg'}  {1×4 double}
  {'vehicleImages/image_00003.jpg'}  {1×4 double}
  {'vehicleImages/image_00004.jpg'}  {1×4 double}
```

```
% Add the fullpath to the local vehicle data folder.
vehicleDataset.imageFilename = fullfile(pwd,vehicleDataset.imageFilename);
```

将数据集分成训练集、验证集和测试集。选择 60% 的数据用于训练，10% 用于验证，其余用于测试经过训练的检测器。

```
rng(0);
shuffledIndices = randperm(height(vehicleDataset));
idx = floor(0.6 * length(shuffledIndices));
```

```

trainingIdx = 1:idx;
trainingDataTbl = vehicleDataset(shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.1 * length(shuffledIndices) );
validationDataTbl = vehicleDataset(shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);
testDataTbl = vehicleDataset(shuffledIndices(testIdx),:);

```

使用 `imageDatastore` 和 `boxLabelDatastore` 创建数据存储，以便在训练和评估期间加载图像和标签数据。

```

imdsTrain = imageDatastore(trainingDataTbl{,:'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:, 'vehicle'));

imdsValidation = imageDatastore(validationDataTbl{,:'imageFilename'});
bldsValidation = boxLabelDatastore(validationDataTbl(:, 'vehicle'));

imdsTest = imageDatastore(testDataTbl{,:'imageFilename'});
bldsTest = boxLabelDatastore(testDataTbl(:, 'vehicle'));

```

组合图像和边界框标签数据存储。

```

trainingData = combine(imdsTrain,bldsTrain);
validationData = combine(imdsValidation,bldsValidation);
testData = combine(imdsTest,bldsTest);

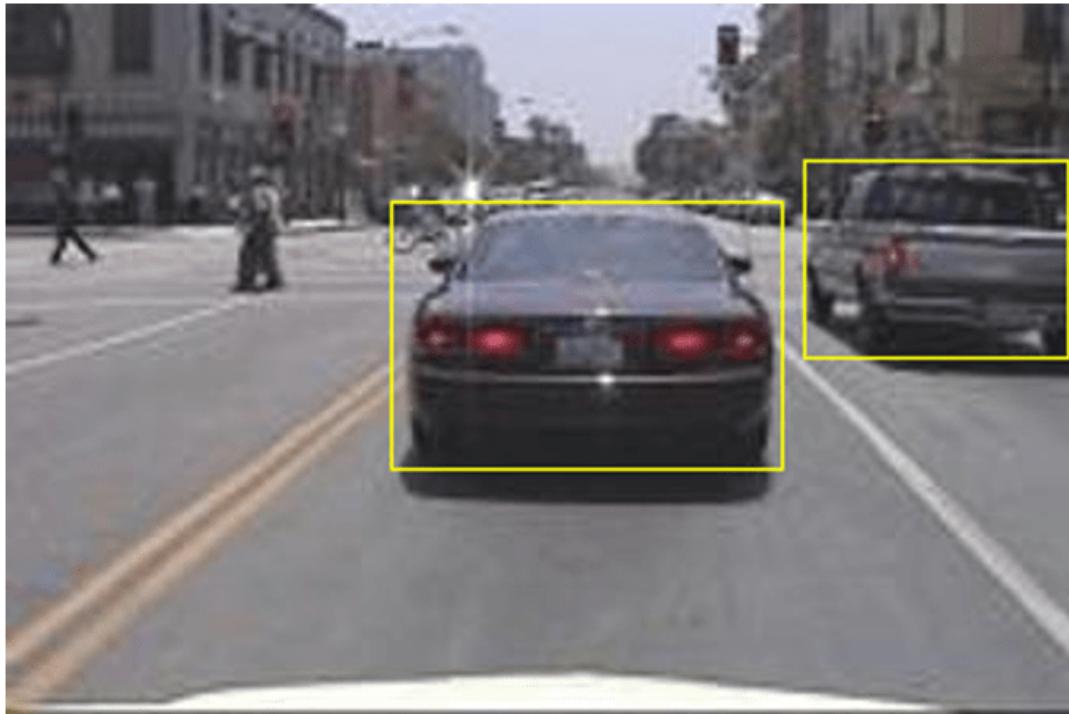
```

显示其中一个训练图像和边界框标签。

```

data = read(trainingData);
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)

```



### 创建 YOLO v2 目标检测网络

YOLO v2 目标检测网络由两个子网络组成。一个特征提取网络，后跟一个检测网络。特征提取网络通常是一个预训练的 CNN（有关详细信息，请参阅“[预训练的深度神经网络](#)”（第 1-8 页））。此示例使用 ResNet-50 进行特征提取。根据应用要求，也可以使用其他预训练网络，如 MobileNet v2 或 ResNet-18。与特征提取网络相比，检测子网络是小型 CNN，它由几个卷积层和特定于 YOLO v2 的层组成。

使用 `yolov2Layers` (Computer Vision Toolbox) 函数自动根据预训练的 ResNet-50 特征提取网络创建 YOLO v2 目标检测网络。`yolov2Layers` 要求您指定几个用于参数化 YOLO v2 网络的输入：

- 网络输入大小
- 锚框
- 特征提取网络

首先，指定网络输入大小和类的数量。选择网络输入大小时，请考虑网络本身所需的最低大小、训练图像的大小以及基于所选大小处理数据所产生的计算成本。如果可行，请选择接近训练图像大小且大于网络所需输入大小的网络输入大小。为了降低运行示例的计算成本，请指定网络输入大小为 [224 224 3]，这是运行网络所需的最低大小。

```
inputSize = [224 224 3];
```

定义要检测的目标类的数量。

```
numClasses = width(vehicleDataset)-1;
```

请注意，此示例中使用的训练图像大于  $224 \times 224$ ，并且大小不同，因此您必须在训练前的预处理步骤中调整图像的大小。

接下来，使用 `estimateAnchorBoxes` (Computer Vision Toolbox) 根据训练数据中目标的大小来估计锚框。考虑到训练前会对图像大小进行调整，用来估计锚框的训练数据的大小也要调整。使用 `transform` 预处理训练数据，然后定义锚框数量并估计锚框。使用支持函数 `preprocessData` 将训练数据的大小调整为网络的输入图像大小。

```
trainingDataForEstimation = transform(trainingData,@(data)preprocessData(data,inputSize));
numAnchors = 7;
[anchorBoxes, meanIoU] = estimateAnchorBoxes(trainingDataForEstimation, numAnchors)
```

`anchorBoxes = 7×2`

```
162 136
85 80
149 123
43 32
65 63
117 105
33 27
```

`meanIoU = 0.8472`

有关选择锚框的详细信息，请参阅“Estimate Anchor Boxes From Training Data” (Computer Vision Toolbox) (Computer Vision Toolbox™) 和“Anchor Boxes for Object Detection” (Computer Vision Toolbox)。

现在，使用 `resnet50` 加载预训练的 ResNet-50 模型。

```
featureExtractionNetwork = resnet50;
```

选择 '`activation_40_relu`' 作为特征提取层，以将 '`activation_40_relu`' 后面的层替换为检测子网络。此特征提取层输出以 16 为因子的下采样特征图。该下采样量是空间分辨率和提取特征强度之间一个很好的折中，因为在网络更深层提取的特征能够对更强的图像特征进行编码，但以空间分辨率为代价。选择最佳特征提取层需要依靠经验分析。

```
featureLayer = 'activation_40_relu';
```

创建 YOLO v2 目标检测网络。

```
lgraph = yolov2Layers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);
```

您可以使用 `analyzeNetwork` 或 Deep Learning Toolbox™ 中的深度网络设计器来可视化网络。

如果需要对 YOLO v2 网络架构进行更多控制，请使用深度网络设计器手动设计 YOLO v2 检测网络。有关详细信息，请参阅“Design a YOLO v2 Detection Network” (Computer Vision Toolbox)。

## 数据增强

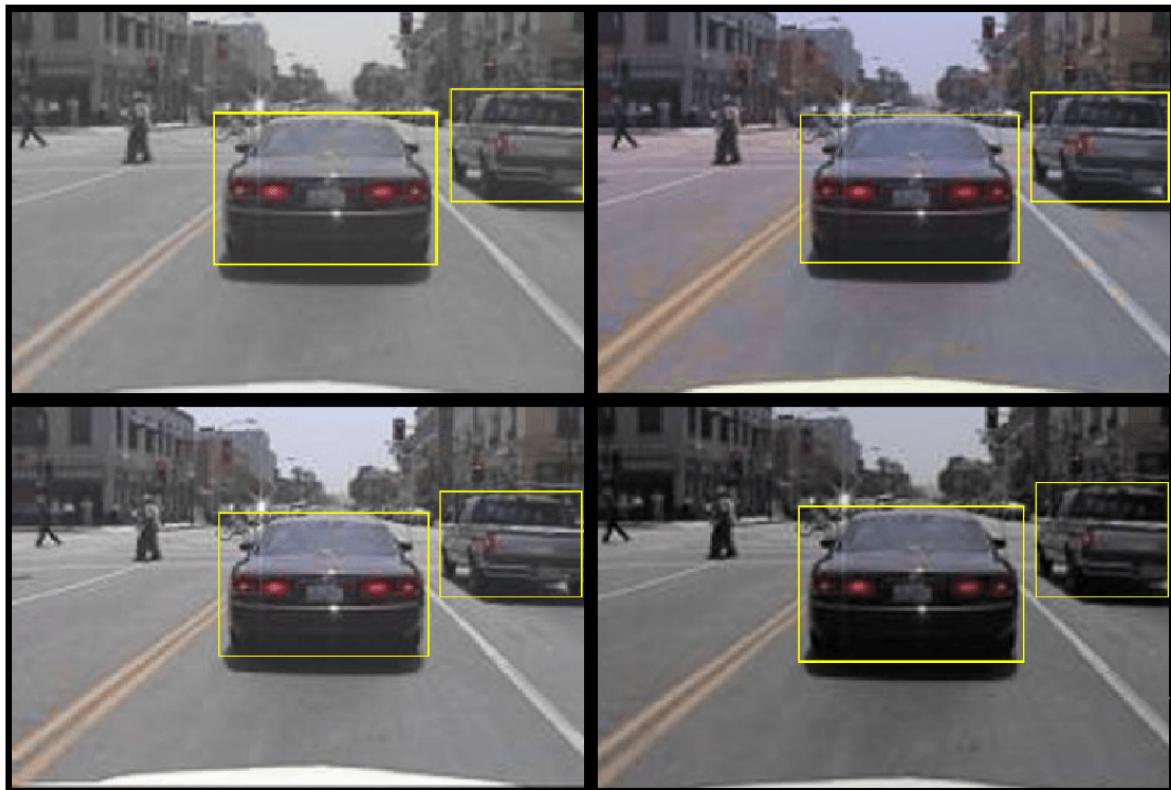
数据增强可通过在训练期间随机变换原始数据来提高网络准确度。通过使用数据增强，您可以为训练数据添加更多变化，但又不必增加已标注训练样本的数量。

使用 `transform` 通过随机水平翻转图像和相关边界框标签来增强训练数据。请注意，数据增强不适用于测试数据和验证数据。理想情况下，测试数据和验证数据应代表原始数据并且保持不变，以便进行无偏置的评估。

```
augmentedTrainingData = transform(trainingData,@augmentData);
```

多次读取同一图像，并显示增强的训练数据。

```
% Visualize the augmented images.
augmentedData = cell(4,1);
for k = 1:4
    data = read(augmentedTrainingData);
    augmentedData{k} = insertShape(data{1}, 'Rectangle', data{2});
    reset(augmentedTrainingData);
end
figure
montage(augmentedData,'BorderSize',10)
```



### 预处理训练数据

预处理增强的训练数据和验证数据以准备进行训练。

```
preprocessedTrainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));
preprocessedValidationData = transform(validationData,@(data)preprocessData(data,inputSize));
```

读取预处理的训练数据。

```
data = read(preprocessedTrainingData);
```

显示图像和边界框。

```
I = data{1};  
bbox = data{2};  
annotatedImage = insertShape(I,'Rectangle',bbox);  
annotatedImage = imresize(annotatedImage,2);  
figure  
imshow(annotatedImage)
```



## 训练 YOLO v2 目标检测器

使用 `trainingOptions` 指定网络训练选项。将 `'ValidationData'` 设置为经过预处理的验证数据。将 `'CheckpointPath'` 设置为临时位置。这样可在训练过程中保存经过部分训练的检测器。如果由于停电或系统故障等原因导致训练中断，您可以从保存的检查点继续训练。

```
options = trainingOptions('sgdm', ...  
    'MiniBatchSize',16, ...  
    'InitialLearnRate',1e-3, ...  
    'MaxEpochs',20, ...  
    'CheckpointPath',tempdir, ...  
    'ValidationData',preprocessedValidationData);
```

如果 `doTraining` 为 `true`, 使用 `trainYOLOv2ObjectDetector` (Computer Vision Toolbox) 函数训练 YOLO v2 目标检测器。否则, 加载预训练的网络。

```
if doTraining
    % Train the YOLO v2 detector.
    [detector,info] = trainYOLOv2ObjectDetector(preprocessedTrainingData,lgraph,options);
else
    % Load pretrained detector for the example.
    pretrained = load('yolov2ResNet50VehicleExample_19b.mat');
    detector = pretrained.detector;
end
```

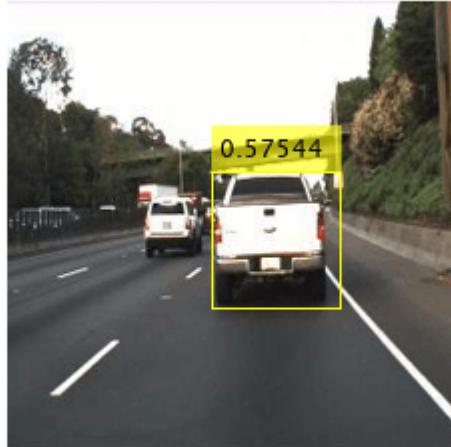
此示例在具有 12 GB 内存的 NVIDIA™ Titan X GPU 上进行了验证。如果您的 GPU 内存较少, 则可能内存不足。如果出现这种情况, 请使用 `trainingOptions` 函数降低 '`MiniBatchSize`'。使用此设置训练此网络大约需要 7 分钟。具体训练时间因您使用的硬件而异。

为了快速测试, 对一张测试图像运行检测器。确保将图像的大小调整为与训练图像相同。

```
I = imread('highway.png');
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);
```

显示结果。

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```



### 使用测试集评估检测器

基于大量图像评估经过训练的目标检测器以测量其性能。Computer Vision Toolbox™ 提供目标检测器评估函数, 用于测量常见指标, 如平均精确率 (`evaluateDetectionPrecision`) 和对数平均泄漏检率 (`evaluateDetectionMissRate`)。对于此示例, 使用平均精确率指标来评估性能。平均准确率提供单一数字, 该数字综合反映了检测器进行正确分类的能力 (精确率) 和检测器找到所有相关对象的能力 (召回率)。

将应用于训练数据的同一预处理变换应用于测试数据。请注意，数据增强不适用于测试数据。测试数据应代表原始数据并且保持不变，以便进行无偏置的评估。

```
preprocessedTestData = transform(testData,@(data)preprocessData(data,inputSize));
```

对所有测试图像运行检测器。

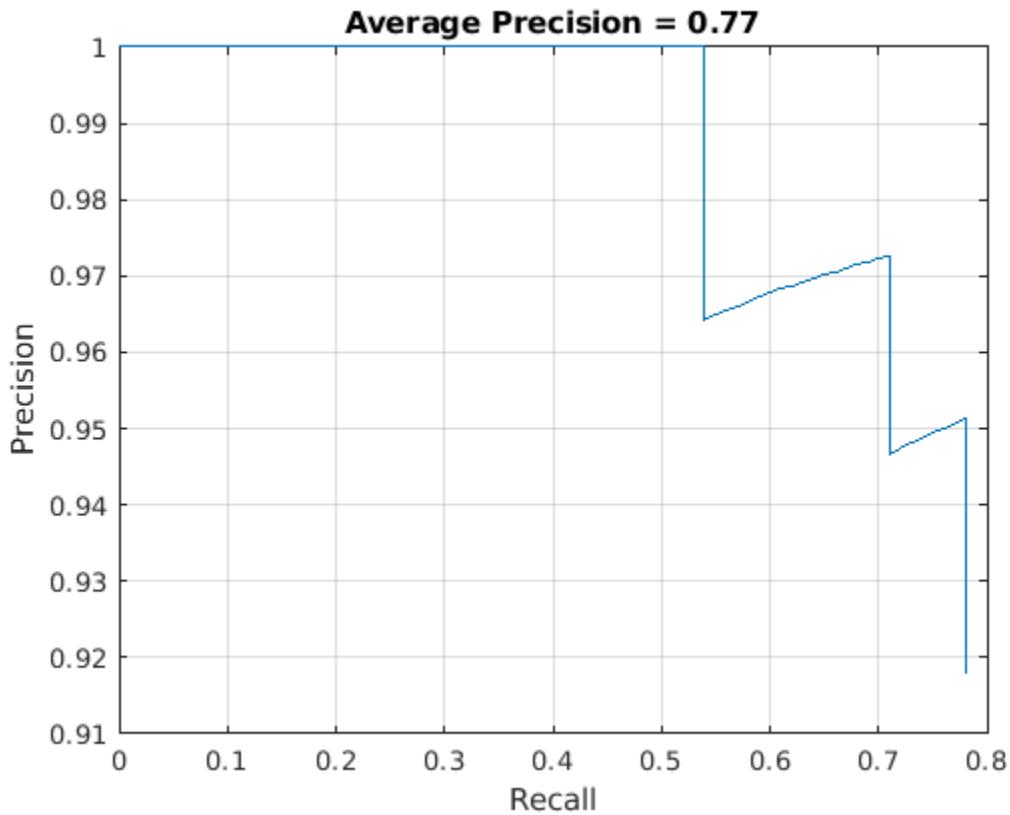
```
detectionResults = detect(detector, preprocessedTestData);
```

使用平均精确率指标评估目标检测器。

```
[ap,recall,precision] = evaluateDetectionPrecision(detectionResults, preprocessedTestData);
```

精确率/召回率 (PR) 曲线强调检测器在不同召回水平下的精确程度。理想情况下，所有召回水平的精确率均为 1。使用更多数据有助于提高平均精确率，但可能需要更多训练时间。绘制 PR 曲线。

```
figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f',ap))
```



## 代码生成

一旦检测器经过训练和评估，您就可以使用 GPU Coder™ 为 `yolov2ObjectDetector` 生成代码。有关详细信息，请参阅“Code Generation for Object Detection by Using YOLO v2” (GPU Coder) 示例。

## 支持函数

```

function B = augmentData(A)
% Apply random horizontal flipping, and random X/Y scaling. Boxes that get
% scaled outside the bounds are clipped if the overlap is above 0.25. Also,
% jitter image color.

B = cell(size(A));

I = A{1};
sz = size(I);
if numel(sz)==3 && sz(3) == 3
    I = jitterColorHSV(I, ...
        'Contrast',0.2, ...
        'Hue',0, ...
        'Saturation',0.1, ...
        'Brightness',0.2);
end

% Randomly flip and scale image.
tform = randomAffine2d('XReflection',true,'Scale',[1 1.1]);
rout = affineOutputView(sz,tform,'BoundsStyle','CenterOutput');
B{1} = imwarp(I,tform,'OutputView',rout);

% Sanitize box data, if needed.
A{2} = helperSanitizeBoxes(A{2}, sz);

% Apply same transform to boxes.
[B{2},indices] = bboxwarp(A{2},tform,rout,'OverlapThreshold',0.25);
B{3} = A{3}(indices);

% Return original data only when all boxes are removed by warping.
if isempty(indices)
    B = A;
end
end

function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to the targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));

% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2},sz);

% Resize boxes to new image size.
data{2} = bboxresize(data{2},scale);
end

```

## 参考资料

- [1] Redmon, Joseph, and Ali Farhadi. "YOLO9000:Better, Faster, Stronger." In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 6517–25.Honolulu, HI:IEEE, 2017. <https://doi.org/10.1109/CVPR.2017.690>.

# 使用深度学习进行语义分割

此示例说明如何使用深度学习训练语义分割网络。

语义分割网络对图像中的每个像素进行分类，从而生成按类分割的图像。语义分割的应用包括自动驾驶中的道路分割以及医疗诊断中的癌细胞分割。要了解详细信息，请参阅“Getting Started with Semantic Segmentation Using Deep Learning”(Computer Vision Toolbox)。

为了说明训练过程，此示例将训练 Deeplab v3+ [1]，这是一种专门用于语义图像分割的卷积神经网络(CNN)。其他类型的用于语义分割的网络包括全卷积网络(FCN)、SegNet 和 U-Net。此处所示的训练过程也可应用于这些网络。

此示例使用剑桥大学的 CamVid 数据集 [2] 进行训练。此数据集是包含驾驶时获得的街道级视图的图像集合。该数据集提供了 32 个语义类的像素级标签，包括汽车、行人和道路。

## 设置

此示例创建的 Deeplab v3+ 网络具有从预训练的 Resnet-18 网络初始化的权重。ResNet-18 是一种高效的网络，非常适合处理资源有限的应用情形。根据应用要求，也可以使用 MobileNet v2 或 ResNet-50 等其他预训练网络。有关详细信息，请参阅“预训练的深度神经网络”(第 1-8 页)。

要获得预训练的 Resnet-18，请安装 `resnet18`。安装完成后，运行以下代码以验证安装是否正确。

```
resnet18();
```

此外，还要下载 DeepLab v3+ 的预训练版本。借助预训练模型，您无需等待训练完成，即可运行整个示例。

```
pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/deeplabv3plusResnet18CamVid.mat';
pretrainedFolder = fullfile(tempdir,'pretrainedNetwork');
pretrainedNetwork = fullfile(pretrainedFolder,'deeplabv3plusResnet18CamVid.mat');
if ~exist(pretrainedNetwork,'file')
    mkdir(pretrainedFolder);
    disp('Downloading pretrained network (58 MB...)');
    websave(pretrainedNetwork,pretrainedURL);
end
```

强烈推荐使用支持 CUDA 的 NVIDIA™ GPU 来运行此示例。使用 GPU 需要 Parallel Computing Toolbox™。有关支持的计算功能的信息，请参阅“GPU Support by Release”(Parallel Computing Toolbox)。

## 下载 CamVid 数据集

从以下 URL 下载 CamVid 数据集。

```
 imageURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/files/701_StillsRaw_full.zip';
labelURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/LabeledApproved_full.zip';

outputFolder = fullfile(tempdir,'CamVid');
labelsZip = fullfile(outputFolder,'labels.zip');
imagesZip = fullfile(outputFolder,'images.zip');

if ~exist(labelsZip,'file') || ~exist(imagesZip,'file')
    mkdir(outputFolder)

    disp('Downloading 16 MB CamVid dataset labels...');
```

```
websave(labelsZip, labelURL);
unzip(labelsZip, fullfile(outputFolder,'labels'));

disp('Downloading 557 MB CamVid dataset images...!');
websave(imagesZip, imageURL);
unzip(imagesZip, fullfile(outputFolder,'images'));
end
```

注意：数据的下载时间取决于您的 Internet 连接。下载完成之前，上面使用的命令会阻止 MATLAB。您也可以使用 Web 浏览器先将数据集下载到本地磁盘。要使用从 Web 下载的文件，请将上面的 **outputFolder** 变量更改为下载的文件的位置。

### 加载 CamVid 图像

使用 `imageDatastore` 加载 CamVid 图像。通过 `imageDatastore` 可以高效加载磁盘上的大量图像。

```
imgDir = fullfile(outputFolder,'images','701_StillsRaw_full');
imds = imageDatastore(imgDir);
```

显示其中一个图像。

```
I = readimage(imds,559);
I = histeq(I);
imshow(I)
```



## 加载 CamVid 像素标注图像

使用 `pixelLabelDatastore` (Computer Vision Toolbox) 加载 CamVid 像素标签图像数据。`pixelLabelDatastore` 将像素标签数据和标签 ID 封装到类名映射中。

为了使训练变得更容易，我们将 CamVid 中的 32 个原始类分组为 11 个类。指定这些类。

```
classes = [
    "Sky"
    "Building"
    "Pole"
    "Road"
    "Pavement"
    "Tree"
    "SignSymbol"
    "Fence"
    "Car"
    "Pedestrian"
    "Bicyclist"
];

```

要将 32 个类减少为 11 个，需要将原始数据集中的多个类组合在一起。例如，“Car”是“Car”、“SUVPickupTruck”、“Truck\_Bus”、“Train”和“OtherMoving”的组合。使用支持函数 `camvidPixelLabelIDs` 返回分组的标签 ID，该函数在此示例的末尾列出。

```
labelIDs = camvidPixelLabelIDs();
```

使用类和标签 ID 创建 `pixelLabelDatastore`。

```
labelDir = fullfile(outputFolder,'labels');
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
```

读取一个像素标注图像，并将其叠加在图像上方显示。

```
C = readimage(pxds,559);
cmap = camvidColorMap;
B = labeloverlay(I,C,'ColorMap',cmap);
imshow(B)
pixelLabelColorbar(cmap,classes);
```



没有颜色叠加的区域没有像素标签，在训练过程中不被使用。

### 分析数据集统计信息

要查看 CamVid 数据集中类标签的分布，请使用 **countEachLabel** (Computer Vision Toolbox)。此函数按类标签计算像素数。

```
tbl = countEachLabel(pxds)
```

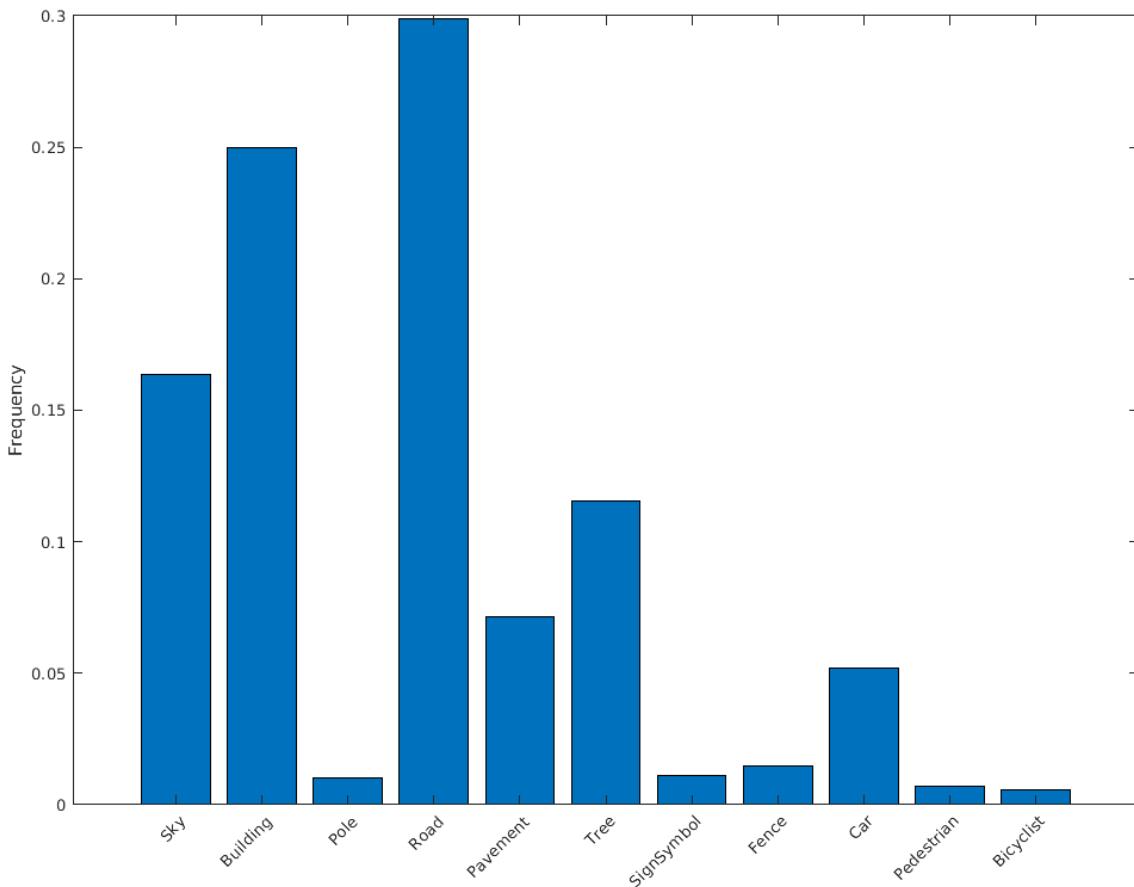
Name	PixelCount	ImagePixelCount
{'Sky' }	7.6801e+07	4.8315e+08
{'Building' }	1.1737e+08	4.8315e+08
{'Pole' }	4.7987e+06	4.8315e+08
{'Road' }	1.4054e+08	4.8453e+08
{'Pavement' }	3.3614e+07	4.7209e+08
{'Tree' }	5.4259e+07	4.479e+08
{'SignSymbol'}	5.2242e+06	4.6863e+08
{'Fence' }	6.9211e+06	2.516e+08
{'Car' }	2.4437e+07	4.8315e+08
{'Pedestrian'}	3.4029e+06	4.4444e+08

```
{'Bicyclist' } 2.5912e+06 2.6196e+08
```

按类可视化像素计数。

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);

bar(1:numel(classes),frequency)
xticks(1:numel(classes))
xticklabels(tbl.Name)
xtickangle(45)
ylabel('Frequency')
```



理想情况下，所有类都有相同数量的观测值。但是，CamVid 中的类是不平衡的，这是街景汽车数据集的常见问题。此类场景的天空、建筑物和道路像素比行人和骑车人像素多，因为天空、建筑物和道路覆盖了图像中的更多区域。如果处理不当，这种不平衡可能对学习过程不利，因为学习会偏向于占主导的类。在此示例的稍后部分，您将使用类权重来处理此问题。

CamVid 数据集中的图像大小为  $720 \times 960$ 。选择合适的图像大小，以便在拥有 12 GB 内存的 NVIDIA™ Titan X 上进行训练时，可以将一批足够大的图像放入内存中。如果您的 GPU 没有足够的内存，您可能需要将图像调整到较小的大小或减小训练批量大小。

## 准备训练集、验证集和测试集

使用数据集中 60% 的图像训练 DeepLab v3+。其余的图像平分成 20% 和 20%，分别用于验证和测试。以下代码将图像和像素标签数据随机分成训练集、验证集和测试集。

```
[imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] = partitionCamVidData(imds,pxds);
```

60/20/20 拆分将产生以下数量的训练、验证和测试图像：

```
numTrainingImages = numel(imdsTrain.Files)
numTrainingImages = 421
numValImages = numel(imdsVal.Files)
numValImages = 140
numTestingImages = numel(imdsTest.Files)
numTestingImages = 140
```

## 创建网络

使用 `deeplabv3plusLayers` 函数基于 ResNet-18 创建一个 DeepLab v3+ 网络。为您的应用选择最佳网络需要根据经验分析，而且涉及另一级别的超参数调整。例如，您可以使用不同基础网络进行试验，如 ResNet-50 或 MobileNet v2，您也可以尝试其他语义分割网络架构，如 SegNet、全卷积网络 (FCN) 或 U-Net。

```
% Specify the network image size. This is typically the same as the traing image sizes.
imageSize = [720 960 3];
% Specify the number of classes.
numClasses = numel(classes);
% Create DeepLab v3+.
lgraph = deeplabv3plusLayers(imageSize, numClasses, "resnet18");
```

## 使用类权重平衡类

如前文所示，CamVid 中的类不平衡。要改善训练，您可以使用类权重来平衡类。使用先前通过 `countEachLabel` (Computer Vision Toolbox) 计算的像素标签计数，计算具有中位数频率的类的权重。

```
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq
classWeights = 11×1
0.3182
0.2082
5.0924
0.1744
0.7103
0.4175
4.5371
1.8386
1.0000
6.6059
```

使用 `pixelClassificationLayer` (Computer Vision Toolbox) 指定类权重。

```
pxLayer = pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',classWeights);
lgraph = replaceLayer(lgraph,"classification",pxLayer);
```

### 选择训练选项

用于训练的优化算法是具有动量的随机梯度下降 (SGDM)。使用 `trainingOptions` 指定用于 SGDM 的超参数。

```
% Define validation data.
dsVal = combine(imdsVal,pxdsVal);

% Define training options.
options = trainingOptions('sgdm',...
    'LearnRateSchedule','piecewise',...
    'LearnRateDropPeriod',10,...
    'LearnRateDropFactor',0.3,...
    'Momentum',0.9, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',0.005, ...
    'ValidationData',dsVal, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',8, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2, ...
    'Plots','training-progress',...
    'ValidationPatience', 4);
```

学习率采用分段调度。学习率每 10 轮降低 0.3。这允许网络以更高的初始学习率快速学习，而一旦学习率下降，能够求得接近局部最优的解。

通过设置 `'ValidationData'` 参数，在每轮都对照验证数据对网络进行测试。`'ValidationPatience'` 设置为 4，以在验证准确度收敛时提前停止训练。这可以防止网络对训练数据集进行过拟合。

使用大小为 8 的小批量以减少训练时的内存使用量。您可以根据系统上的 GPU 内存量增大或减小此值。

此外，`'CheckpointPath'` 设置为临时位置。此名称-值对组让您能够在每轮训练结束时保存网络检查点。如果由于系统故障或停电而导致训练中断，您可以从保存的检查点处恢复训练。确保 `'CheckpointPath'` 指定的位置有足够的空间来存储网络检查点。例如，保存 100 个 Deeplab v3+ 检查点需要大约 6 GB 的磁盘空间，因为每个检查点大小为 61 MB。

### 数据增强

数据增强可通过在训练期间随机变换原始数据来提高网络准确度。通过使用数据增强，您可以为训练数据添加更多变化，而不必增加带标签的训练样本的数量。要对图像和像素标签数据应用相同的随机变换，请使用数据存储 `combine` 和 `transform`。首先，合并 `imdsTrain` 和 `pxdsTrain`。

```
dsTrain = combine(imdsTrain, pxdsTrain);
```

接下来，使用数据存储 `transform` 应用在支持函数 `augmentImageAndLabel` 中定义的所需数据增强。此处使用随机左/右翻转和随机 X/Y 平移 +/- 10 个像素来进行数据增强。

```
xTrans = [-10 10];
yTrans = [-10 10];
dsTrain = transform(dsTrain, @(data)augmentImageAndLabel(data,xTrans,yTrans));
```

请注意，数据增强不适用于测试数据和验证数据。理想情况下，测试数据和验证数据应代表原始数据并且保持不变，以便进行无偏置的评估。

### 开始训练

如果 `doTraining` 标志为 `true`，则使用 `trainNetwork` 开始训练。否则，请加载预训练的网络。

注意：训练在具有 12 GB GPU 内存的 NVIDIA™ Titan X 上进行了验证。如果您的 GPU 内存较少，则训练期间可能内存不足。如果出现这种情况，请尝试在 `trainingOptions` 中将 `'MiniBatchSize'` 设置为 1，或减少网络输入并调整训练数据的大小。训练此网络大约需要 5 个小时。根据您的 GPU 硬件情况，可能需要更长时间。

```
doTraining = false;
if doTraining
    [net, info] = trainNetwork(dsTrain,lgraph,options);
else
    data = load(pretrainedNetwork);
    net = data.net;
end
```

### 基于一个图像测试网络

要快速检验合理性，可基于一个测试图像运行经过训练的网络。

```
I = readimage(imdsTest,35);
C = semanticseg(I, net);
```

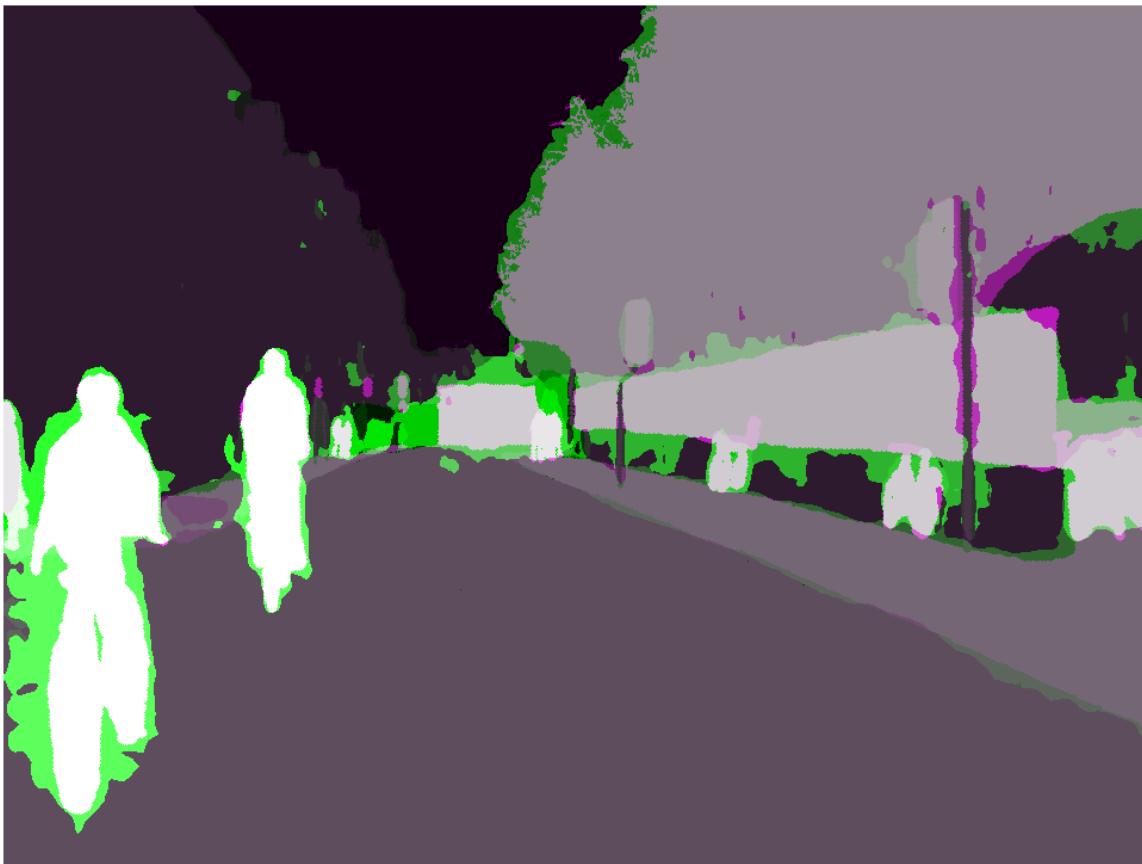
显示结果。

```
B = labeloverlay(I,C,'Colormap','cmap','Transparency',0.4);
imshow(B)
pixelLabelColorbar(cmap, classes);
```



将 C 中的结果与 pxdsTest 中存储的预期真实值进行比较。绿色和品红色区域突出显示了分割结果与预期真实值不同的区域。

```
expectedResult = readimage(pxdsTest,35);
actual = uint8(C);
expected = uint8(expectedResult);
imshowpair(actual, expected)
```



从视觉上看，语义分割结果与道路、天空和建筑物等类很好地重叠。然而，行人和汽车等较小的对象并不那么准确。可以使用交并比 (IoU) 指标（也称为 Jaccard 索引）来衡量每个类的重叠量。使用 **jaccard** (Image Processing Toolbox) 函数计算 IoU。

```
iou = jaccard(C,expectedResult);
table(classes,iou)
```

```
ans=11×2 table
    classes      iou
    _____
    "Sky"        0.91837
    "Building"   0.84479
    "Pole"        0.31203
    "Road"        0.93698
    "Pavement"   0.82838
    "Tree"        0.89636
    "SignSymbol" 0.57644
    "Fence"       0.71046
    "Car"         0.66688
    "Pedestrian" 0.48417
    "Bicyclist"   0.68431
```

IoU 指标印证了可视化的结果。道路、天空和建筑物类具有较高的 IoU 分数，而行人和汽车等类的分数较低。其他常见的分割指标包括 **dice** (Image Processing Toolbox) 和 **bfscore** (Image Processing Toolbox) 轮廓匹配分数。

### 评估经过训练的网络

要衡量多个测试图像的准确度，请对整个测试集运行 **semanticseg** (Computer Vision Toolbox)。使用大小为 4 的小批量以减少分割图像时的内存使用量。您可以根据系统上的 GPU 内存量增大或减小此值。

```
pxdsResults = semanticseg(imdsTest,net, ...
    'MiniBatchSize',4, ...
    'WriteLocation',tempdir, ...
    'Verbose',false);
```

**semanticseg** 将基于测试集的结果以 **pixelLabelDatastore** 对象的形式返回。**imdsTest** 中每个测试图像的实际像素标签数据都写入到 'WriteLocation' 参数指定的磁盘位置。使用 **evaluateSemanticSegmentation** (Computer Vision Toolbox) 基于测试集结果计算各个语义分割指标。

```
metrics = evaluateSemanticSegmentation(pxdsResults,pxdsTest,'Verbose',false);
```

**evaluateSemanticSegmentation** 返回基于整个数据集、单个类以及单个测试图像的各种指标。要查看数据集级别指标，请检查 **metrics.DataSetMetrics**。

```
metrics.DataSetMetrics
```

```
ans=1×5 table
    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    _____
    0.87695         0.85392        0.6302     0.80851       0.65051
```

数据集指标提供了网络性能的高级概览。要查看每个类对整体性能的影响，请使用 **metrics.ClassMetrics** 检查每个类的指标。

```
metrics.ClassMetrics
```

```
ans=11×3 table
    Accuracy    IoU    MeanBFScore
    _____
    Sky         0.93112   0.90209    0.8952
    Building    0.78453   0.76098    0.58511
    Pole        0.71586   0.21477    0.51439
    Road        0.93024   0.91465    0.76696
    Pavement    0.88466   0.70571    0.70919
    Tree        0.87377   0.76323    0.70875
    SignSymbol  0.79358   0.39309    0.48302
    Fence       0.81507   0.46484    0.48566
    Car         0.90956   0.76799    0.69233
    Pedestrian  0.87629   0.4366     0.60792
    Bicyclist   0.87844   0.60829    0.55089
```

尽管数据集整体性能非常高，但类指标显示，**Pedestrian**、**Bicyclist** 和 **Car** 等类表示不充分，分割效果不如 **Road**、**Sky** 和 **Building** 等类。增加包含更多表现不足类的样本的数据可能有助于改善结果。

## 支持函数

```

function labelIDs = camvidPixelLabelIDs()
% Return the label IDs corresponding to each class.
%
% The CamVid dataset has 32 classes. Group them into 11 classes following
% the original SegNet training methodology [1].
%
% The 11 classes are:
% "Sky" "Building", "Pole", "Road", "Pavement", "Tree", "SignSymbol",
% "Fence", "Car", "Pedestrian", and "Bicyclist".
%
% CamVid pixel label IDs are provided as RGB color values. Group them into
% 11 classes and return them as a cell array of M-by-3 matrices. The
% original CamVid class names are listed alongside each RGB value. Note
% that the Other/Void class are excluded below.
labelIDs = { ...
    %
    % "Sky"
    [
    128 128 128; ... % "Sky"
    ]

    % "Building"
    [
    000 128 064; ... % "Bridge"
    128 000 000; ... % "Building"
    064 192 000; ... % "Wall"
    064 000 064; ... % "Tunnel"
    192 000 128; ... % "Archway"
    ]

    % "Pole"
    [
    192 192 128; ... % "Column_Pole"
    000 000 064; ... % "TrafficCone"
    ]

    % "Road"
    [
    128 064 128; ... % "Road"
    128 000 192; ... % "LaneMkgsDriv"
    192 000 064; ... % "LaneMkgsNonDriv"
    ]

    % "Pavement"
    [
    000 000 192; ... % "Sidewalk"
    064 192 128; ... % "ParkingBlock"
    128 128 192; ... % "RoadShoulder"
    ]

    % "Tree"
    [
    128 128 000; ... % "Tree"
    192 192 000; ... % "VegetationMisc"
    ]
}

```

```

% "SignSymbol"
[
192 128 128; ... % "SignSymbol"
128 128 064; ... % "Misc_Text"
000 064 064; ... % "TrafficLight"
]

% "Fence"
[
064 064 128; ... % "Fence"
]

% "Car"
[
064 000 128; ... % "Car"
064 128 192; ... % "SUVPickupTruck"
192 128 192; ... % "Truck_Bus"
192 064 128; ... % "Train"
128 064 064; ... % "OtherMoving"
]

% "Pedestrian"
[
064 064 000; ... % "Pedestrian"
192 128 064; ... % "Child"
064 000 192; ... % "CartLuggagePram"
064 128 064; ... % "Animal"
]

% "Bicyclist"
[
000 128 192; ... % "Bicyclist"
192 000 192; ... % "MotorcycleScooter"
]

};

end

function pixelLabelColorbar(cmap, classNames)
% Add a colorbar to the current axis. The colorbar is formatted
% to display the class names with the color.

colormap(gca,cmap)

% Add colorbar to current figure.
c = colorbar('peer', gca);

% Use class names for tick marks.
c.TickLabels = classNames;
numClasses = size(cmap,1);

% Center tick labels.
c.Ticks = 1/(numClasses*2):1/numClasses:1;

% Remove tick mark.
c.TickLength = 0;
end

```

```
function cmap = camvidColorMap()
% Define the colormap used by CamVid dataset.

cmap = [
    128 128 128 % Sky
    128 0 0 % Building
    192 192 192 % Pole
    128 64 128 % Road
    60 40 222 % Pavement
    128 128 0 % Tree
    192 128 128 % SignSymbol
    64 64 128 % Fence
    64 0 128 % Car
    64 64 0 % Pedestrian
    0 128 192 % Bicyclist
];
;

% Normalize between [0 1].
cmap = cmap ./ 255;
end

function [imdsTrain, imdsVal, imdsTest, pxdsTrain, pxdsVal, pxdsTest] = partitionCamVidData(imds,pxds)
% Partition CamVid data by randomly selecting 60% of the data for training. The
% rest is used for testing.

% Set initial random state for example reproducibility.
rng(0);
numFiles = numel(imds.Files);
shuffledIndices = randperm(numFiles);

% Use 60% of the images for training.
numTrain = round(0.60 * numFiles);
trainingIdx = shuffledIndices(1:numTrain);

% Use 20% of the images for validation
numVal = round(0.20 * numFiles);
valIdx = shuffledIndices(numTrain+1:numTrain+numVal);

% Use the rest for testing.
testIdx = shuffledIndices(numTrain+numVal+1:end);

% Create image datastores for training and test.
trainingImages = imds.Files(trainingIdx);
valImages = imds.Files(valIdx);
testImages = imds.Files(testIdx);

imdsTrain = imageDatastore(trainingImages);
imdsVal = imageDatastore(valImages);
imdsTest = imageDatastore(testImages);

% Extract class and label IDs info.
classes = pxds.ClassNames;
labelIDs = camvidPixelLabelIDs();

% Create pixel label datastores for training and test.
trainingLabels = pxds.Files(trainingIdx);
valLabels = pxds.Files(valIdx);
testLabels = pxds.Files(testIdx);
```

```

pxdsTrain = pixelLabelDatastore(trainingLabels, classes, labelIDs);
pxdsVal = pixelLabelDatastore(valLabels, classes, labelIDs);
pxdsTest = pixelLabelDatastore(testLabels, classes, labelIDs);
end

function data = augmentImageAndLabel(data, xTrans, yTrans)
% Augment images and pixel label images using random reflection and
% translation.

for i = 1:size(data,1)

tform = randomAffine2d...
    'XReflection',true,...
    'XTranslation', xTrans, ...
    'YTranslation', yTrans);

% Center the view at the center of image in the output space while
% allowing translation to move the output image out of view.
rout = affineOutputView(size(data{i,1}), tform, 'BoundsStyle', 'centerOutput');

% Warp the image and pixel labels using the same transform.
data{i,1} = imwarp(data{i,1}, tform, 'OutputView', rout);
data{i,2} = imwarp(data{i,2}, tform, 'OutputView', rout);

end
end

```

## 参考资料

- [1] Chen, Liang-Chieh et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation." ECCV (2018).
- [2] Brostow, G. J., J. Fauqueur, and R. Cipolla."Semantic object classes in video:A high-definition ground truth database." Pattern Recognition Letters.Vol. 30, Issue 2, 2009, pp 88-97.

## 另请参阅

[pixelLabelDatastore](#) | [pixelLabelImageDatastore](#) | [semanticseg](#) | [labeloverlay](#) |  
[countEachLabel](#) | [segnetLayers](#) | [pixelClassificationLayer](#) | [trainingOptions](#) |  
[imageDataAugmenter](#) | [trainNetwork](#) | [evaluateSemanticSegmentation](#)

## 详细信息

- “Semantic Segmentation” (Computer Vision Toolbox)
- “Object Detection” (Computer Vision Toolbox)
- “使用深度学习对多光谱图像进行语义分割” (第 8-31 页)
- “使用扩张卷积进行语义分割” (第 8-26 页)
- “Getting Started with Semantic Segmentation Using Deep Learning” (Computer Vision Toolbox)
- “Label Pixels for Semantic Segmentation” (Computer Vision Toolbox)
- “预训练的深度神经网络” (第 1-8 页)

## 使用扩张卷积进行语义分割

使用扩张卷积训练语义分割网络。

语义分割网络对图像中的每个像素进行分类，从而生成按类分割的图像。语义分割的应用包括自动驾驶中的道路分割以及医疗诊断中的癌细胞分割。要了解详细信息，请参阅“Getting Started with Semantic Segmentation Using Deep Learning”(Computer Vision Toolbox)。

像 DeepLab [1] 这样的语义分割网络广泛使用扩张卷积（也称为空洞卷积），因为它们可以增加层的感受野（层可以看到的输入区域），而不增加参数或计算量。

### 加载训练数据

该示例使用  $32 \times 32$  三角形图像的简单数据集进行说明。该数据集包括附带的像素标签真实值数据。使用 `imageDatastore` 和 `pixelLabelDatastore` 加载训练数据。

```
dataFolder = fullfile(toolboxdir('vision'), 'visiondata', 'triangleImages');
imageFolderTrain = fullfile(dataFolder, 'trainingImages');
labelFolderTrain = fullfile(dataFolder, 'trainingLabels');
```

为图像创建一个 `imageDatastore`。

```
imdsTrain = imageDatastore(imageFolderTrain);
```

为真实值像素标签创建一个 `pixelLabelDatastore`。

```
classNames = ["triangle" "background"];
labels = [255 0];
pxdsTrain = pixelLabelDatastore(labelFolderTrain, classNames, labels)
```

```
pxdsTrain =
  PixelLabelDatastore with properties:
```

```
    Files: {200x1 cell}
    ClassNames: {2x1 cell}
    ReadSize: 1
    ReadFcn: @readDatastoreImage
    AlternateFileSystemRoots: {}
```

### 创建语义分割网络

此示例使用一个基于扩张卷积的简单语义分割网络。

创建一个用于训练数据的数据源，并获取每个标签的像素计数。

```
pximdsTrain = pixelLabelImageDatastore(imdsTrain, pxdsTrain);
tbl = countEachLabel(pxdsTrain)
```

Name	PixelCount	ImagePixelCount
{'triangle'}	10326	$2.048e+05$
{'background'}	$1.9447e+05$	$2.048e+05$

大多数像素标签用于背景。这种类不平衡使学习过程偏向主导类。要解决此问题，请使用类权重来平衡各类。您可以使用几种方法来计算类权重。一种常见的方法是逆频率加权，其中类权重是类频率的倒数。此方法会增加指定给表示不足的类的权重。使用逆频率加权计算类权重。

```
numberPixels = sum(tbl.PixelCount);
frequency = tbl.PixelCount / numberPixels;
classWeights = 1 ./ frequency;
```

使用输入大小对应于输入图像大小的图像输入层创建一个用于像素分类的网络。接下来，指定对应于卷积层、批量归一化层和 ReLU 层的三个块。对于每个卷积层，指定 32 个具有递增扩张系数的  $3 \times 3$  滤波器，并通过将 'Padding' 选项设置为 'same' 来填充输入以使输入的大小与输出相同。要对像素进行分类，请包括一个具有  $K$  个  $1 \times 1$  卷积的卷积层（其中  $K$  是类的数量），其后是一个 softmax 层和一个具有逆类权重的 pixelClassificationLayer。

```
inputSize = [32 32 1];
filterSize = 3;
numFilters = 32;
numClasses = numel(classNames);

layers = [
    imageInputLayer(inputSize)

    convolution2dLayer(filterSize,numFilters,'DilationFactor',1,'Padding','same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(filterSize,numFilters,'DilationFactor',2,'Padding','same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(filterSize,numFilters,'DilationFactor',4,'Padding','same')
    batchNormalizationLayer
    reluLayer

    convolution2dLayer(1,numClasses)
    softmaxLayer
    pixelClassificationLayer('Classes',classNames,'ClassWeights',classWeights)];
```

## 训练网络

指定训练选项。

```
options = trainingOptions('sgdm',...
    'MaxEpochs', 100, ...
    'MiniBatchSize', 64, ...
    'InitialLearnRate', 1e-3);
```

使用 trainNetwork 训练网络。

```
net = trainNetwork(pximdsTrain,layers,options);
```

Training on single CPU.

Initializing input data normalization.

=====   Epoch   Iteration   Time Elapsed   Mini-batch   Mini-batch   Base Learning         (hh:mm:ss)   Accuracy   Loss   Rate   =====
1   1   00:00:00   91.62%   1.6825   0.0010

	17	50	00:00:20	88.56%	0.2393	0.0010
	34	100	00:00:39	92.08%	0.1672	0.0010
	50	150	00:00:57	93.17%	0.1472	0.0010
	67	200	00:01:14	94.15%	0.1313	0.0010
	84	250	00:01:33	94.47%	0.1167	0.0010
	100	300	00:01:51	95.04%	0.1100	0.0010

## 测试网络

加载测试数据。为图像创建一个 `imageDatastore`。为真实值像素标签创建一个 `pixelLabelDatastore`。

```
imageFolderTest = fullfile(dataFolder,'testImages');
imdsTest = imageDatastore(imageFolderTest);
labelFolderTest = fullfile(dataFolder,'testLabels');
pxdsTest = pixelLabelDatastore(labelFolderTest,classNames,labels);
```

使用测试数据和经过训练的网络进行预测。

```
pxdsPred = semanticseg(imdsTest,net,'MiniBatchSize',32,'WriteLocation',tempdir);
Running semantic segmentation network
-----
* Processed 100 images.
```

使用 `evaluateSemanticSegmentation` 评估预测准确度。

```
metrics = evaluateSemanticSegmentation(pxdsPred,pxdsTest);
Evaluating semantic segmentation results
-----
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 100 images.
* Finalizing... Done.
* Data set metrics:
```

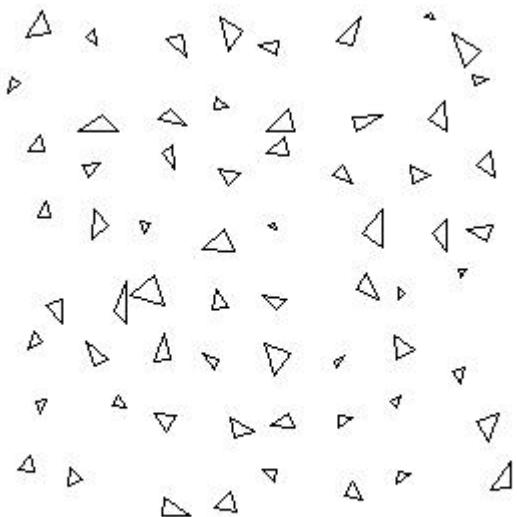
GlobalAccuracy	MeanAccuracy	MeanIoU	WeightedIoU	MeanBFScore
0.95237	0.97352	0.72081	0.92889	0.46416

有关评估语义分割网络的详细信息，请参阅 `evaluateSemanticSegmentation` (Computer Vision Toolbox)。

## 分割新图像

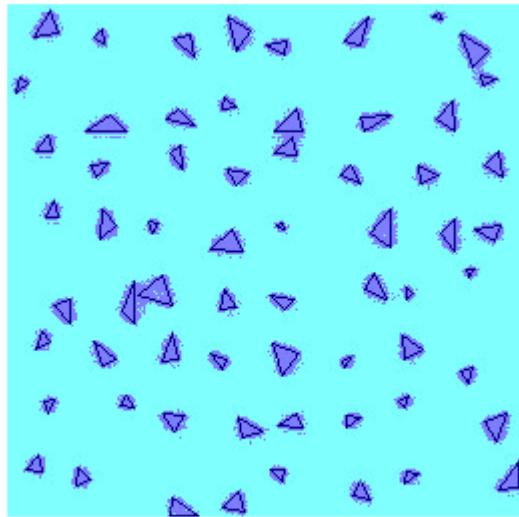
读取并显示测试图像 `triangleTest.jpg`。

```
imgTest = imread('triangleTest.jpg');
figure
imshow(imgTest)
```



使用 **semanticseg** 分割测试图像，并使用 **labeloverlay** 显示结果。

```
C = semanticseg(imgTest,net);
B = labeloverlay(imgTest,C);
figure
imshow(B)
```



## 另请参阅

[pixelLabelDatastore](#) | [pixelLabelImageDatastore](#) | [semanticseg](#) | [labeloverlay](#) |  
[countEachLabel](#) | [pixelClassificationLayer](#) | [trainingOptions](#) | [trainNetwork](#) |  
[evaluateSemanticSegmentation](#) | [convolution2dLayer](#)

## 详细信息

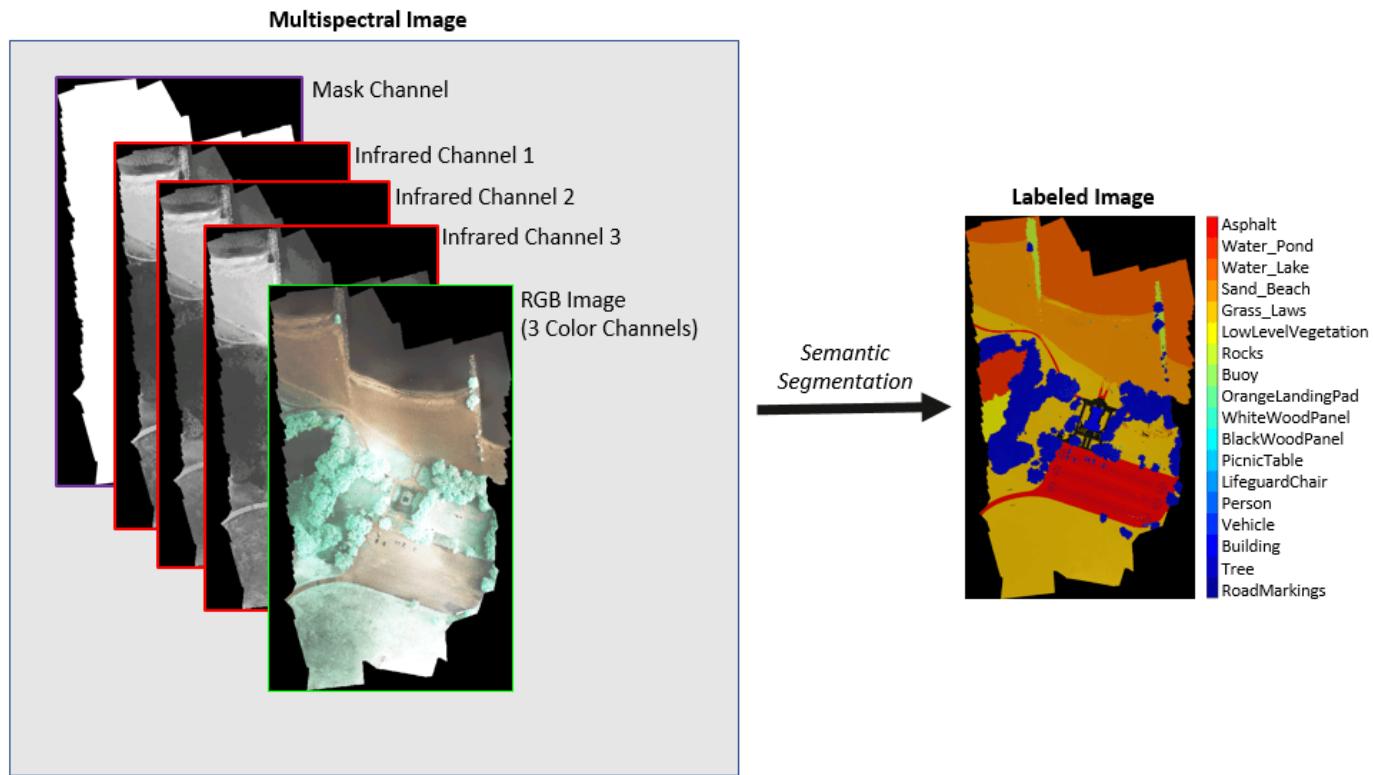
- “使用深度学习进行语义分割” (第 8-11 页)
- “使用深度学习对多光谱图像进行语义分割” (第 8-31 页)
- “Getting Started with Semantic Segmentation Using Deep Learning” (Computer Vision Toolbox)
- “Label Pixels for Semantic Segmentation” (Computer Vision Toolbox)
- “预训练的深度神经网络” (第 1-8 页)

# 使用深度学习对多光谱图像进行语义分割

此示例说明如何使用 U-Net 对包含七个通道的多光谱图像执行语义分割。

语义分割涉及用类标注图像中的每个像素。语义分割的一个应用是跟踪森林采伐，即森林植被随时间的变化。环保机构需要跟踪森林采伐，以评估和量化某个地区的环境和生态健康。

基于深度学习的语义分割可以通过高分辨率航拍照片精确测量植被覆盖度。一个挑战是区分具有相似视觉特性的类，例如尝试将绿色像素分类为草、灌木或树。为了提高分类准确度，一些数据集包含多光谱图像，这些图像提供关于每个像素的附加信息。例如，Hamlin Beach 国家公园数据集用三个近红外通道为彩色图像补充信息，以提供更清晰的分类。



此示例说明如何使用基于深度学习的语义分割方法，根据一组多光谱图像计算某个区域的植被覆盖率。

## 下载数据

此示例使用高分辨率多光谱数据集来训练网络 [1 (第 8-0 页)]。图像集是用无人机在纽约 Hamlin Beach 州立公园拍摄的。数据包含已标注的训练集、验证集和测试集，有 18 个对象类标签。数据文件的大小约为 3.0 GB。

使用 `downloadHamlinBeachMSIData` 辅助函数下载数据集的 MAT 文件版本。此函数作为支持文件包含在本示例中。

```
imageDir = tempdir;
url = 'http://www.cis.rit.edu/~rmk6217/rit18\_data.mat';
downloadHamlinBeachMSIData(url,imageDir);
```

### 检查训练数据

将数据集加载到工作区中。

```
load(fullfile(imageDir,'rit18_data','rit18_data.mat'));
```

检查数据的结构。

```
whos train_data val_data test_data
```

Name	Size	Bytes	Class	Attributes
test_data	7x12446x7654	1333663576	uint16	
train_data	7x9393x5642	741934284	uint16	
val_data	7x8833x6918	855493716	uint16	

多光谱图像数据排列为通道数×宽度×高度数组。但是，在 MATLAB® 中，多通道图像排列为宽度×高度×通道数数组。要重构数据以使通道处于第三个维度中，请使用辅助函数 `switchChannelsToThirdPlane`。此函数作为支持文件包含在本示例中。

```
train_data = switchChannelsToThirdPlane(train_data);
val_data = switchChannelsToThirdPlane(val_data);
test_data = switchChannelsToThirdPlane(test_data);
```

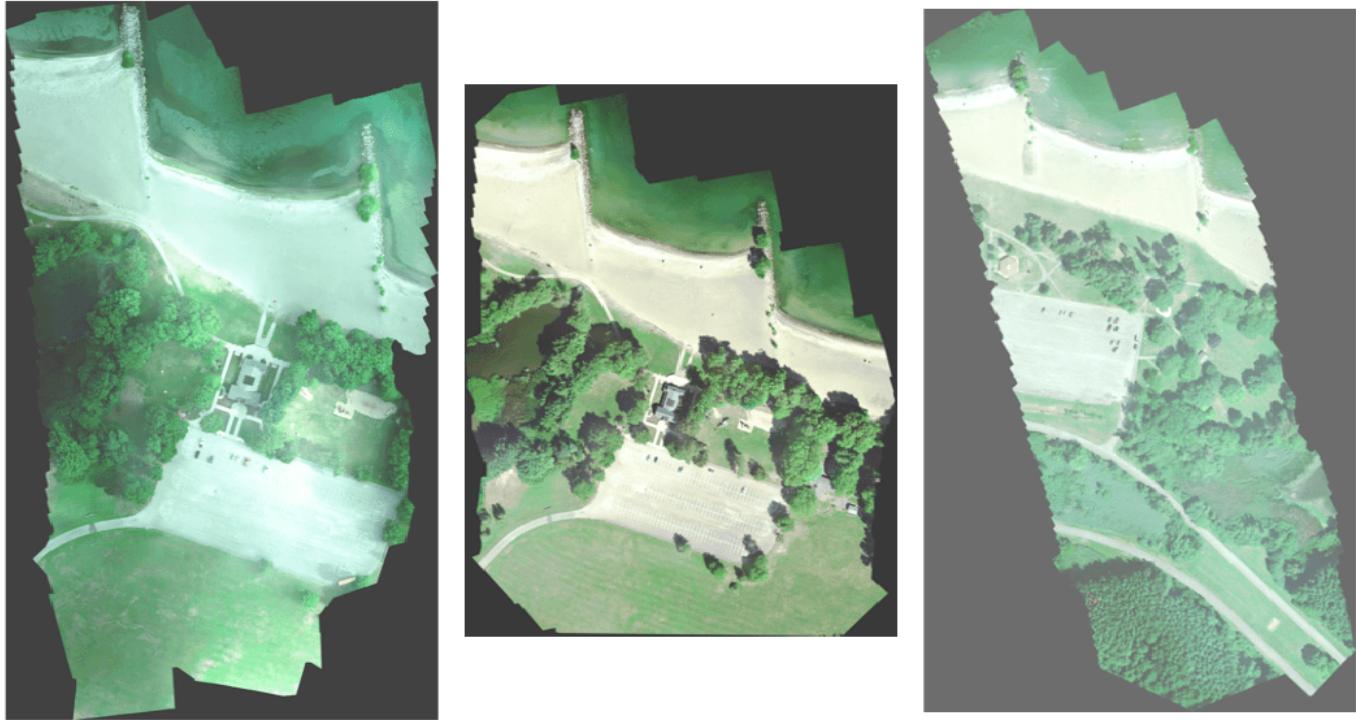
确认数据结构正确。

```
whos train_data val_data test_data
```

Name	Size	Bytes	Class	Attributes
test_data	12446x7654x7	1333663576	uint16	
train_data	9393x5642x7	741934284	uint16	
val_data	8833x6918x7	855493716	uint16	

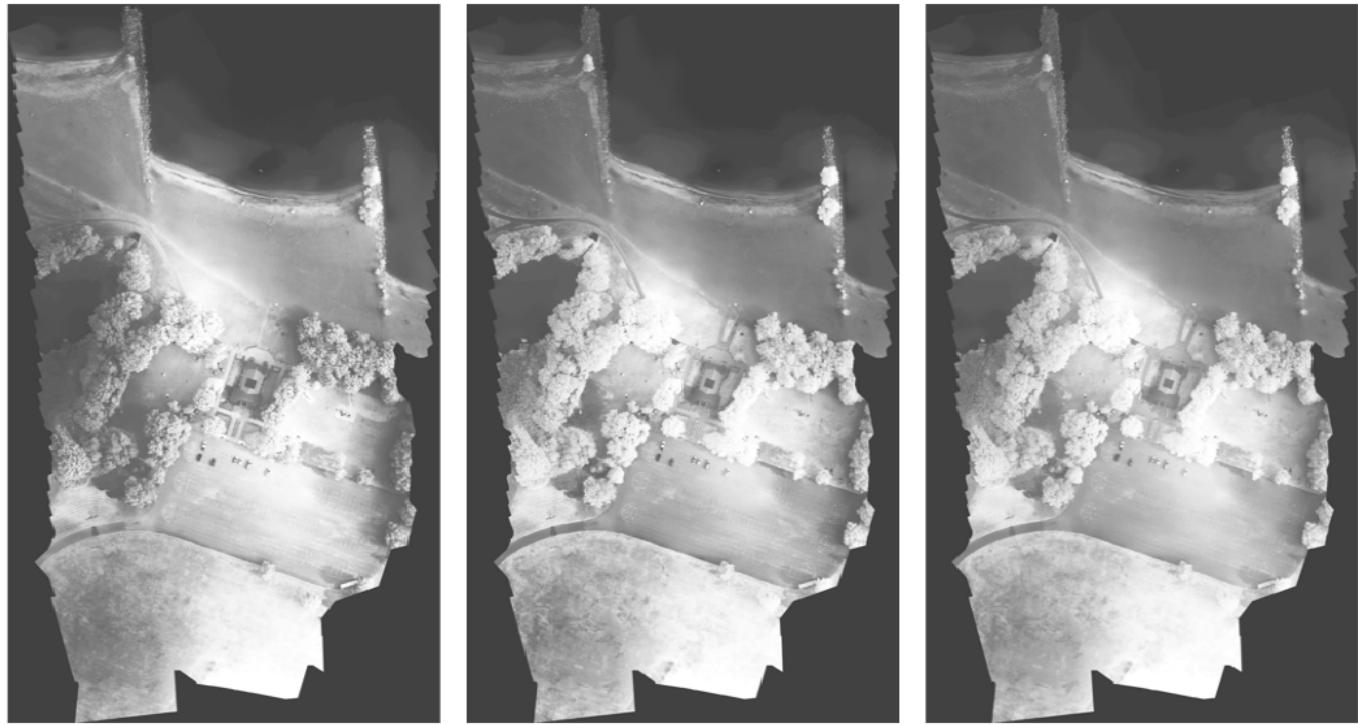
RGB 颜色通道是第 3 个、第 2 个和第 1 个图像通道。将训练、验证和测试图像的颜色分量显示为蒙太奇形式。要使图像在屏幕上看起来更亮，请使用 `histeq` (Image Processing Toolbox) 函数执行直方图均衡化处理。

```
figure
montage([
    histeq(train_data(:,:, [3 2 1])), ...
    histeq(val_data(:,:, [3 2 1])), ...
    histeq(test_data(:,:, [3 2 1])), ...
    'BorderSize', 10, 'BackgroundColor', 'white'
], 'Title', 'RGB Component of Training Image (Left), Validation Image (Center), and Test Image (Right)')
```



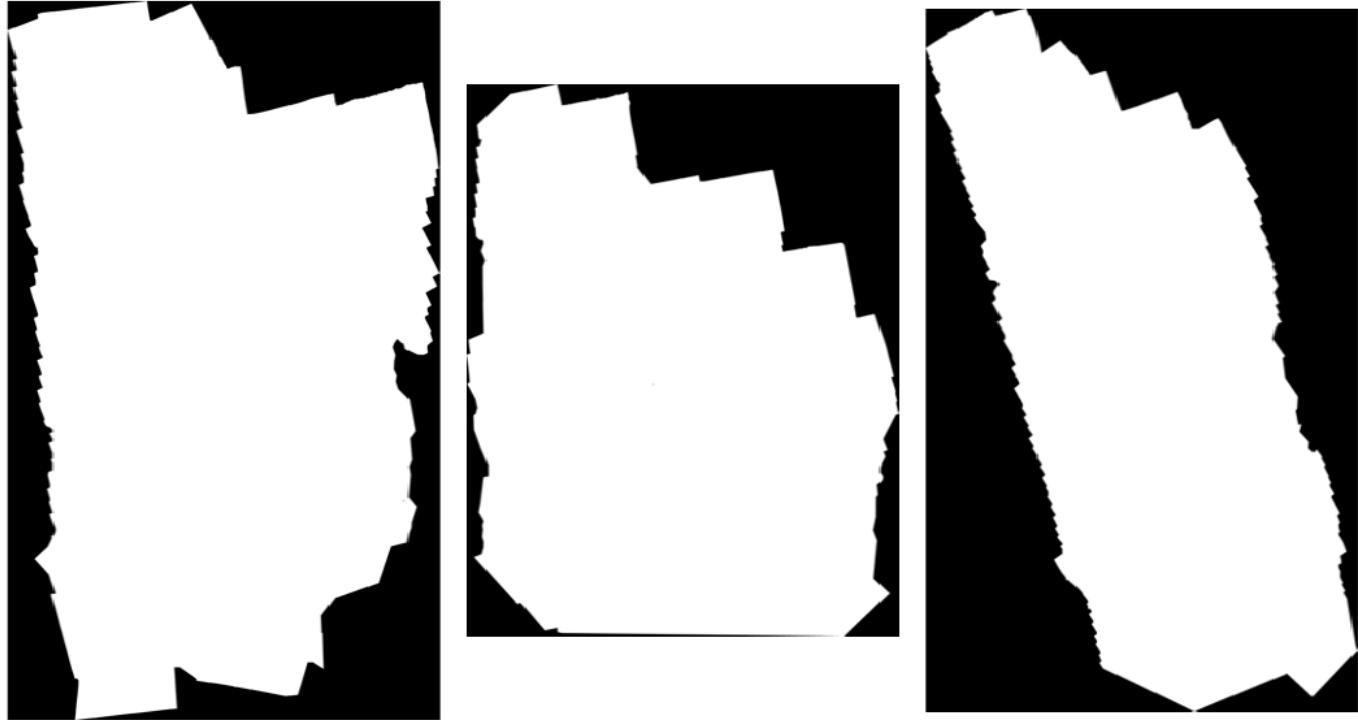
将训练数据的后三个经过直方图均衡化处理的通道显示为蒙太奇形式。这些通道对应于近红外波段，并基于其热特征突出显示图像的不同分量。例如，第二个通道图像中心附近的树比其他两个通道中的树显示的细节要多。

```
figure
montage(...  
    {histeq(train_data(:,:,4)), ...  
     histeq(train_data(:,:,5)), ...  
     histeq(train_data(:,:,6))}, ...  
    'BorderSize',10,'BackgroundColor','white')  
title('IR Channels 1 (Left), 2, (Center), and 3 (Right) of Training Image')
```



通道 7 是指示有效分割区域的掩膜。显示训练、验证和测试图像的掩膜。

```
figure
montage(... ...
    {train_data(:,:,7), ...
    val_data(:,:,7), ...
    test_data(:,:,7)}, ...
    'BorderSize',10,'BackgroundColor','white')
title('Mask of Training Image (Left), Validation Image (Center), and Test Image (Right)')
```



已标注的图像包含用于分割的真实值数据，每个像素分配给 18 个类之一。获取具有对应 ID 的类的列表。

**disp(classes)**

0. Other Class/Image Border
1. Road Markings
2. Tree
3. Building
4. Vehicle (Car, Truck, or Bus)
5. Person
6. Lifeguard Chair
7. Picnic Table
8. Black Wood Panel
9. White Wood Panel
10. Orange Landing Pad
11. Water Buoy
12. Rocks
13. Other Vegetation
14. Grass
15. Sand
16. Water (Lake)
17. Water (Pond)
18. Asphalt (Parking Lot/Walkway)

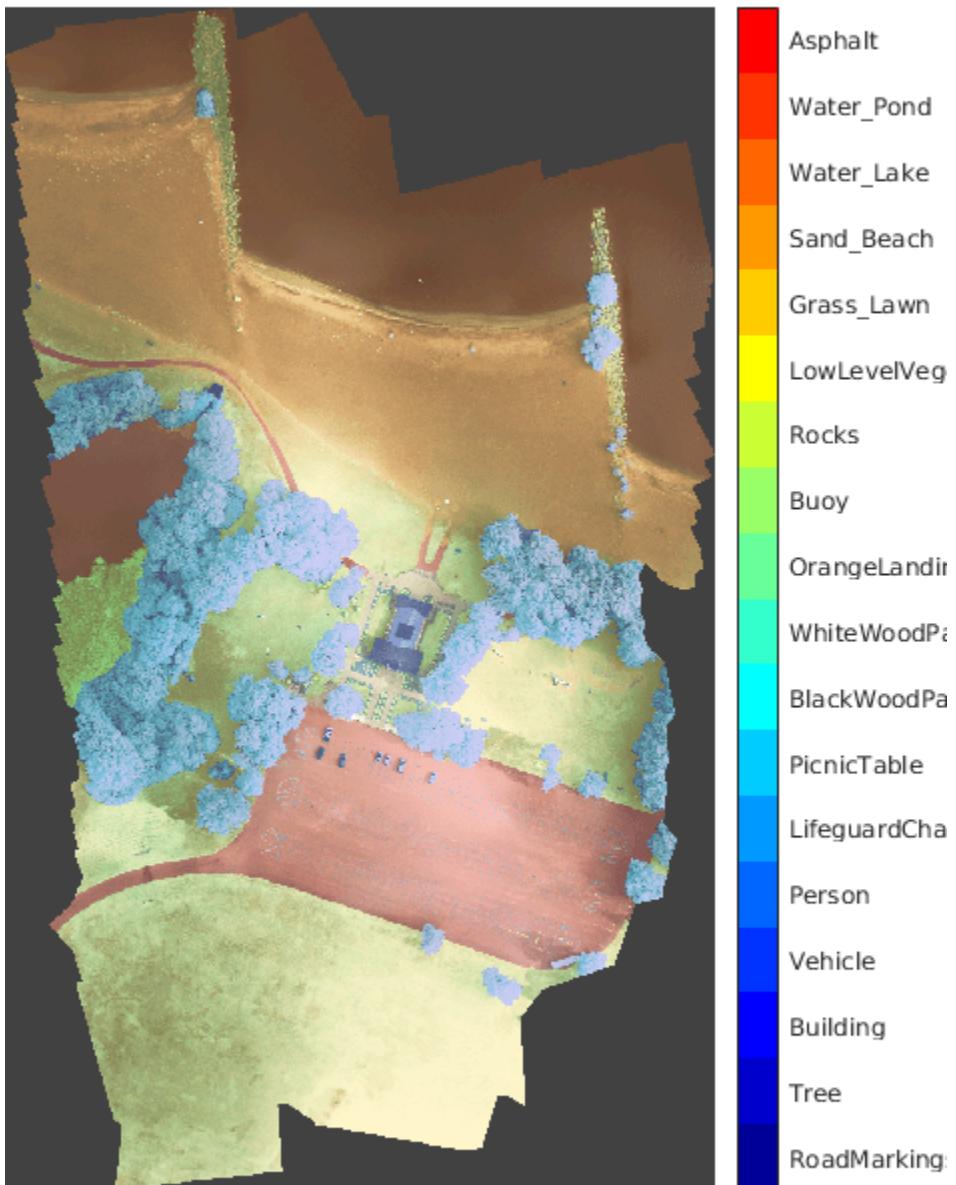
创建一个类名向量。

```
classNames = [ "RoadMarkings","Tree","Building","Vehicle","Person", ...
    "LifeguardChair","PicnicTable","BlackWoodPanel',...
    "WhiteWoodPanel","OrangeLandingPad","Buoy","Rocks",...
    "LowLevelVegetation","Grass_Lawn","Sand_Beach",...
    "Water_Lake","Water_Pond","Asphalt"];
```

在经过直方图均衡化处理的 RGB 训练图像上叠加标签。在图像中添加一个颜色栏。

```
cmap = jet(numel(classNames));
B = labeloverlay(histeq(train_data(:,:,4:6)),train_labels,'Transparency',0.8,'Colormap',cmap);

figure
title('Training Labels')
imshow(B)
N = numel(classNames);
ticks = 1/(N*2):1/N:1;
colorbar('TickLabels',cellstr(classNames),'Ticks',ticks,'TickLength',0,'TickLabelInterpreter','none');
colormap(cmap)
```



将训练数据保存为一个 MAT 文件，将训练标签保存为一个 PNG 文件。

```
save('train_data.mat','train_data');
imwrite(train_labels,'train_labels.png');
```

### 为训练创建随机补片提取数据存储

使用随机补片提取数据存储将训练数据馈送给网络。此数据存储从一个包含真实值图像的图像数据存储和包含像素标签的像素标签数据存储中提取多个对应的随机补片。补片是防止大图像耗尽内存和有效增加可用训练数据量的常用方法。

首先将来自 'train\_data.mat' 的训练图像存储在 **imageDatastore** 中。由于 MAT 文件格式是非标准图像格式，您必须使用 MAT 文件读取器来读取图像数据。您可以使用 MAT 文件读取器辅助函数 **matReader**，它从训练数据中提取前六个通道，省略包含掩膜的最后一个通道。此函数作为支持文件包含在本示例中。

```
imds = imageDatastore('train_data.mat','FileExtensions','.mat','ReadFcn',@matReader);
```

创建一个 **pixelLabelDatastore** (Computer Vision Toolbox) 以存储包含 18 个已标注区域的标签补片。

```
pixelLabelIds = 1:18;
pxds = pixelLabelDatastore('train_labels.png',classNames,pixelLabelIds);
```

从图像数据存储和像素标签数据存储创建一个 **randomPatchExtractionDatastore** (Image Processing Toolbox)。每个小批量包含 16 个大小为  $256 \times 256$  像素的补片。在轮的每次迭代中提取一千个小批量。

```
dsTrain = randomPatchExtractionDatastore(imds,pxds,[256,256],'PatchesPerImage',16000);
```

随机补片提取数据存储 **dsTrain** 在一轮训练的每次迭代中向网络提供小批量数据。预览数据存储以探查数据。

```
inputBatch = preview(dsTrain);
disp(inputBatch)
```

InputImage	ResponsePixelLabelImage
{256×256×6 uint16}	{256×256 categorical}

### 创建 U-Net 网络层

此示例使用 U-Net 网络的一种变体。U-Net 中的初始卷积层序列与最大池化层交叠，从而逐步降低输入图像的分辨率。这些层后跟一系列使用上采样算子散布的卷积层，从而会连续增加输入图像的分辨率 [2 (第 8-0 页) ]。U-Net 的名称源于网络可以绘制成形似字母 U 的对称形状。

此示例对 U-Net 进行了修改以在卷积中使用零填充，从而使卷积的输入和输出具有相同的大小。使用辅助函数 **createUnet** 创建一个具有几个预选超参数的 U-Net。此函数作为支持文件包含在本示例中。

```
inputTileSize = [256,256,6];
lgraph = createUnet(inputTileSize);
disp(lgraph.Layers)
```

58x1 Layer array with layers:

1 'ImageInputLayer'	Image Input	256x256x6 images with 'zerocenter' normalization
2 'Encoder-Section-1-Conv-1'	Convolution	64 3x3x6 convolutions with stride [1 1] and padding [1 1]
3 'Encoder-Section-1-ReLU-1'	ReLU	ReLU
4 'Encoder-Section-1-Conv-2'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1]
5 'Encoder-Section-1-ReLU-2'	ReLU	ReLU
6 'Encoder-Section-1-MaxPool'	Max Pooling	2x2 max pooling with stride [2 2] and padding [1 1]
7 'Encoder-Section-2-Conv-1'	Convolution	128 3x3x64 convolutions with stride [1 1] and padding [1 1]
8 'Encoder-Section-2-ReLU-1'	ReLU	ReLU
9 'Encoder-Section-2-Conv-2'	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1]
10 'Encoder-Section-2-ReLU-2'	ReLU	ReLU
11 'Encoder-Section-2-MaxPool'	Max Pooling	2x2 max pooling with stride [2 2] and padding [1 1]
12 'Encoder-Section-3-Conv-1'	Convolution	256 3x3x128 convolutions with stride [1 1] and padding [1 1]
13 'Encoder-Section-3-ReLU-1'	ReLU	ReLU
14 'Encoder-Section-3-Conv-2'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1]
15 'Encoder-Section-3-ReLU-2'	ReLU	ReLU
16 'Encoder-Section-3-MaxPool'	Max Pooling	2x2 max pooling with stride [2 2] and padding [1 1]
17 'Encoder-Section-4-Conv-1'	Convolution	512 3x3x256 convolutions with stride [1 1] and padding [1 1]
18 'Encoder-Section-4-ReLU-1'	ReLU	ReLU
19 'Encoder-Section-4-Conv-2'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1]
20 'Encoder-Section-4-ReLU-2'	ReLU	ReLU
21 'Encoder-Section-4-DropOut'	Dropout	50% dropout
22 'Encoder-Section-4-MaxPool'	Max Pooling	2x2 max pooling with stride [2 2] and padding [1 1]
23 'Mid-Conv-1'	Convolution	1024 3x3x512 convolutions with stride [1 1] and padding [1 1]
24 'Mid-ReLU-1'	ReLU	ReLU
25 'Mid-Conv-2'	Convolution	1024 3x3x1024 convolutions with stride [1 1] and padding [1 1]
26 'Mid-ReLU-2'	ReLU	ReLU
27 'Mid-DropOut'	Dropout	50% dropout
28 'Decoder-Section-1-UpConv'	Transposed Convolution	512 2x2x1024 transposed convolutions with stride [1 1] and padding [1 1]
29 'Decoder-Section-1-UpReLU'	ReLU	ReLU
30 'Decoder-Section-1-DepthConcatenation'	Depth concatenation	Depth concatenation of 2 inputs
31 'Decoder-Section-1-Conv-1'	Convolution	512 3x3x1024 convolutions with stride [1 1] and padding [1 1]
32 'Decoder-Section-1-ReLU-1'	ReLU	ReLU
33 'Decoder-Section-1-Conv-2'	Convolution	512 3x3x512 convolutions with stride [1 1] and padding [1 1]
34 'Decoder-Section-1-ReLU-2'	ReLU	ReLU
35 'Decoder-Section-2-UpConv'	Transposed Convolution	256 2x2x512 transposed convolutions with stride [1 1] and padding [1 1]
36 'Decoder-Section-2-UpReLU'	ReLU	ReLU
37 'Decoder-Section-2-DepthConcatenation'	Depth concatenation	Depth concatenation of 2 inputs
38 'Decoder-Section-2-Conv-1'	Convolution	256 3x3x512 convolutions with stride [1 1] and padding [1 1]
39 'Decoder-Section-2-ReLU-1'	ReLU	ReLU
40 'Decoder-Section-2-Conv-2'	Convolution	256 3x3x256 convolutions with stride [1 1] and padding [1 1]
41 'Decoder-Section-2-ReLU-2'	ReLU	ReLU
42 'Decoder-Section-3-UpConv'	Transposed Convolution	128 2x2x256 transposed convolutions with stride [1 1] and padding [1 1]
43 'Decoder-Section-3-UpReLU'	ReLU	ReLU
44 'Decoder-Section-3-DepthConcatenation'	Depth concatenation	Depth concatenation of 2 inputs
45 'Decoder-Section-3-Conv-1'	Convolution	128 3x3x256 convolutions with stride [1 1] and padding [1 1]
46 'Decoder-Section-3-ReLU-1'	ReLU	ReLU
47 'Decoder-Section-3-Conv-2'	Convolution	128 3x3x128 convolutions with stride [1 1] and padding [1 1]
48 'Decoder-Section-3-ReLU-2'	ReLU	ReLU
49 'Decoder-Section-4-UpConv'	Transposed Convolution	64 2x2x128 transposed convolutions with stride [1 1] and padding [1 1]
50 'Decoder-Section-4-UpReLU'	ReLU	ReLU
51 'Decoder-Section-4-DepthConcatenation'	Depth concatenation	Depth concatenation of 2 inputs
52 'Decoder-Section-4-Conv-1'	Convolution	64 3x3x128 convolutions with stride [1 1] and padding [1 1]
53 'Decoder-Section-4-ReLU-1'	ReLU	ReLU
54 'Decoder-Section-4-Conv-2'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1]
55 'Decoder-Section-4-ReLU-2'	ReLU	ReLU
56 'Final-ConvolutionLayer'	Convolution	18 1x1x64 convolutions with stride [1 1] and padding [1 1]

```
57 'Softmax-Layer'
58 'Segmentation-Layer'
```

Softmax	softmax
Pixel Classification Layer	Cross-entropy loss

### 选择训练选项

使用具有动量的随机梯度下降 (SGDM) 优化来训练网络。使用 `trainingOptions` 函数指定 SGDM 的超参数设置。

训练深度网络是很费时间的。通过指定高学习率可加快训练速度。然而，这可能会导致网络的梯度爆炸或不受控制地增长，阻碍网络训练成功。要将梯度保持在有意义的范围内，请通过将 '`GradientThreshold`' 指定为 **0.05** 来启用梯度裁剪，并指定 '`GradientThresholdMethod`' 使用梯度的 L2-范数。

```
initialLearningRate = 0.05;
maxEpochs = 150;
minibatchSize = 16;
l2reg = 0.0001;

options = trainingOptions('sgdm',...
    'InitialLearnRate',initialLearningRate, ...
    'Momentum',0.9, ...
    'L2Regularization',l2reg, ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',minibatchSize, ...
    'LearnRateSchedule','piecewise',...
    'Shuffle','every-epoch',...
    'GradientThresholdMethod','l2norm',...
    'GradientThreshold',0.05, ...
    'Plots','training-progress',...
    'VerboseFrequency',20);
```

### 训练网络

默认情况下，示例使用 `downloadTrainedUnet` 辅助函数为该数据集下载 U-Net 的预训练版本。此函数作为支持文件包含在本示例中。借助预训练网络，您无需等待训练完成即可运行整个示例。

要训练网络，请将以下代码中的 `doTraining` 变量设置为 `true`。使用 `trainNetwork` 函数训练模型。

在 GPU 上（如果有）进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。在 NVIDIA Titan X 上训练大约需要 20 个小时。

```
doTraining = false;
if doTraining
    [net,info] = trainNetwork(dsTrain,lgraph,options);
    modelDateTime = string(datetime('now','Format','yyyy-MM-dd-HH-mm-ss'));
    save(strcat("multispectralUnet-",modelDateTime,"-Epoch-",num2str(maxEpochs),".mat"),'net');

else
    trainedUnet_url = 'https://www.mathworks.com/supportfiles/vision/data/multispectralUnet.mat';
    downloadTrainedUnet(trainedUnet_url,imageDir);
    load(fullfile(imageDir,'trainedUnet','multispectralUnet.mat'));
end
```

现在，您可以使用 U-Net 对多光谱图像进行语义分割。

## 对测试数据预测结果

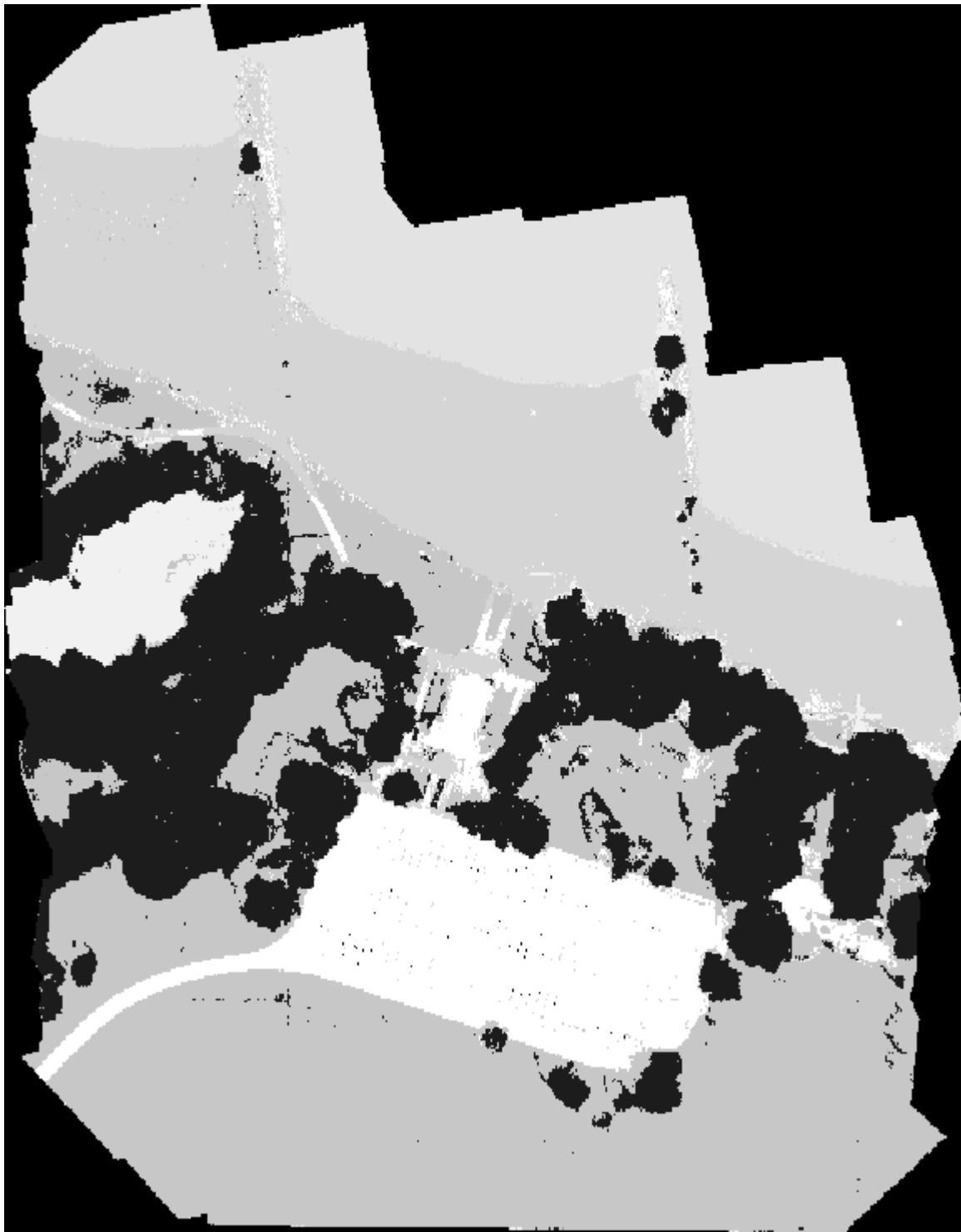
要基于经过训练的网络执行前向传导，请对验证数据集使用辅助函数 `segmentImage`。此函数作为支持文件包含在本示例中。`segmentImage` 使用 `semanticseg` (Computer Vision Toolbox) 函数对图像补片执行分割。

```
predictPatchSize = [1024 1024];
segmentedImage = segmentImage(val_data,net,predictPatchSize);
```

为了只提取分割的有效部分，需要将分割的图像乘以验证数据的掩膜通道。

```
segmentedImage = uint8(val_data(:,:,7)~=0) .* segmentedImage;
```

```
figure
imshow(segmentedImage,[])
title('Segmented Image')
```



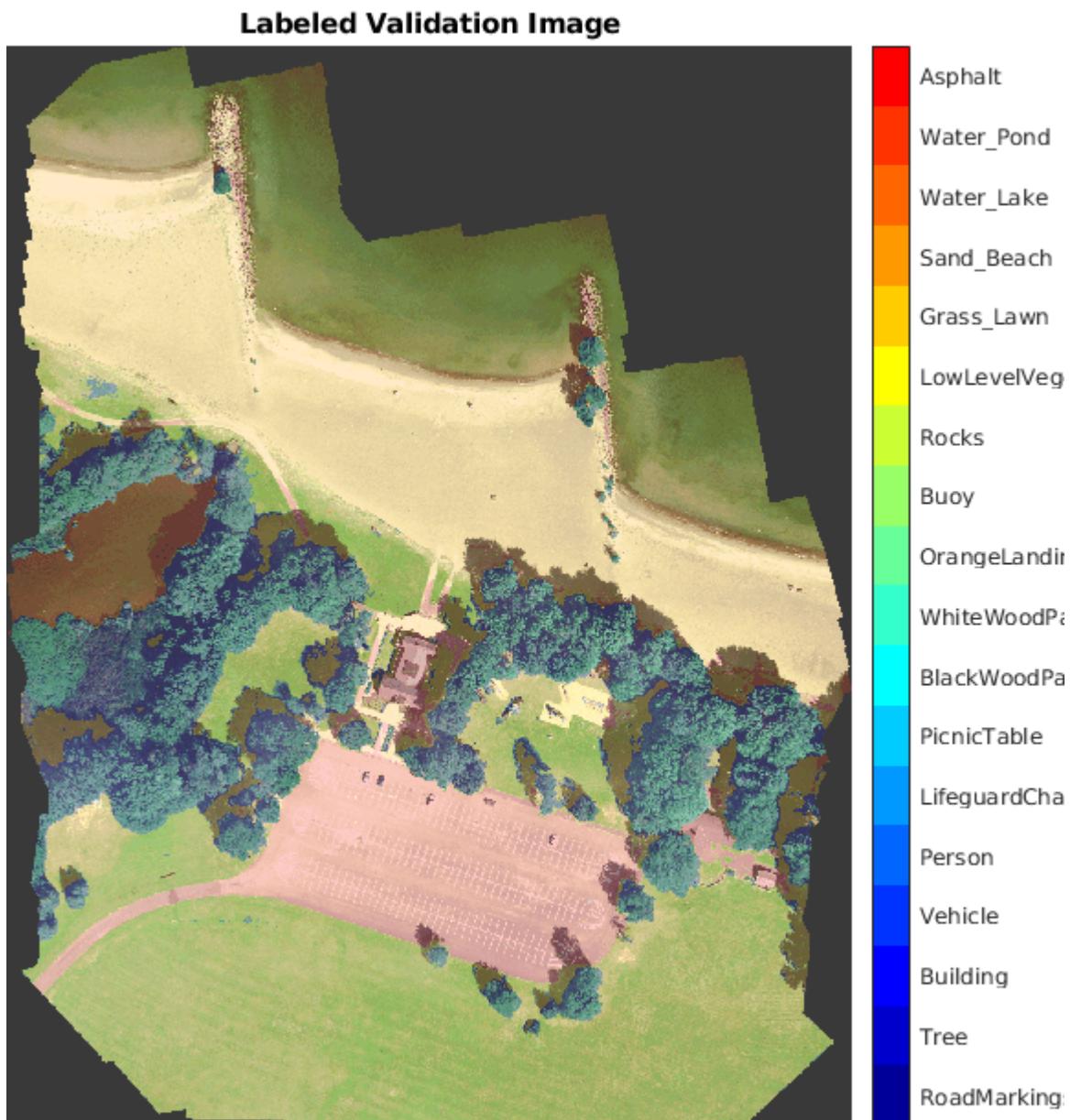
语义分割的输出含有噪声。执行图像后处理以去除噪声和杂散像素。使用 **medfilt2** (Image Processing Toolbox) 函数从分割中去除椒盐噪声。去除噪声后，可视化分割后的图像。

```
segmentedImage = medfilt2(segmentedImage,[7,7]);
imshow(segmentedImage,[]);
title('Segmented Image with Noise Removed')
```



在经过直方图均衡化处理的 RGB 验证图像上叠加分割后的图像。

```
B = labeloverlay(histeq(val_data(:,:,3 2 1)),segmentedImage,'Transparency',0.8,'Colormap',cmap);  
figure  
imshow(B)  
title('Labeled Validation Image')  
colorbar('TickLabels',cellstr(classNames),'Ticks',ticks,'TickLength',0,'TickLabelInterpreter','none');  
colormap(cmap)
```



将分割后的图像和真实值标签另存为 PNG 文件。这些文件将用于计算准确度指标。

```
imwrite(segmentedImage,'results.png');  
imwrite(val_labels,'gtruth.png');
```

## 量化分割准确度

为分割结果和真实值标签创建一个 `pixelLabelDatastore` (Computer Vision Toolbox)。

```
pxdsResults = pixelLabelDatastore('results.png',classNames,pixelLabelIds);
pxdsTruth = pixelLabelDatastore('gtruth.png',classNames,pixelLabelIds);
```

使用 `evaluateSemanticSegmentation` (Computer Vision Toolbox) 函数衡量语义分割的全局准确度。

```
ssm = evaluateSemanticSegmentation(pxdsResults,pxdsTruth,'Metrics','global-accuracy');
```

Evaluating semantic segmentation results

---

- \* Selected metrics: global accuracy.
- \* Processed 1 images.
- \* Finalizing... Done.
- \* Data set metrics:

GlobalAccuracy

---

0.90698

全局准确度分数表明，正确分类的像素稍大于 90%。

## 计算植被覆盖率

此示例的最终目标是计算多光谱图像中的植被覆盖率。

求已加植被标签的像素的数量。标签 ID 2 ("Trees")、13 ("LowLevelVegetation") 和 14 ("Grass\_Lawn") 都属于植被类。通过对掩膜图像的关注区域中的像素求和，还可以求得有效像素的总数。

```
vegetationClassIds = uint8([2,13,14]);
vegetationPixels = ismember(segmentedImage(:,vegetationClassIds));
validPixels = (segmentedImage~=0);

numVegetationPixels = sum(vegetationPixels(:));
numValidPixels = sum(validPixels(:));
```

通过用植被像素数除以有效像素数来计算植被覆盖率。

```
percentVegetationCover = (numVegetationPixels/numValidPixels)*100;
fprintf('The percentage of vegetation cover is %3.2f%%.',percentVegetationCover);
```

The percentage of vegetation cover is 51.72%.

## 参考资料

[1] Kemker, R., C. Salvaggio, and C. Kanan."High-Resolution Multispectral Dataset for Semantic Segmentation."CoRR, abs/1703.01918. 2017.

[2] Ronneberger, O., P. Fischer, and T. Brox."U-Net:Convolutional Networks for Biomedical Image Segmentation."CoRR, abs/1505.04597. 2015.

## 另请参阅

`trainingOptions` | `trainNetwork` | `randomPatchExtractionDatastore` | `pixelLabelDatastore` | `semanticseg` | `evaluateSemanticSegmentation` | `imageDatastore` | `histeq` | `unetLayers`

## 详细信息

- “Getting Started with Semantic Segmentation Using Deep Learning” (Computer Vision Toolbox)
- “使用深度学习进行语义分割” (第 8-11 页)
- “使用扩张卷积进行语义分割” (第 8-26 页)
- “Datastores for Deep Learning”

## 外部网站

- <https://github.com/rmkemker/RIT-18>

## 使用深度学习进行三维脑肿瘤分割

此示例说明如何基于三维医学图像训练三维 U-Net 神经网络，并执行脑肿瘤的语义分割。

语义分割中，图像中的每个像素或三维体的每个体素都被标注为某一类。此示例说明如何在磁共振成像 (MRI) 扫描中使用深度学习方法执行脑肿瘤二元语义分割。在该二值分割中，将每个像素标注为肿瘤或背景。

此示例使用三维 U-Net 架构 [1 (第 8-0 页)] 执行脑肿瘤分割。U-Net 是一种快速、高效、简单的网络，在语义分割领域非常流行。

医学图像分割面临的挑战之一是存储和处理三维体所需的内存量。由于 GPU 资源的限制，基于整个输入图像体训练网络是不切实际的。此示例通过基于图像补片训练网络来解决此问题。该示例使用重叠分块策略，将测试补片缝合为完整的经过分割的测试图像体。该示例使用神经网络中卷积的有效部分来避免边界伪影 [5 (第 8-0 页)]。

医学图像分割面临的另一挑战是当使用常规的交叉熵损失时，数据中的类不平衡会妨碍训练。此示例通过使用加权多类 Dice 损失函数 [4 (第 8-0 页)] 解决此问题。对类进行加权有助于抵消较大区域对 Dice 分数的影响，使网络更容易学习如何分割较小区域。

### 下载训练、验证和测试数据

此示例使用 BraTS 数据集 [2 (第 8-0 页)]。BraTS 数据集包含脑肿瘤（即胶质瘤，最常见的原发性恶性脑肿瘤）的 MRI 扫描。该数据文件的大小约为 7 GB。如果您不想下载 BraTS 数据集，请直接转至本示例中的下载预训练网络和样本测试集 (第 8-0 页) 部分。

创建一个目录以存储 BraTS 数据集。

```
imageDir = fullfile(tempdir,'BraTS');
if ~exist(imageDir,'dir')
    mkdir(imageDir);
end
```

要下载 BraTS 数据，请访问 Medical Segmentation Decathlon 网站，然后点击 “Download Data” 链接。下载 "Task01\_BrainTumour.tar" 文件 [3 (第 8-0 页)]。将该 TAR 文件解压缩到由 `imageDir` 变量指定的目录中。成功解压缩后，`imageDir` 将包含一个名为 `Task01_BrainTumour` 的目录，该目录有三个子目录：`imagesTr`、`imagesTs` 和 `labelsTr`。

该数据集包含 750 个四维体，每个四维体代表一组三维图像。每个四维体的大小为  $240 \times 240 \times 155 \times 4$ ，其中前三个维度对应于三维体图像的高度、宽度和深度。第四个维度对应于不同的扫描形态。该数据集分为 484 个带体素标签的训练图像体和 266 个测试图像体。测试图像体没有标签，因此该示例不使用这些测试数据。取而代之，示例将 484 个训练图像体拆分成三个独立的数据集，分别用于训练、验证和测试。

### 预处理训练和验证数据

为了更高效地训练三维 U-Net 网络，使用辅助函数 `preprocessBraTSdataset` 预处理 MRI 数据。此函数作为支持文件包含在本示例中。

该辅助函数执行以下操作：

- 将数据裁剪到主要包含大脑和肿瘤的区域。裁剪数据可以减小数据的大小，同时保留每个 MRI 图像体的最关键部分及其对应标签。
- 通过减去均值并除以裁剪后的大脑区域的标准差，独立地对每个图像体的每个形态进行归一化。
- 将 484 个训练图像体拆分成 400 个训练集、29 个验证集和 55 个测试集。

数据预处理可能需要大约 30 分钟才能完成。

```
sourceDataLoc = [imageDir filesep 'Task01_BrainTumour'];
preprocessDataLoc = fullfile(tempdir,'BraTS','preprocessedDataset');
preprocessBraTSdataset(preprocessDataLoc,sourceDataLoc);
```

### 为训练和验证创建随机补片提取数据存储

使用随机补片提取数据存储将训练数据馈送到网络，并验证训练进度。此数据存储从真实值图像提取随机补片和对应的像素标签数据。补片是一种常用方法，用于在使用任意大的图像体训练时防止内存耗尽。

创建一个  `imageDatastore` 来存储三维图像数据。由于 MAT 文件格式是非标准图像格式，您必须使用 MAT 文件读取器来读取图像数据。您可以使用 MAT 文件读取器辅助函数 `matRead`。此函数作为支持文件包含在本示例中。

```
volReader = @(x) matRead(x);
volLoc = fullfile(preprocessDataLoc,'imagesTr');
voldS = imageDatastore(volLoc, ...
    'FileExtensions','.mat','ReadFcn',volReader);
```

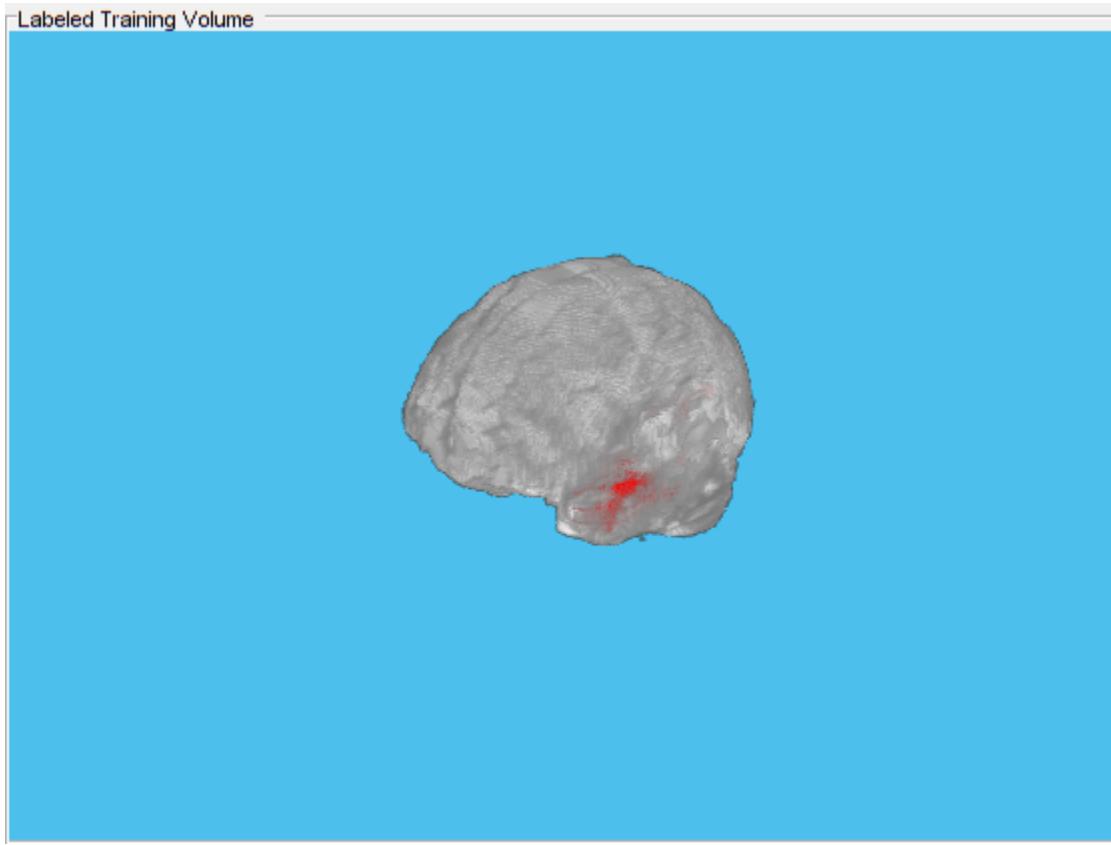
创建一个  `pixelLabelDatastore` (Computer Vision Toolbox) 来存储标签。

```
lblLoc = fullfile(preprocessDataLoc,'labelsTr');
classNames = ["background","tumor"];
pixelLabelID = [0 1];
pxds = pixelLabelDatastore(lblLoc,classNames,pixelLabelID, ...
    'FileExtensions','.mat','ReadFcn',volReader);
```

预览一个带标签的图像体。使用 `labelvolshow` (Image Processing Toolbox) 函数显示标注图像体。通过将背景标签的可见性 (1) 设置为 0，使背景完全透明。

```
volume = preview(voldS);
label = preview(pxds);

viewPnl = uipanel(figure,'Title','Labeled Training Volume');
hPred = labelvolshow(label,volume(:,:,:,:),'Parent',viewPnl, ...
    'LabelColor',[0 0 0;1 0 0]);
hPred.LabelVisibility(1) = 0;
```



创建一个包含训练图像和像素标签数据的 `randomPatchExtractionDatastore` (Image Processing Toolbox)。指定  $132 \times 132 \times 132$  体素的补片大小。指定 'PatchesPerImage' 以在训练期间从每对图像体和标签中提取 16 个随机定位的补片。指定小批量大小为 8。

```
patchSize = [132 132 132];
patchPerImage = 16;
miniBatchSize = 8;
patchds = randomPatchExtractionDatastore(volds,pxds,patchSize, ...
    'PatchesPerImage',patchPerImage);
patchds.MiniBatchSize = miniBatchSize;
```

按照相同的步骤创建一个包含验证图像和像素标签数据的 `randomPatchExtractionDatastore`。您可以使用验证数据来评估网络随着时间的推移是在持续学习、欠拟合还是过拟合。

```
volLocVal = fullfile(preprocessDataLoc,'imagesVal');
voldsVal = imageDatastore(volLocVal, ...
    'FileExtensions','.mat','ReadFcn',volReader);

lblLocVal = fullfile(preprocessDataLoc,'labelsVal');
pxdsVal = pixelLabelDatastore(lblLocVal,classNames,pixelLabelID, ...
    'FileExtensions','.mat','ReadFcn',volReader);

dsVal = randomPatchExtractionDatastore(voldsVal,pxdsVal,patchSize, ...
    'PatchesPerImage',patchPerImage);
dsVal.MiniBatchSize = miniBatchSize;
```

通过使用 `transform` 函数和辅助函数 `augmentAndCrop3dPatch` 指定的自定义预处理操作来增强训练数据和验证数据。此函数作为支持文件包含在本示例中。

**augmentAndCrop3dPatch** 函数执行以下操作：

- 1 随机旋转和翻转训练数据，使训练更加稳健。该函数不旋转或翻转验证数据。
- 2 将响应补片裁剪为网络的输出大小  $44 \times 44 \times 44$  体素。

```
dataSource = 'Training';
dsTrain = transform(patchds,@(patchIn)augmentAndCrop3dPatch(patchIn,dataSource));

dataSource = 'Validation';
dsVal = transform(dsVal,@(patchIn)augmentAndCrop3dPatch(patchIn,dataSource));
```

### 设置三维 U-Net 层

此示例使用三维 U-Net 网络 [1 (第 8-0 页)]。U-Net 中的初始卷积层序列与最大池化层交叠，从而逐步降低输入图像的分辨率。这些层后跟一系列使用上采样算子散布的卷积层，从而会连续增加输入图像的分辨率。在每个 ReLU 层之前引入一个批量归一化层。U-Net 的名称源于网络可以绘制成形似字母 U 的对称形状。

使用 **unetLayers** (Computer Vision Toolbox) 函数创建一个默认的三维 U-Net 网络。指定二类分割。还要指定有效的卷积填充，以避免在使用重叠分块策略预测测试图像体时出现边界伪影。

```
inputPatchSize = [132 132 132 4];
numClasses = 2;
[lgraph,outPatchSize] = unet3dLayers(inputPatchSize,numClasses,'ConvolutionPadding','valid');
```

为了更好地分割较小的肿瘤区域并减少较大背景区域的影响，此示例使用 **dicePixelClassificationLayer** (Computer Vision Toolbox)。将像素分类层替换为 Dice 像素分类层。

```
outputLayer = dicePixelClassificationLayer('Name','Output');
lgraph = replaceLayer(lgraph,'Segmentation-Layer',outputLayer);
```

数据已在此示例的预处理训练和验证数据 (第 8-0 页) 部分进行归一化。**image3dInputLayer** 中没有必要进行数据归一化，因此将输入层替换为没有数据归一化的输入层。

```
inputLayer = image3dInputLayer(inputPatchSize,'Normalization','none','Name','ImageInputLayer');
lgraph = replaceLayer(lgraph,'ImageInputLayer',inputLayer);
```

或者，您可以使用 Deep Learning Toolbox™ 中的深度网络设计器来修改三维 U-Net 网络。

绘制更新后的三维 U-Net 网络图。

```
analyzeNetwork(lgraph)
```

### 指定训练选项

使用 **adam** 优化求解器来训练网络。使用 **trainingOptions** 函数指定超参数设置。初始学习率设置为  $5e-4$ ，并在训练期间逐渐降低。您可以根据您的 GPU 内存情况尝试调整 **MiniBatchSize** 属性。为了最大限度地利用 GPU 内存，最好使用大输入补片而不是大批量大小。请注意，批量归一化层对于较小的 **MiniBatchSize** 值不是很有效。根据 **MiniBatchSize** 调整初始学习率。

```
options = trainingOptions('adam',...
    'MaxEpochs',50, ...
    'InitialLearnRate',5e-4, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',5, ...
    'LearnRateDropFactor',0.95, ...
    'ValidationData',dsVal, ...
```

```
'ValidationFrequency',400, ...
'Plots','training-progress', ...
'Verbose',false, ...
'MiniBatchSize',miniBatchSize);
```

### 下载预训练网络和样本测试集

下载预训练版本的三维 U-Net 和 Brats 数据集 [3 (第 8-0 页)] 中的五个样本测试图像体及其对应的标签。预训练的模型和样本数据使您能够对测试数据执行分割，而无需下载完整的数据集或等待网络完成训练。

```
trained3DUnet_url = 'https://www.mathworks.com/supportfiles/vision/data/brainTumor3DUNetValid.mat';
sampleData_url = 'https://www.mathworks.com/supportfiles/vision/data/sampleBraTSTestSetValid.tar.gz';

imageDir = fullfile(tempdir,'BraTS');
if ~exist(imageDir,'dir')
    mkdir(imageDir);
end
```

```
downloadTrained3DUnetSampleData(trained3DUnet_url,sampleData_url,imageDir);
```

### 训练网络

默认情况下，该示例加载一个预训练的三维 U-Net 网络。借助预训练网络，您无需等待训练完成，即可运行整个示例。

要训练网络，请将以下代码中的 `doTraining` 变量设置为 `true`。使用 `trainNetwork` 函数训练模型。

在 GPU 上（如果有）进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。在包含 4 个 NVIDIA™ Titan Xp GPU 的多 GPU 系统上进行训练需要大约 30 个小时，根据您的 GPU 硬件情况，训练时间可能会更长。

```
doTraining = false;
if doTraining
    modelDateTime = string(datetime('now','Format','yyyy-MM-dd-HH-mm-ss'));
    [net,info] = trainNetwork(dsTrain,lgraph,options);
    save(strcat("trained3DUNet-",modelDateTime,"-Epoch-",num2str(options.MaxEpochs),".mat"),'net');
else
    inputPatchSize = [132 132 132 4];
    outPatchSize = [44 44 44 2];
    load(fullfile(imageDir,'trained3DUNet','brainTumor3DUNetValid.mat'));
end
```

### 对测试数据执行分割

强烈建议使用 GPU 来执行图像体的语义分割（需要 Parallel Computing Toolbox™）。

选择包含真实值图像体和测试标签的测试数据源。如果您在以下代码中将 `useFullTestSet` 变量保留为 `false`，则该示例使用五个图像体进行测试。如果将 `useFullTestSet` 变量设置为 `true`，则该示例使用从完整数据集选择的 55 个测试图像。

```
useFullTestSet = false;
if useFullTestSet
    volLocTest = fullfile(preprocessDataLoc,'imagesTest');
    lblLocTest = fullfile(preprocessDataLoc,'labelsTest');
else
    volLocTest = fullfile(imageDir,'sampleBraTSTestSetValid','imagesTest');
```

```

lblLocTest = fullfile(imageDir,'sampleBraTSTestSetValid','labelsTest');
classNames = ["background","tumor"];
pixelLabelID = [0 1];
end

```

voldsTest 变量存储真实值测试图像。pxdsTest 变量存储真实值标签。

```

volReader = @(x) matRead(x);
voldsTest = imageDatastore(volLocTest, ...
    'FileExtensions','.mat','ReadFcn',volReader);
pxdsTest = pixelLabelDatastore(lblLocTest,classNames,pixelLabelID, ...
    'FileExtensions','.mat','ReadFcn',volReader);

```

使用重叠分块策略来预测每个测试图像体的标签。对每个测试图像体都进行填充，以使输入大小是网络输出大小的倍数，并补偿有效卷积的效应。重叠分块算法选择重叠补片，使用 `semanticseg` (Computer Vision Toolbox) 函数预测每个补片的标签，然后重新组合补片。

```

id = 1;
while hasdata(voldsTest)
    disp(['Processing test volume ' num2str(id)]);

    tempGroundTruth = read(pxdsTest);
    groundTruthLabels{id} = tempGroundTruth{1};
    vol{id} = read(voldsTest);

    % Use reflection padding for the test image.
    % Avoid padding of different modalities.
    volSize = size(vol{id},(1:3));
    padSizePre = (inputPatchSize(1:3)-outPatchSize(1:3))/2;
    padSizePost = (inputPatchSize(1:3)-outPatchSize(1:3))/2 + (outPatchSize(1:3)-mod(volSize,outPatchSize(1:3)));
    volPaddedPre = padarray(vol{id},padSizePre,'symmetric','pre');
    volPadded = padarray(volPaddedPre,padSizePost,'symmetric','post');
    [heightPad,widthPad,depthPad,~] = size(volPadded);
    [height,width,depth,~] = size(vol{id});

    tempSeg = categorical(zeros([height,width,depth],'uint8'),[0;1],classNames);

    % Overlap-tile strategy for segmentation of volumes.
    for k = 1:outPatchSize(3):depthPad-inputPatchSize(3)+1
        for j = 1:outPatchSize(2):widthPad-inputPatchSize(2)+1
            for i = 1:outPatchSize(1):heightPad-inputPatchSize(1)+1
                patch = volPadded( i:i+inputPatchSize(1)-1, ...
                    j:j+inputPatchSize(2)-1, ...
                    k:k+inputPatchSize(3)-1,:);
                patchSeg = semanticseg(patch,net);
                tempSeg(i:i+outPatchSize(1)-1, ...
                    j:j+outPatchSize(2)-1, ...
                    k:k+outPatchSize(3)-1) = patchSeg;
            end
        end
    end

    % Crop out the extra padded region.
    tempSeg = tempSeg(1:height,1:width,1:depth);

    % Save the predicted volume result.
    predictedLabels{id} = tempSeg;

```

```

    id=id+1;
end

Processing test volume 1
Processing test volume 2
Processing test volume 3
Processing test volume 4
Processing test volume 5

```

### 将真实值与网络预测进行比较

选择测试图像之一来评估语义分割的准确度。从四维体数据中提取第一个形态，并将此三维体存储在变量 `vol3d` 中。

```

volId = 1;
vol3d = vol{volId}(:,:,:,:1);

```

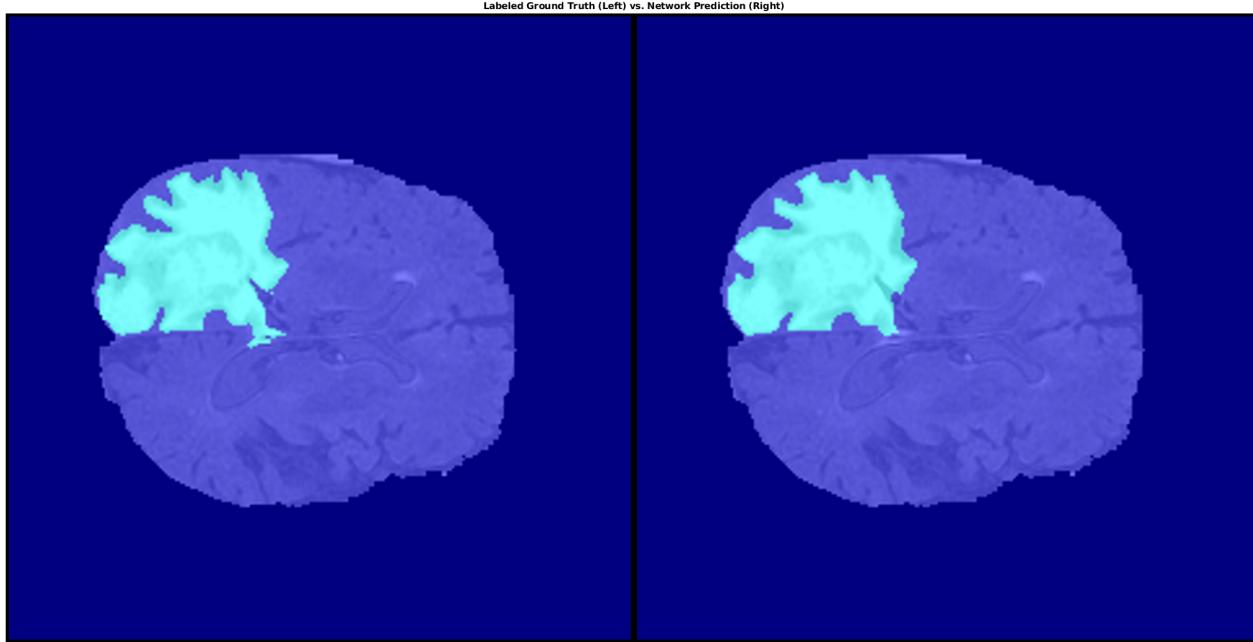
使用蒙太奇显示真实值和预测标签沿深度方向的中心切片。

```

zID = size(vol3d,3)/2;
zSliceGT = labeloverlay(vol3d(:,:,:,zID),groundTruthLabels{volId}(:,:,zID));
zSlicePred = labeloverlay(vol3d(:,:,:,zID),predictedLabels{volId}(:,:,zID));

figure
montage({zSliceGT,zSlicePred},'Size',[1 2],'BorderSize',5)
title('Labeled Ground Truth (Left) vs. Network Prediction (Right)')

```

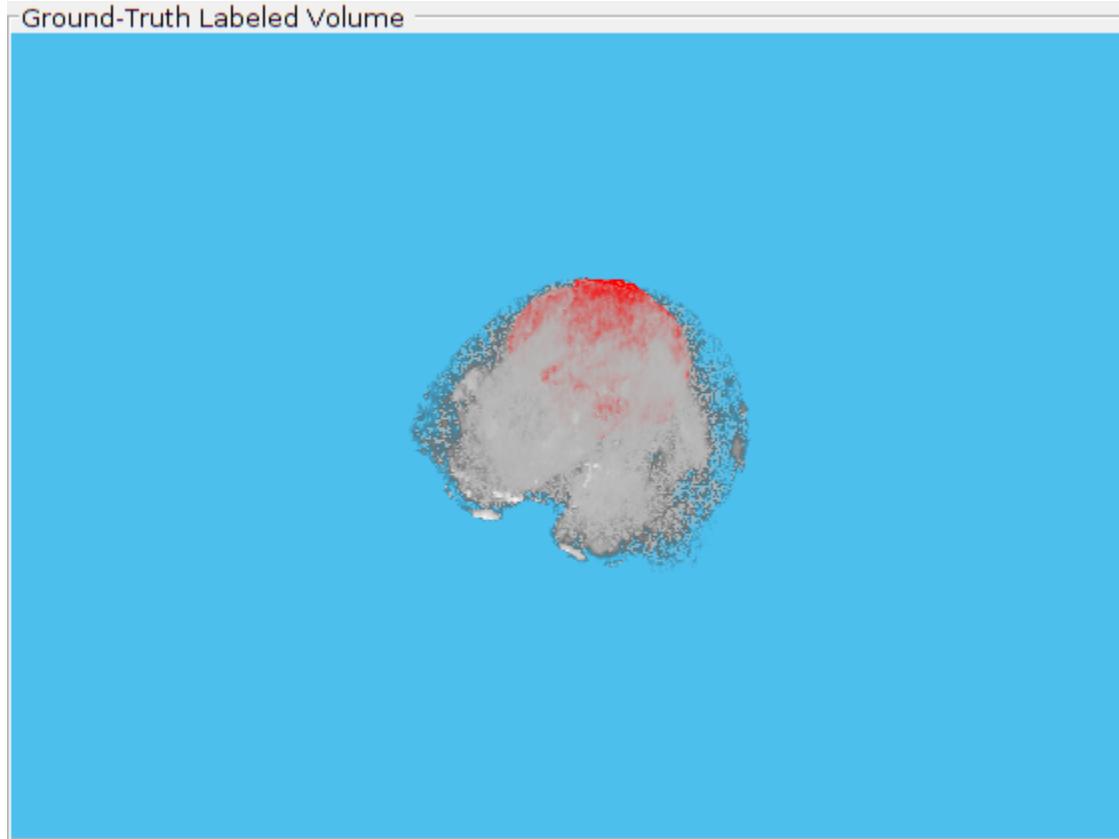


使用 `labelvolshow` (Image Processing Toolbox) 函数显示标注真实值的图像体。通过将背景标签的可见性 (1) 设置为 0，使背景完全透明。由于肿瘤在脑组织内部，因此要使一些大脑体素透明，以便肿瘤可见。要使一些大脑体素透明，请将图像体阈值指定为 [0, 1] 范围内的一个数字。归一化的图像体强度低于此阈值时即完全透明。此示例将图像体阈值设置为小于 1，以使一些大脑像素保持可见，从而给出肿瘤在大脑中所处空间位置的上下文。

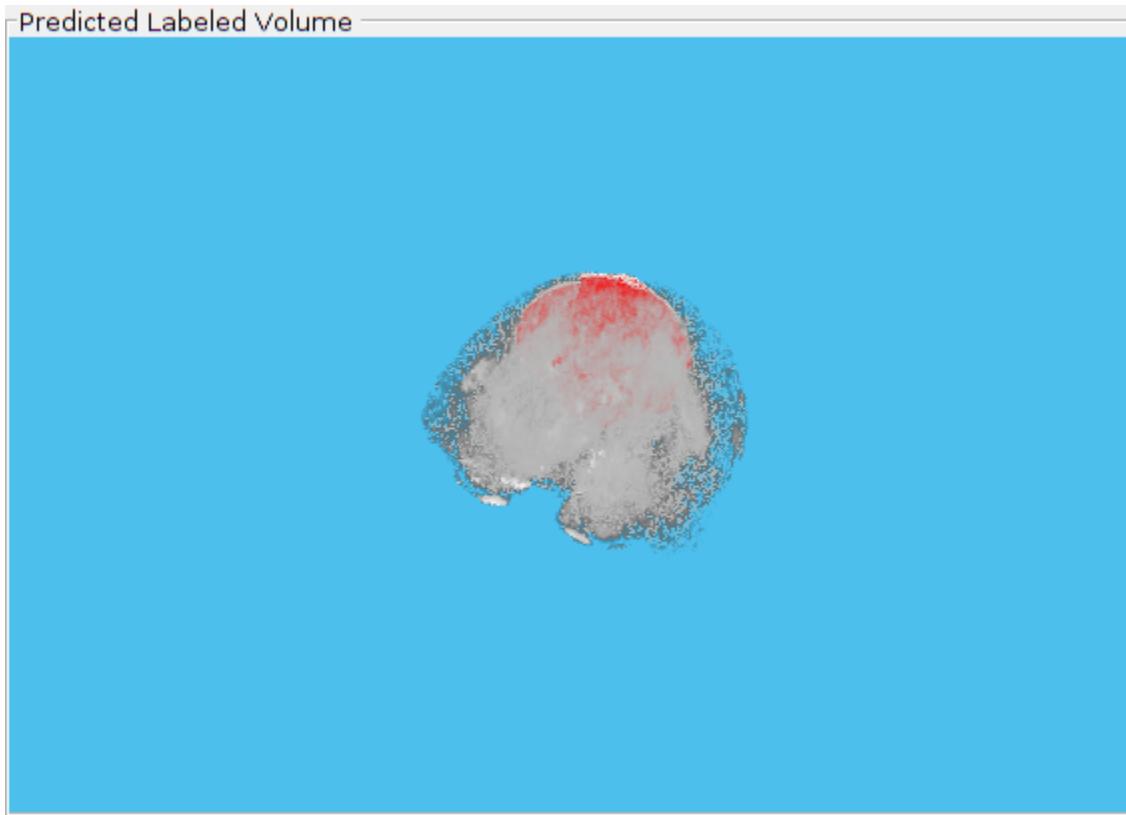
```
viewPnlTruth = uipanel.figure,'Title','Ground-Truth Labeled Volume');
hTruth = labelvolshow(groundTruthLabels{volId},vol3d,'Parent',viewPnlTruth, ...
    'LabelColor',[0 0 0;1 0 0],'VolumeThreshold',0.68);
hTruth.LabelVisibility(1) = 0;
```

对于同一图像体，显示预测的标签。

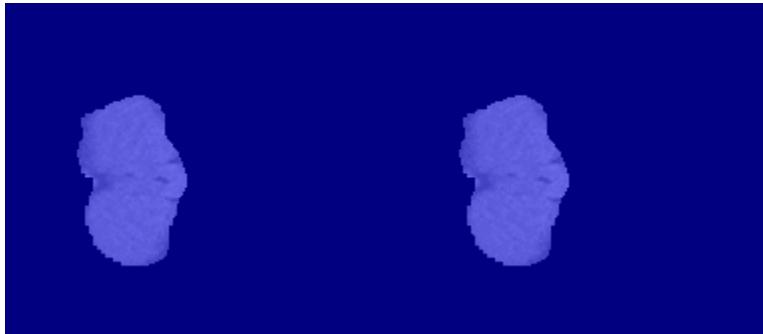
```
viewPnlPred = uipanel.figure,'Title','Predicted Labeled Volume');
hPred = labelvolshow(predictedLabels{volId},vol3d,'Parent',viewPnlPred, ...
    'LabelColor',[0 0 0;1 0 0],'VolumeThreshold',0.68);
```



```
hPred.LabelVisibility(1) = 0;
```



以下图像展示了逐层显示一个图像体切片的过程。标注的真实值在左侧，网络预测在右侧。



### 量化分割准确度

使用 `dice` (Image Processing Toolbox) 函数测量分割准确度。此函数计算预测分割和真实值分割之间的 Dice 相似性系数。

```
diceResult = zeros(length(voldsTest.Files),2);  
  
for j = 1:length(vol)  
    diceResult(j,:) = dice(groundTruthLabels{j},predictedLabels{j});  
end
```

计算整个测试图像体集合的平均 Dice 分数。

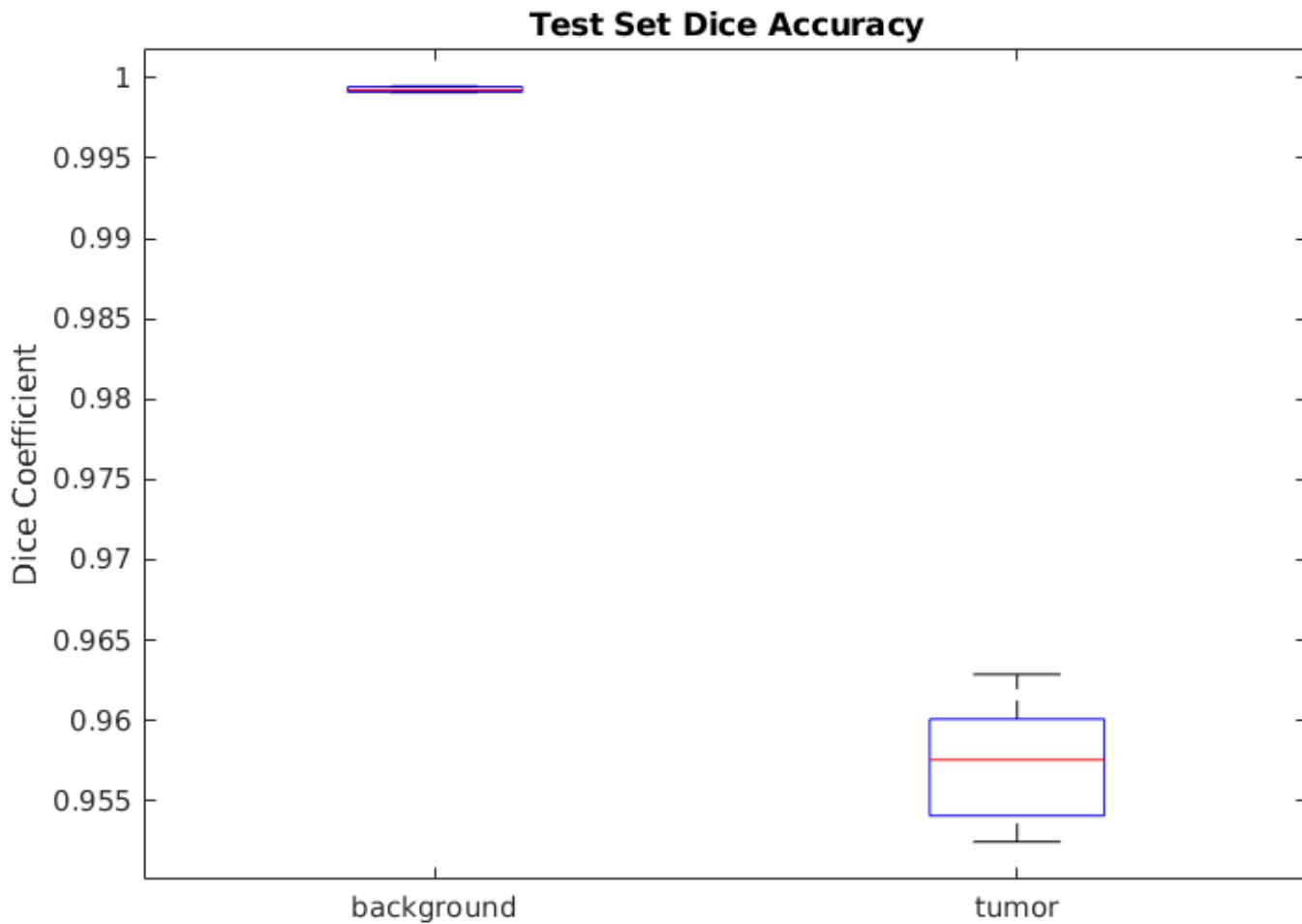
```
meanDiceBackground = mean(diceResult(:,1));
disp(['Average Dice score of background across ',num2str(j), ...
    ' test volumes = ',num2str(meanDiceBackground)])
```

Average Dice score of background across 5 test volumes = 0.9993

```
meanDiceTumor = mean(diceResult(:,2));
disp(['Average Dice score of tumor across ',num2str(j), ...
    ' test volumes = ',num2str(meanDiceTumor)])
```

Average Dice score of tumor across 5 test volumes = 0.9585

下图显示一个 **boxplot** (Statistics and Machine Learning Toolbox)，它可视化包含五个样本测试图像体的集合中 Dice 分数的统计量。图中的红线显示类的 Dice 值中位数。蓝框的上界和下界分别表示第 25 个和第 75 个百分位数。黑色须线会延伸到不是离群值的最远端数据点。



如果您有 Statistics and Machine Learning Toolbox™，则可以使用 **boxplot** 函数来可视化所有测试图像体的 Dice 分数的统计量。要创建一个 **boxplot**，请将以下代码中的 **createBoxplot** 变量设置为 **true**。

```
createBoxplot = false;
if createBoxplot
    figure
```

```

boxplot(diceResult)
title("Test Set Dice Accuracy")
xticklabels(classNames)
ylabel("Dice Coefficient")
end

```

## 参考资料

[1] Çiçek, Ö., A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger."3D U-Net:Learning Dense Volumetric Segmentation from Sparse Annotation."In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2016.Athens, Greece, Oct. 2016, pp. 424-432.

[2] Isensee, F., P. Kickingereder, W. Wick, M. Bendszus, and K. H. Maier-Hein."Brain Tumor Segmentation and Radiomics Survival Prediction:Contribution to the BRATS 2017 Challenge."In Proceedings of BrainLes:International MICCAI Brainlesion Workshop.Quebec City, Canada, Sept. 2017, pp. 287-297.

[3] "Brain Tumours".Medical Segmentation Decathlon. <http://medicaldecathlon.com/>

BraTS 数据集由 Medical Segmentation Decathlon 在 CC-BY-SA 4.0 许可证下提供。所有保证和陈述均有相应的免责声明；有关详细信息，请参阅许可文档。MathWorks® 已修改此示例的下载预训练网络和样本测试集（第 8-0 页）部分中链接的数据集。修改后的样本数据集已裁剪为主要包含大脑和肿瘤的区域，并且通过减去均值并除以裁剪后的大脑区域的标准差，独立地对每个通道进行归一化。

[4] Sudre, C. H., W. Li, T. Vercauteren, S. Ourselin, and M. J. Cardoso."Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations."Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support:Third International Workshop.Quebec City, Canada, Sept. 2017, pp. 240-248.

[5] Ronneberger, O., P. Fischer, and T. Brox."U-Net:Convolutional Networks for Biomedical Image Segmentation."In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015.Munich, Germany, Oct. 2015, pp. 234-241.Available at arXiv:1505.04597.

## 另请参阅

`randomPatchExtractionDatastore` | `trainNetwork` | `trainingOptions` | `transform` |  
`pixelLabelDatastore` | `imageDatastore` | `semanticseg` | `dicePixelClassificationLayer`

## 详细信息

- “Preprocess Volumes for Deep Learning”
- “Datastores for Deep Learning”
- “深度学习层列表”（第 1-18 页）

# 定义使用 Tversky 损失的自定义像素分类层

此示例说明如何定义和创建使用 Tversky 损失的自定义像素分类层。

此层可用于训练语义分割网络。要了解有关创建自定义深度学习层的详细信息，请参阅“[定义自定义深度学习层](#)”（第 17-2 页）。

## Tversky 损失

Tversky 损失基于 Tversky 指数，用于测量两个分割图像之间的重叠 [1 (第 8-0 页)]。一个图像  $Y$  较其对应真实值  $T$  之间的 Tversky 指数  $\text{TI}_c$  由下式给出

$$\text{TI}_c = \frac{\sum_{m=1}^M Y_{cm}T_{cm}}{\sum_{m=1}^M Y_{cm}T_{cm} + \alpha \sum_{m=1}^M Y_{cm}T_{\bar{cm}} + \beta \sum_{m=1}^M Y_{\bar{cm}}T_{cm}}$$

- $c$  对应于类， $\bar{c}$  对应于不在  $c$  类中。
- $M$  是沿  $Y$  的前两个维度的元素数目。
- $\alpha$  和  $\beta$  是控制每个类的假正和假负对损失的贡献的加权因子。

基于类数目  $C$  的损失  $L$  由下式给出

$$L = \sum_{c=1}^C 1 - \text{TI}_c$$

## 分类层模板

在 MATLAB® 中将分类层模板复制到一个新文件中。此模板提供分类层的大致结构，并包括用于定义层行为的函数。示例的后续部分将说明如何完成 `tverskyPixelClassificationLayer`。

```
classdef tverskyPixelClassificationLayer < nnet.layer.ClassificationLayer

    properties
        % Optional properties
    end

    methods

        function loss = forwardLoss(layer, Y, T)
            % Layer forward loss function goes here
        end

    end
end
```

## 声明层属性

默认情况下，自定义输出层具有以下属性：

- **Name** - 层名称，指定为字符向量或字符串标量。要将此层包括在层次图中，您必须指定非空的唯一层名称。如果您使用此层训练串行网络并且 **Name** 设置为 " "，则软件会在训练时自动指定名称。
- **Description** - 层的单行描述，指定为字符向量或字符串标量。当层显示在 **Layer** 数组中时，会出现此描述。如果没有指定层描述，则软件将显示层类名。

- **Type** - 层的类型，指定为字符向量或字符串标量。当层显示在 Layer 数组中时，会显示 Type 的值。如果未指定层类型，则软件将显示 'Classification layer' 或 'Regression layer'。

自定义分类层还具有以下属性：

- **Classes** - 输出层的类，指定为分类向量、字符串数组、字符向量元胞数组或 'auto'。如果 Classes 是 'auto'，则软件会在训练时自动设置类。如果您指定字符串数组或字符向量元胞数组 str，则软件会将输出层的类设置为 categorical(str,str)。默认值为 'auto'。

如果层没有其他属性，则您可以省略 properties 部分。

Tversky 损失必须具有一个小常量值以防止除零错误。指定属性 Epsilon 以保留此值。它还需要两个可变属性：Alpha 和 Beta，分别控制假正和假负的加权。

```
classdef tverskyPixelClassificationLayer < nnet.layer.ClassificationLayer

    properties(Constant)
        % Small constant to prevent division by zero.
        Epsilon = 1e-8;
    end

    properties
        % Default weighting coefficients for false positives and false negatives
        Alpha = 0.5;
        Beta = 0.5;
    end

    ...
end
```

### 创建构造函数

创建用于构造层并初始化层属性的函数。将创建层所必需的所有变量指定为构造函数的输入。

指定可选的输入参数 name，以便在创建层时赋给 Name 属性。

```
function layer = tverskyPixelClassificationLayer(name, alpha, beta)
    % layer = tverskyPixelClassificationLayer(name) creates a Tversky
    % pixel classification layer with the specified name.

    % Set layer name
    layer.Name = name;

    % Set layer properties
    layer.Alpha = alpha;
    layer.Beta = beta;

    % Set layer description
    layer.Description = 'Tversky loss';
end
```

### 创建前向损失函数

创建名为 forwardLoss 的函数，该函数返回网络做出的预测和训练目标之间的加权交叉熵损失。forwardLoss 的语法是 loss = forwardLoss(layer,Y,T)，其中 Y 是前一个层的输出，T 表示训练目标。

对于语义分割问题， $T$  的维度与  $Y$  的维度相匹配，其中  $Y$  是 4 维数组，大小为  $H \times W \times K \times N$ ，其中  $K$  是类的数量， $N$  是小批量大小。

$Y$  的大小取决于前一个层的输出。为了确保  $Y$  与  $T$  的大小相同，您必须在输出层之前包含一个输出正确大小的层。例如，为了确保  $Y$  是  $K$  个类的预测分数的四维数组，您可以包含一个大小为  $K$  的全连接层或一个具有  $K$  个滤波器的卷积层，后跟一个 softmax 层，然后是输出层。

```
function loss = forwardLoss(layer, Y, T)
    % loss = forwardLoss(layer, Y, T) returns the Tversky loss between
    % the predictions Y and the training targets T.

    Penot = 1-Y;
    Gcnot = 1-T;
    TP = sum(sum(Y.*T,1),2);
    FP = sum(sum(Y.*Gcnot,1),2);
    FN = sum(sum(Penot.*T,1),2);

    numer = TP + layer.Epsilon;
    denom = TP + layer.Alpha*FP + layer.Beta*FN + layer.Epsilon;

    % Compute Tversky index
    lossTlc = 1 - numer./denom;
    lossTI = sum(lossTlc,3);

    % Return average Tversky index loss
    N = size(Y,4);
    loss = sum(lossTI)/N;

end
```

## 后向损失函数

由于 `forwardLoss` 函数完全支持自动微分，因此无需为后向损失创建函数。

有关支持自动微分的函数列表，请参阅 “List of Functions with dlarray Support” 。

## 完成的层

`tverskyPixelClassificationLayer.m` 提供了完成的层。

```
classdef tverskyPixelClassificationLayer < nnet.layer.ClassificationLayer
    % This layer implements the Tversky loss function for training
    % semantic segmentation networks.

    % References
    % Salehi, Seyed Sadegh Mohseni, Deniz Erdogmus, and Ali Ghodipour.
    % "Tversky loss function for image segmentation using 3D fully
    % convolutional deep networks." International Workshop on Machine
    % Learning in Medical Imaging. Springer, Cham, 2017.
    % -----
```

```
properties(Constant)
    % Small constant to prevent division by zero.
    Epsilon = 1e-8;
end
```

```
properties
```

```

% Default weighting coefficients for False Positives and False
% Negatives
Alpha = 0.5;
Beta = 0.5;
end

methods

function layer = tverskyPixelClassificationLayer(name, alpha, beta)
    % layer = tverskyPixelClassificationLayer(name, alpha, beta) creates a Tversky
    % pixel classification layer with the specified name and properties alpha and beta.

    % Set layer name.
    layer.Name = name;

    layer.Alpha = alpha;
    layer.Beta = beta;

    % Set layer description.
    layer.Description = 'Tversky loss';
end

function loss = forwardLoss(layer, Y, T)
    % loss = forwardLoss(layer, Y, T) returns the Tversky loss between
    % the predictions Y and the training targets T.

    Penot = 1-Y;
    Genot = 1-T;
    TP = sum(sum(Y.*T,1),2);
    FP = sum(sum(Y.*Genot,1),2);
    FN = sum(sum(Penot.*T,1),2);

    numer = TP + layer.Epsilon;
    denom = TP + layer.Alpha*FP + layer.Beta*FN + layer.Epsilon;

    % Compute tversky index
    lossTIC = 1 - numer./denom;
    lossTI = sum(lossTIC,3);

    % Return average tversky index loss.
    N = size(Y,4);
    loss = sum(lossTI)/N;

end
end
end

```

## GPU 兼容性

tverskyPixelClassificationLayer 中的 forwardLoss 使用的 MATLAB 函数都支持 gpuArray 输入，因此该层与 GPU 兼容。

## 检查输出层有效性

创建层的一个实例。

```
layer = tverskyPixelClassificationLayer('tversky',0.7,0.3);
```

使用 `checkLayer` 检查层的有效性。将有效输入大小指定为层的典型输入的单个观测值的大小。该层需要一个  $H \times W \times K \times N$  数组输入，其中  $K$  是类的数量， $N$  是小批量中的观测值数目。

```
numClasses = 2;
validInputSize = [4 4 numClasses];
checkLayer(layer,validInputSize,'ObservationDimension',4)
```

Skipping GPU tests. No compatible GPU device found.

Skipping code generation compatibility tests. To check validity of the layer for code generation, specify the 'CheckCodegen' argument as true.

Running `nnet.checklayer.TestOutputLayerWithoutBackward`

.....

Done `nnet.checklayer.TestOutputLayerWithoutBackward`

Test Summary:

8 Passed, 0 Failed, 0 Incomplete, 2 Skipped.

Time elapsed: 1.5279 seconds.

测试摘要报告通过、失败、不完整和跳过的测试数量。

### 在语义分割网络中使用自定义层

创建一个使用 `tverskyPixelClassificationLayer` 的语义分割网络。

```
layers = [
    imageInputLayer([32 32 1])
    convolution2dLayer(3,64,'Padding',1)
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding',1)
    reluLayer
    transposedConv2dLayer(4,64,'Stride',2,'Cropping',1)
    convolution2dLayer(1,2)
    softmaxLayer
    tverskyPixelClassificationLayer('tversky',0.3,0.7)]
```

layers =

11x1 Layer array with layers:

```
1 " Image Input      32x32x1 images with 'zerocenter' normalization
2 " Convolution      64 3x3 convolutions with stride [1 1] and padding [1 1 1 1]
3 " Batch Normalization  Batch normalization
4 " ReLU              ReLU
5 " Max Pooling      2x2 max pooling with stride [2 2] and padding [0 0 0 0]
6 " Convolution      64 3x3 convolutions with stride [1 1] and padding [1 1 1 1]
7 " ReLU              ReLU
8 " Transposed Convolution 64 4x4 transposed convolutions with stride [2 2] and cropping [1 1 1 1]
9 " Convolution      2 1x1 convolutions with stride [1 1] and padding [0 0 0 0]
10 " Softmax           softmax
11 'tversky' Classification Output  Tversky loss
```

使用 `imageDatastore` 和 `pixelLabelDatastore` 加载用于语义分割的训练数据。

```
dataSetDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataSetDir,'trainingImages');
```

```

labelDir = fullfile(dataSetDir,'trainingLabels');

imds = imageDatastore(imageDir);

classNames = ["triangle" "background"];
labelIDs = [255 0];
pxds = pixelLabelDatastore(labelDir, classNames, labelIDs);

使用 pixelLabelImageDatastore 关联图像和像素标签数据。

ds = pixelLabelImageDatastore(imds,pxds);

设置训练选项并训练网络。

options = trainingOptions('adam', ...
    'InitialLearnRate',1e-3, ...
    'MaxEpochs',100, ...
    'LearnRateDropFactor',5e-1, ...
    'LearnRateDropPeriod',20, ...
    'LearnRateSchedule','piecewise', ...
    'MiniBatchSize',50);

net = trainNetwork(ds,layers,options);

```

Training on single CPU.

Initializing input data normalization.

Epoch	Iteration	Time Elapsed	Mini-batch	Mini-batch	Base Learning
		(hh:mm:ss)	Accuracy	Loss	Rate
1	1	00:00:01	50.32%	1.2933	0.0010
13	50	00:00:20	98.83%	0.0987	0.0010
25	100	00:00:39	99.33%	0.0547	0.0005
38	150	00:01:04	99.37%	0.0473	0.0005
50	200	00:01:25	99.48%	0.0401	0.0003
63	250	00:01:44	99.48%	0.0384	0.0001
75	300	00:02:05	99.54%	0.0348	0.0001
88	350	00:02:25	99.51%	0.0352	6.2500e-05
100	400	00:02:43	99.56%	0.0330	6.2500e-05

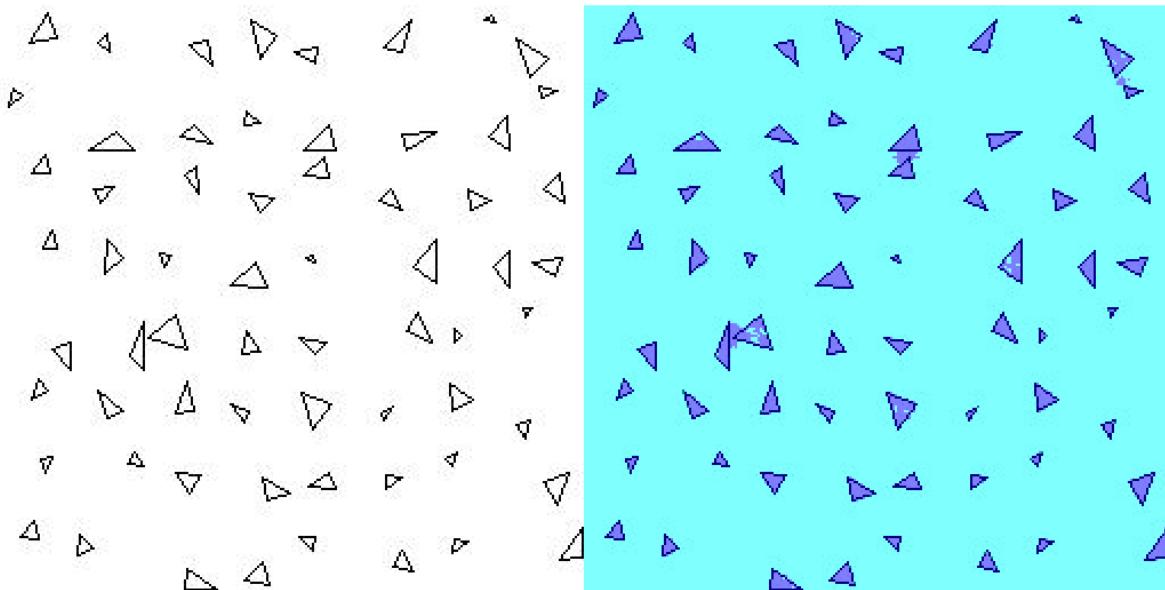
通过分割测试图像并显示分割结果来评估训练的网络。

```

I = imread('triangleTest.jpg');
[C,scores] = semanticseg(I,net);

B = labeloverlay(I,C);
montage({I,B})

```



## 参考资料

[1] Salehi, Seyed Sadegh Mohseni, Deniz Erdogmus, and Ali Gholipour."Tversky loss function for image segmentation using 3D fully convolutional deep networks."International Workshop on Machine Learning in Medical Imaging.Springer, Cham, 2017.

## 另请参阅

[checkLayer](#) | [trainingOptions](#) | [trainNetwork](#) | [pixelLabelDatastore](#) | [semanticseg](#)

## 详细信息

- “[定义自定义深度学习层](#)” (第 17-2 页)
- “[Getting Started with Semantic Segmentation Using Deep Learning](#)” (Computer Vision Toolbox)
- “[使用深度学习进行语义分割](#)” (第 8-11 页)

# 使用 R-CNN 深度学习训练目标检测器

此示例说明如何使用深度学习和 R-CNN（区域卷积神经网络）训练目标检测器。

## 概述

此示例说明如何训练用于检测停车标志的 R-CNN 目标检测器。R-CNN 是一个目标检测框架，它使用卷积神经网络 (CNN) 对图像中的图像区域进行分类 [1]。R-CNN 检测器不使用滑动窗对每个区域进行分类，而是只处理那些可能包含对象的区域。这大幅降低了运行 CNN 时的计算成本。

为了说明如何训练 R-CNN 停车标志检测器，本示例遵循深度学习应用中常用的迁移学习工作流。在迁移学习中，您可以从基于大型图像集合（如 ImageNet [2]）训练的网络开始，处理新的分类或检测任务。使用这种方法的优点是，预训练的网络已学习一系列适用于各种图像的丰富图像特征。这种学习可以通过微调网络迁移到新任务中。通过对权重进行细微调整来微调网络，使得针对原始任务学习的特征表示发生微调以支持新任务。

迁移学习的优点是可以减少训练所需的图像数量和训练时间。为了说明这些优点，此示例使用迁移学习工作流来训练停车标志检测器。首先，使用 CIFAR-10 数据集对一个 CNN 进行预训练，该数据集有 50,000 个训练图像。然后，只使用 41 个训练图像针对停车标志检测对这个预训练的 CNN 进行微调。如果没有预训练 CNN，训练停车标志检测器会需要更多图像。

注意：此示例需要 Computer Vision Toolbox™、Image Processing Toolbox™、Deep Learning Toolbox™ 和 Statistics and Machine Learning Toolbox™。

强烈推荐使用支持 CUDA 的 NVIDIA™ GPU 来运行此示例。使用 GPU 需要 Parallel Computing Toolbox™。有关支持的计算功能的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。

## 下载 CIFAR-10 图像数据

下载 CIFAR-10 数据集 [1]。此数据集包含 50,000 个训练图像，将用于训练 CNN。

将 CIFAR-10 数据下载到一个临时目录

```
cifar10Data = tempdir;
url = 'https://www.cs.toronto.edu/~kriz/cifar-10-matlab.tar.gz';
helperCIFAR10Data.download(url,cifar10Data);
```

加载 CIFAR-10 训练和测试数据。

```
[trainingImages,trainingLabels,testImages,testLabels] = helperCIFAR10Data.load(cifar10Data);
```

每个图像是一个  $32 \times 32$  RGB 图像，共有 50,000 个训练样本。

```
size(trainingImages)
```

```
ans = 1×4
```

```
32      32      3    50000
```

CIFAR-10 有 10 个图像类别。列出图像类别：

```
numImageCategories = 10;
categories(trainingLabels)
```

```
ans = 10×1 cell
  {'airplane' }
  {'automobile'}
  {'bird'   }
  {'cat'    }
  {'deer'   }
  {'dog'    }
  {'frog'   }
  {'horse'  }
  {'ship'   }
  {'truck'  }
```

您可以使用以下代码显示一些训练图像。

```
figure
thumbnails = trainingImages(:, :, 1:100);
montage(thumbnails)
```

### 创建卷积神经网络 (CNN)

CNN 由一系列层组成，每层定义一项特定计算。Deep Learning Toolbox™ 的一些功能可以帮助您轻松逐层设计 CNN。本示例使用以下层创建 CNN：

- **imageInputLayer** - 图像输入层
- **convolution2dLayer** - 卷积神经网络的二维卷积层
- **reluLayer** - 修正线性单元 (ReLU) 层
- **maxPooling2dLayer** - 最大池化层
- **fullyConnectedLayer** - 全连接层
- **softmaxLayer** - Softmax 层
- **classificationLayer** - 神经网络的分类输出层

此处定义的网络类似于 [4] 中所述的网络，以 **imageInputLayer** 开头。输入层定义 CNN 可以处理的数据的类型和大小。在本示例中，CNN 用于处理 CIFAR-10 图像，它们是 32×32 RGB 图像：

```
% Create the image input layer for 32x32x3 CIFAR-10 images.
[height,width,numChannels, ~] = size(trainingImages);

imageSize = [height width numChannels];
inputLayer = imageInputLayer(imageSize)

inputLayer =
  ImageInputLayer with properties:
    Name: ''
    InputSize: [32 32 3]
  Hyperparameters
    DataAugmentation: 'none'
    Normalization: 'zerocenter'
    NormalizationDimension: 'auto'
    Mean: []
```

接下来，定义网络的中间层。中间层包含多个由卷积层、ReLU（修正线性单元）层和池化层组成的重复块。这三个层构成卷积神经网络的核心构建块。卷积层定义滤波器权重集，这些权重集在网络训练期间会更新。ReLU 层在网络中引入非线性，让网络能够逼近非线性函数，这些函数将图像像素映射到图像语义内

容。池化层在数据流经网络时对其进行下采样。在具有许多层的网络中，应谨慎使用池化层，以避免过早对网络中的数据进行下采样。

```
% Convolutional layer parameters
filterSize = [5 5];
numFilters = 32;

middleLayers = [

    % The first convolutional layer has a bank of 32 5x5x3 filters. A
    % symmetric padding of 2 pixels is added to ensure that image borders
    % are included in the processing. This is important to avoid
    % information at the borders being washed away too early in the
    % network.
    convolution2dLayer(filterSize,numFilters,'Padding',2)

    % Note that the third dimension of the filter can be omitted because it
    % is automatically deduced based on the connectivity of the network. In
    % this case because this layer follows the image layer, the third
    % dimension must be 3 to match the number of channels in the input
    % image.

    % Next add the ReLU layer:
    reluLayer()

    % Follow it with a max pooling layer that has a 3x3 spatial pooling area
    % and a stride of 2 pixels. This down-samples the data dimensions from
    % 32x32 to 15x15.
    maxPooling2dLayer(3,'Stride',2)

    % Repeat the 3 core layers to complete the middle of the network.
    convolution2dLayer(filterSize,numFilters,'Padding',2)
    reluLayer()
    maxPooling2dLayer(3, 'Stride',2)

    convolution2dLayer(filterSize,2 * numFilters,'Padding',2)
    reluLayer()
    maxPooling2dLayer(3,'Stride',2)

]

middleLayers =
9x1 Layer array with layers:

1 " Convolution 32 5x5 convolutions with stride [1 1] and padding [2 2 2 2]
2 " ReLU      ReLU
3 " Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
4 " Convolution 32 5x5 convolutions with stride [1 1] and padding [2 2 2 2]
5 " ReLU      ReLU
6 " Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
7 " Convolution 64 5x5 convolutions with stride [1 1] and padding [2 2 2 2]
8 " ReLU      ReLU
9 " Max Pooling 3x3 max pooling with stride [2 2] and padding [0 0 0 0]
```

通过重复这三个基本层，可以创建更深的网络。但是，应该减少池化层的数量，以避免过早地对数据进行下采样。在网络中过早进行下采样会丢弃对学习有用的图像信息。

CNN 的最终层通常包括全连接层和 softmax 损失层。

```

finalLayers = [ % Add a fully connected layer with 64 output neurons. The output size of
% this layer will be an array with a length of 64.
fullyConnectedLayer(64)

% Add an ReLU non-linearity.
reluLayer

% Add the last fully connected layer. At this point, the network must
% produce 10 signals that can be used to measure whether the input image
% belongs to one category or another. This measurement is made using the
% subsequent loss layers.
fullyConnectedLayer(numImageCategories)

% Add the softmax loss layer and classification layer. The final layers use
% the output of the fully connected layer to compute the categorical
% probability distribution over the image classes. During the training
% process, all the network weights are tuned to minimize the loss over this
% categorical distribution.
softmaxLayer
classificationLayer
]

finalLayers =
5x1 Layer array with layers:

1 " Fully Connected    64 fully connected layer
2 " ReLU              ReLU
3 " Fully Connected    10 fully connected layer
4 " Softmax            softmax
5 " Classification Output crossentropyex

```

对输入层、中间层和最终层进行合并。

```

layers = [
    inputLayer
    middleLayers
    finalLayers
]

layers =
15x1 Layer array with layers:

1 " Image Input      32x32x3 images with 'zerocenter' normalization
2 " Convolution       32 5x5 convolutions with stride [1 1] and padding [2 2 2]
3 " ReLU              ReLU
4 " Max Pooling       3x3 max pooling with stride [2 2] and padding [0 0 0 0]
5 " Convolution       32 5x5 convolutions with stride [1 1] and padding [2 2 2 2]
6 " ReLU              ReLU
7 " Max Pooling       3x3 max pooling with stride [2 2] and padding [0 0 0 0]
8 " Convolution       64 5x5 convolutions with stride [1 1] and padding [2 2 2 2]
9 " ReLU              ReLU
10 " Max Pooling      3x3 max pooling with stride [2 2] and padding [0 0 0 0]
11 " Fully Connected   64 fully connected layer
12 " ReLU              ReLU
13 " Fully Connected   10 fully connected layer
14 " Softmax            softmax
15 " Classification Output crossentropyex

```

使用标准差为 0.0001 的正态分布随机数初始化第一个卷积层的权重。这有助于改善训练的收敛性。

```
layers(2).Weights = 0.0001 * randn([filterSize numChannels numFilters]);
```

### 使用 CIFAR-10 数据训练 CNN

完成网络架构定义后，就可以使用 CIFAR-10 训练数据对其进行训练。首先，使用 `trainingOptions` 函数设置网络训练算法。网络训练算法使用具有动量的随机梯度下降 (SGDM)，初始学习率为 0.001。在训练期间，初始学习率每 8 轮降低一次（1 轮定义为对整个训练数据集进行一次完整遍历）。训练算法运行 40 轮。

请注意，训练算法使用包含 128 个图像的小批量。如果使用 GPU 进行训练，则由于 GPU 上的内存约束，可能需要减少一些图像。

```
% Set the network training options
opts = trainingOptions('sgdm',...
    'Momentum', 0.9, ...
    'InitialLearnRate', 0.001, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.1, ...
    'LearnRateDropPeriod', 8, ...
    'L2Regularization', 0.004, ...
    'MaxEpochs', 40, ...
    'MiniBatchSize', 128, ...
    'Verbose', true);
```

使用 `trainNetwork` 函数训练网络。这是一个计算密集型过程，需要 20-30 分钟才能完成。为了在运行此示例时节省时间，此处磁盘加载预训练网络。如果您要自己训练网络，请将下列代码中的 `doTraining` 变量设置为 `true`。

请注意，强烈推荐使用支持 CUDA 的 NVIDIA™ GPU 进行训练。

```
% A trained network is loaded from disk to save time when running the
% example. Set this flag to true to train the network.
doTraining = false;

if doTraining
    % Train a network.
    cifar10Net = trainNetwork(trainingImages, trainingLabels, layers, opts);
else
    % Load pre-trained detector for the example.
    load('rcnnStopSigns.mat','cifar10Net')
end
```

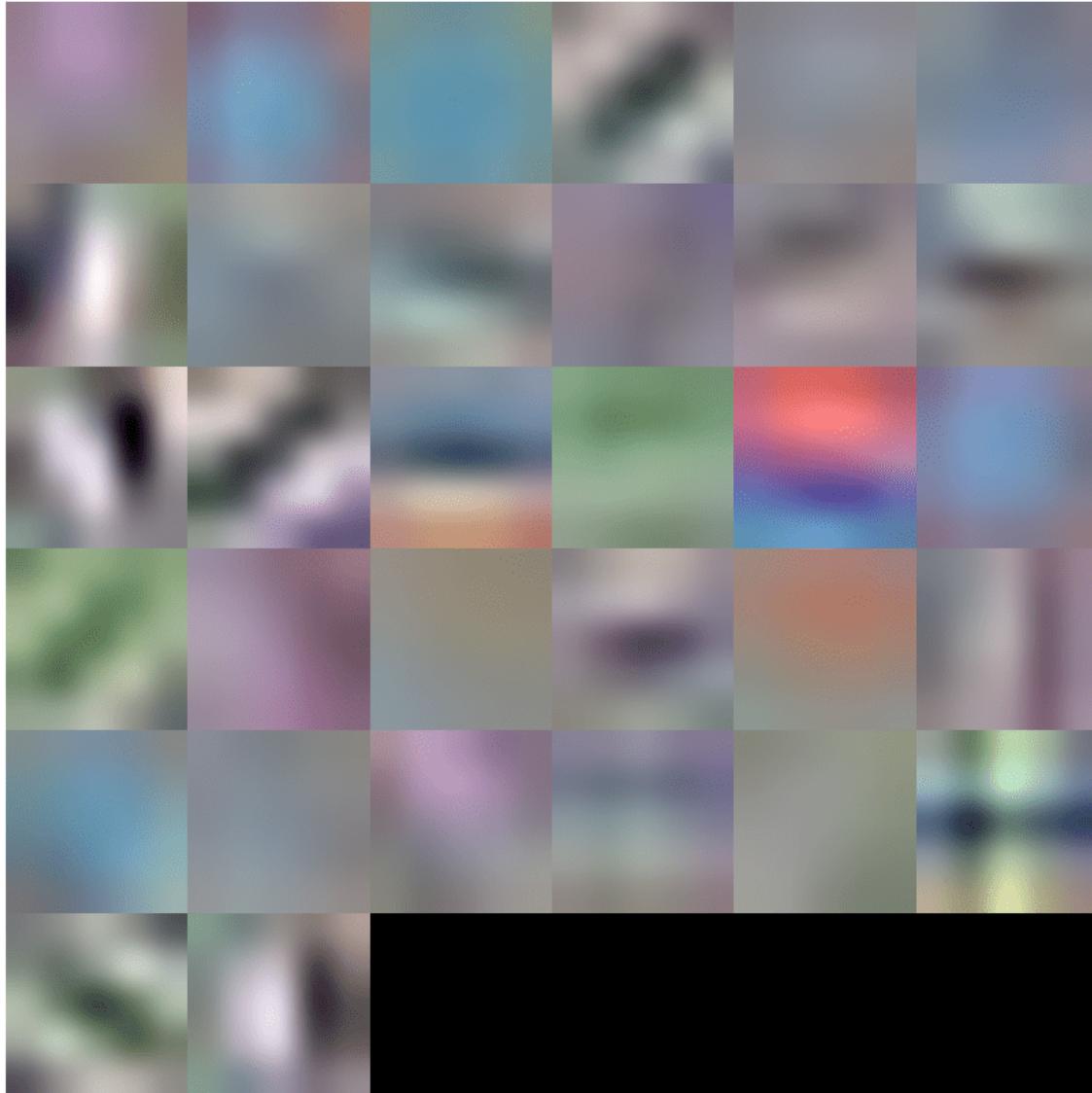
### 验证 CIFAR-10 网络训练

在网络经过训练后，应对其进行验证，以确保训练是成功的。首先，快速可视化第一个卷积层的滤波器权重有助于识别训练中的任何直接问题。

```
% Extract the first convolutional layer weights
w = cifar10Net.Layers(2).Weights;

% rescale the weights to the range [0, 1] for better visualization
w = rescale(w);

figure
montage(w)
```



第一层权重应该有一些明确定义的结构。如果权重看起来仍然是随机的，则表明网络可能需要额外的训练。在本例中，如上所示，第一层滤波器已从 CIFAR-10 训练数据中学习到类似边缘的特征。

要完全验证训练结果，请使用 CIFAR-10 测试数据来测量网络的分类准确度。低准确度分数表示需要更多训练或训练数据。此示例的目标并不是在测试集上实现 100% 的准确度，而是充分训练网络以用于训练目标检测器。

```
% Run the network on the test set.  
YTest = classify(cifar10Net, testImages);  
  
% Calculate the accuracy.  
accuracy = sum(YTest == testLabels)/numel(testLabels)
```

```
accuracy = 0.7456
```

进一步的训练能提高准确度，但这对于训练 R-CNN 目标检测器来说不是必需的。

### 加载训练数据

该网络在 CIFAR-10 分类任务中表现良好，因此可以使用迁移学习方法来微调网络，以便检测停车标志。

首先加载停车标志的真实值数据。

```
% Load the ground truth data
data = load('stopSignsAndCars.mat', 'stopSignsAndCars');
stopSignsAndCars = data.stopSignsAndCars;

% Update the path to the image files to match the local file system
visiondata = fullfile(toolboxdir('vision'), 'visiondata');
stopSignsAndCars.imageFilename = fullfile(visiondata, stopSignsAndCars.imageFilename);

% Display a summary of the ground truth data
summary(stopSignsAndCars)
```

Variables:

```
imageFilename: 41×1 cell array of character vectors
stopSign: 41×1 cell
carRear: 41×1 cell
carFront: 41×1 cell
```

训练数据包含在一个表中，该表包含停车标志、车头和车尾的图像文件名和关注区域标签。每个关注区域标签是图像内关注对象周围的边界框。训练停车标志检测器时，只需要停车标志的关注区域标签。您必须删除车头和车尾的关注区域标签：

```
% Only keep the image file names and the stop sign ROI labels
stopSigns = stopSignsAndCars(:, {'imageFilename', 'stopSign'});

% Display one training image and the ground truth bounding boxes
I = imread(stopSigns.imageFilename{1});
I = insertObjectAnnotation(I, 'Rectangle', stopSigns.stopSign{1}, 'stop sign', 'LineWidth', 8);

figure
imshow(I)
```



请注意，此数据集中只有 41 个训练图像。仅使用 41 个图像从零开始训练一个 R-CNN 目标检测器是不切实际的，也不会得到可靠的停车标志检测器。由于此示例中停车标志检测器是通过微调已基于较大数据集（CIFAR-10 有 50000 个训练图像）预训练的网络来训练的，因此使用小得多的数据集是可行的。

### 训练 R-CNN 停车标志检测器

最后，使用 `trainRCNNObjectDetector` (Computer Vision Toolbox) 训练 R-CNN 目标检测器。该函数的输入是真实值表，其中包含标注的停车标志图像、预训练的 CIFAR-10 网络和训练选项。原始 CIFAR-10 网络将图像分为 10 个类，训练函数自动对该网络进行修改以使图像分为 2 个类：停车标志类和一般背景类。

在训练期间，使用从真实值数据中提取的图像补片来微调输入网络权重。`'PositiveOverlapRange'` 和 `'NegativeOverlapRange'` 参数控制哪些图像补片用于训练。正训练样本是与真实值框的重叠率为 0.5 到 1.0 的样本（按边界框交并比测量）。负训练样本是重叠率为 0 到 0.3 的样本。这些参数的最佳值应该通过在验证集上测试经过训练的检测器来选择。

对于 R-CNN 训练，**强烈推荐使用 MATLAB 工作进程的并行池来缩短训练时间。**  
`trainRCNNObjectDetector` 会根据您的并行预设项设置自动创建和使用并行池。确保在训练前启用并行池的使用。

为了在运行此示例时节省时间，此处磁盘加载预训练网络。如果您要自己训练网络，请将下列代码中的 `doTraining` 变量设置为 `true`。

请注意，强烈推荐使用支持 CUDA 的 NVIDIA™ GPU 进行训练。

```
% A trained detector is loaded from disk to save time when running the
% example. Set this flag to true to train the detector.
doTraining = false;
```

```

if doTraining

    % Set training options
    options = trainingOptions('sgdm', ...
        'MiniBatchSize', 128, ...
        'InitialLearnRate', 1e-3, ...
        'LearnRateSchedule', 'piecewise', ...
        'LearnRateDropFactor', 0.1, ...
        'LearnRateDropPeriod', 100, ...
        'MaxEpochs', 100, ...
        'Verbose', true);

    % Train an R-CNN object detector. This will take several minutes.
    rcnn = trainRCNNObjectDetector(stopSigns, cifar10Net, options, ...
        'NegativeOverlapRange', [0 0.3], 'PositiveOverlapRange',[0.5 1])
else
    % Load pre-trained network for the example.
    load('rcnnStopSigns.mat','rcnn')
end

```

### 测试 R-CNN 停车标志检测器

R-CNN 目标检测器现在可用于检测图像中的停车标志。对测试图像试用该检测器：

```

% Read test image
testImage = imread('stopSignTest.jpg');

% Detect stop signs
[bboxes,score,label] = detect(rcnn,testImage,'MiniBatchSize',128)

bboxes = 1×4
    419   147   31   20

score = single
    0.9955

label = categorical categorical
    stopSign

```

R-CNN 对象 **detect** (Computer Vision Toolbox) 方法返回每个检测的对象边界框、检测分数和类标签。标签在检测多个对象时很有用，例如停车标志、让路标志或限速标志。分数（范围在 0 和 1 之间）表示对检测的信心，可以据此忽略得分较低的检测。

```

% Display the detection results
[score, idx] = max(score);

bbox = bboxes(idx, :);
annotation = sprintf("%s: (%Confidence = %f)", label(idx), score);

outputImage = insertObjectAnnotation(testImage, 'rectangle', bbox, annotation);

figure
imshow(outputImage)

```



## 调试技巧

R-CNN 检测器中使用的网络也可用于处理整个测试图像。直接处理大于网络输入大小的整个图像时，会生成分类分数的二维热图。这是有用的调试工具，因为它有助于识别图像中让网络产生混淆的物体，还可能有助于深入了解如何改进训练。

```
% The trained network is stored within the R-CNN detector
rcnn.Network
```

```
ans =
SeriesNetwork with properties:
```

```
Layers: [15×1 nnet.cnn.layer.Layer]
```

从 softmax 层（网络中的第 14 个层）提取 activations。这些是网络在扫描图像时产生的分类分数。

```
featureMap = activations(rcnn.Network, testImage, 14);
```

```
% The softmax activations are stored in a 3-D array.
size(featureMap)
```

```
ans = 1×3
```

```
43    78    2
```

featureMap 中的第三个维度对应于对象类。

```
rcnn.ClassNames
```

```
ans = 2×1 cell
{'stopSign' }
```

```
{'Background'}
```

停车标志特征图存储在第一个通道中。

```
stopSignMap = featureMap(:, :, 1);
```

由于网络中的下采样操作，激活输出的大小小于输入图像。要生成更好的可视化效果，请将 `stopSignMap` 调整为输入图像的大小。这是非常粗略的近似，它将激活区域映射到图像像素，并且只能用于说明目的。

```
% Resize stopSignMap for visualization  
[height, width, ~] = size(testImage);  
stopSignMap = imresize(stopSignMap, [height, width]);  
  
% Visualize the feature map superimposed on the test image.  
featureMapOnImage = imfuse(testImage, stopSignMap);  
  
figure  
imshow(featureMapOnImage)
```



测试图像中的停车标志与网络的激活函数中的最大峰值准确对应。这验证了 R-CNN 检测器所用的 CNN 确实已学会识别停车标志。如果有其他峰值，则可能表明训练需要更多负数据来帮助防止误报。如果是这样的话，则您可以增大 `trainingOptions` 中的 'MaxEpochs' 并重新训练。

## 总结

此示例说明如何使用基于 CIFAR-10 数据训练的网络来训练一个 R-CNN 停车标志目标检测器。可以遵循类似的步骤来使用深度学习训练其他目标检测器。

## 参考资料

- [1] Girshick, R., J. Donahue, T. Darrell, and J. Malik."Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation."Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition.Columbus, OH, June 2014, pp. 580-587.
- [2] Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei."ImageNet:A Large-Scale Hierarchical Image Database."Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition.Miami, FL, June 2009, pp. 248-255.
- [3] Krizhevsky, A., and G. Hinton."Learning multiple layers of features from tiny images."Master's Thesis.University of Toronto, Toronto, Canada, 2009.
- [4] <https://code.google.com/p/cuda-convnet/>

## 另请参阅

`rcnnObjectDetector` | `trainingOptions` | `trainNetwork` | `trainRCNNObjectDetector` |  
`fastRCNNObjectDetector` | `fasterRCNNObjectDetector` | `trainFastRCNNObjectDetector` |  
`trainFasterRCNNObjectDetector` | `classify` | `detect` | `activations`

## 详细信息

- "使用 Faster R-CNN 深度学习进行目标检测" (第 8-78 页)
- "Semantic Segmentation" (Computer Vision Toolbox)
- "Object Detection" (Computer Vision Toolbox)

## 使用 Faster R-CNN 深度学习进行目标检测

此示例说明如何训练 Faster R-CNN（区域卷积神经网络）目标检测器。

深度学习是一种功能强大的机器学习方法，可用于训练稳健的目标检测器。目标检测深度学习有多种方法，包括 Faster R-CNN 和 you only look once (YOLO) v2。此示例使用 `trainFasterRCNNObjectDetector` 函数训练 Faster R-CNN 车辆检测器。有关详细信息，请参阅“Object Detection” (Computer Vision Toolbox)。

### 下载预训练的检测器

下载预训练的检测器，避免在训练上花费时间。如果要训练检测器，请将 `doTraining` 变量设置为 `true`。

```
doTraining = false;
if ~doTraining && ~exist('fasterRCNNResNet50EndToEndVehicleExample.mat','file')
    disp('Downloading pretrained detector (118 MB)...');
    pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/fasterRCNNResNet50EndToEndVehicleExample.mat';
    websave('fasterRCNNResNet50EndToEndVehicleExample.mat',pretrainedURL);
end
```

### 加载数据集

此示例使用包含 295 个图像的小型标注数据集。其中许多图像来自加州理工学院的 Caltech Cars 1999 和 2001 数据集，可在加州理工学院计算视觉网站上获得，该网站由 Pietro Perona 创建并经许可使用。每个图像包含一个或两个带标签的车辆实例。您可以借助小型数据集探索 Faster R-CNN 的训练过程，但在实际应用中，您需要更多带标签的图像来训练稳健的检测器。解压缩车辆图像并加载车辆真实值数据。

```
unzip vehicleDatasetImages.zip
data = load('vehicleDatasetGroundTruth.mat');
vehicleDataset = data.vehicleDataset;
```

车辆数据存储在一个包含两列的表中，其中第一列包含图像文件路径，第二列包含车辆边界框。

将数据集分成训练集、验证集和测试集。选择 60% 的数据用于训练，10% 用于验证，其余用于测试经过训练的检测器。

```
rng(0)
shuffledIndices = randperm(height(vehicleDataset));
idx = floor(0.6 * height(vehicleDataset));

trainingIdx = 1:idx;
trainingDataTbl = vehicleDataset(shuffledIndices(trainingIdx),:);

validationIdx = idx+1 : idx + 1 + floor(0.1 * length(shuffledIndices) );
validationDataTbl = vehicleDataset(shuffledIndices(validationIdx),:);

testIdx = validationIdx(end)+1 : length(shuffledIndices);
testDataTbl = vehicleDataset(shuffledIndices(testIdx),:);
```

使用 `imageDatastore` 和 `boxLabelDatastore` 创建数据存储，以便在训练和评估期间加载图像和标签数据。

```
imdsTrain = imageDatastore(trainingDataTbl{,:'imageFilename'});
bldsTrain = boxLabelDatastore(trainingDataTbl(:, 'vehicle'));

imdsValidation = imageDatastore(validationDataTbl{,:'imageFilename'});
bldsValidation = boxLabelDatastore(validationDataTbl(:, 'vehicle'));
```

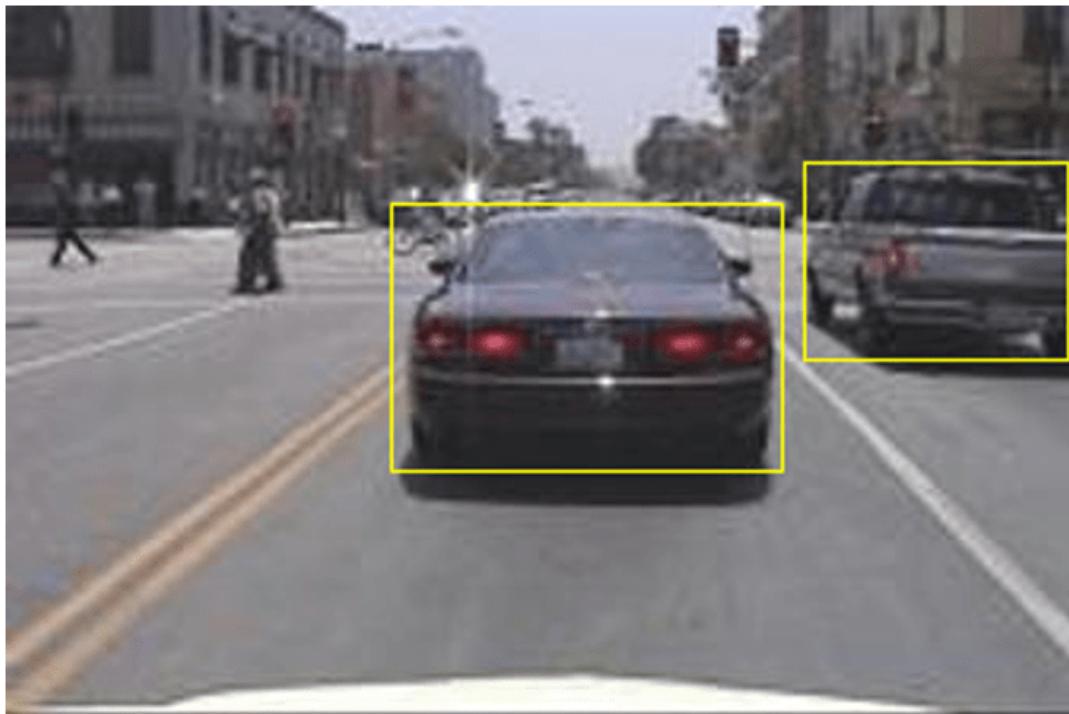
```
imdsTest = imageDatastore(testDataTbl{,:'imageFilename'});  
bldsTest = boxLabelDatastore(testDataTbl{,:'vehicle'});
```

组合图像和边界框标签数据存储。

```
trainingData = combine(imdsTrain,bldsTrain);  
validationData = combine(imdsValidation,bldsValidation);  
testData = combine(imdsTest,bldsTest);
```

显示其中一个训练图像和边界框标签。

```
data = read(trainingData);  
I = data{1};  
bbox = data{2};  
annotatedImage = insertShape(I,'Rectangle',bbox);  
annotatedImage = imresize(annotatedImage,2);  
figure  
imshow(annotatedImage)
```



### 创建 Faster R-CNN 检测网络

Faster R-CNN 目标检测网络由一个特征提取网络后跟两个子网络组成。特征提取网络通常是一个预训练的 CNN，如 ResNet-50 或 Inception v3。特征提取网络之后的第一个子网络是区域提议网络 (RPN)，该网络经训练用于生成目标提议，即图像中可能存在目标的区域。对第二个子网络进行训练来预测每个目标提议的实际类。

特征提取网络通常是一个预训练的 CNN（有关详细信息，请参阅“预训练的深度神经网络”（第 1-8 页））。此示例使用 ResNet-50 进行特征提取。根据应用要求，也可以使用其他预训练网络，如 MobileNet v2 或 ResNet-18。

使用 **fasterRCNNLayers** 自动根据预训练的特征提取网络创建 Faster R-CNN 网络。**fasterRCNNLayers** 要求您指定几个用于参数化 Faster R-CNN 网络的输入：

- 网络输入大小
- 锚框
- 特征提取网络

首先，指定网络输入大小。选择网络输入大小时，请考虑运行网络本身所需的最低大小、训练图像的大小以及基于所选大小处理数据所产生的计算成本。如果可行，请选择接近训练图像大小且大于网络所需输入大小的网络输入大小。为了降低运行示例的计算成本，请指定网络输入大小为 [224 224 3]，这是运行网络所需的最低大小。

```
inputSize = [224 224 3];
```

请注意，此示例中使用的训练图像大于  $224 \times 224$ ，并且大小不同，因此您必须在训练前的预处理步骤中调整图像的大小。

接下来，使用 **estimateAnchorBoxes** 根据训练数据中目标的大小来估计锚框。考虑到训练前会对图像大小进行调整，用来估计锚框的训练数据的大小也要调整。使用 **transform** 预处理训练数据，然后定义锚框数量并估计锚框。

```
preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));
numAnchors = 3;
anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)
```

anchorBoxes = 3×2

```
29 17
46 39
136 116
```

有关选择锚框的详细信息，请参阅“Estimate Anchor Boxes From Training Data”（Computer Vision Toolbox）(Computer Vision Toolbox™) 和“Anchor Boxes for Object Detection”（Computer Vision Toolbox）。

现在，使用 **resnet50** 加载预训练的 ResNet-50 模型。

```
featureExtractionNetwork = resnet50;
```

选择 '**activation\_40\_relu**' 作为特征提取层。此特征提取层输出以 16 为因子的下采样特征图。该下采样量是空间分辨率和提取特征强度之间一个很好的折中，因为在网络更深层提取的特征能够对更强的图像特征进行编码，但以空间分辨率为代价。选择最佳特征提取层需要依靠经验分析。您可以使用 **analyzeNetwork** 查找网络中其他潜在特征提取层的名称。

```
featureLayer = 'activation_40_relu';
```

定义要检测的类的数量。

```
numClasses = width(vehicleDataset)-1;
```

创建 Faster R-CNN 目标检测网络。

```
lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);
```

您可以使用 `analyzeNetwork` 或 Deep Learning Toolbox™ 中的深度网络设计器来可视化网络。

如果需要对 Faster R-CNN 网络架构进行更多控制，请使用深度网络设计器手动设计 Faster R-CNN 检测网络。有关详细信息，请参阅 “Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN” (Computer Vision Toolbox)。

## 数据增强

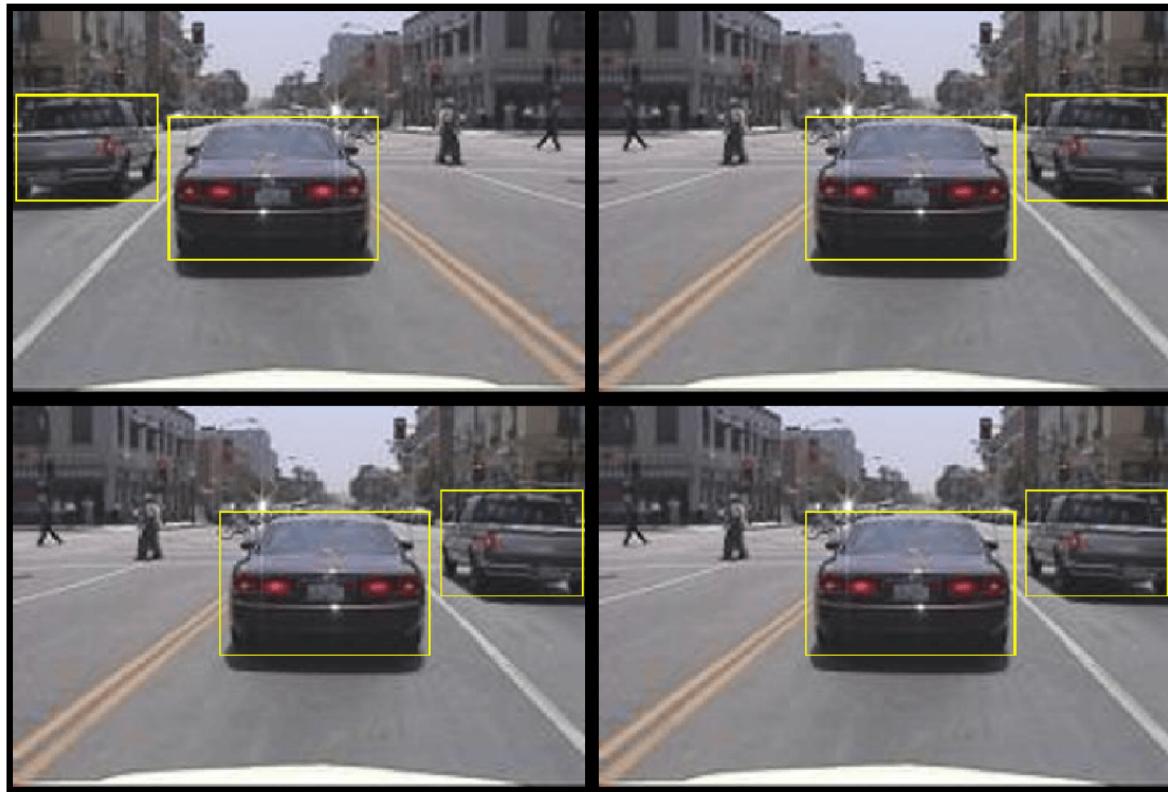
数据增强可通过在训练期间随机变换原始数据来提高网络准确度。通过使用数据增强，您可以为训练数据添加更多变化，但又不必增加带标签的训练样本的数量。

使用 `transform` 通过随机水平翻转图像和相关边界框标签来增强训练数据。请注意，数据增强不适用于测试数据和验证数据。理想情况下，测试数据和验证数据代表原始数据并且保持不变，以便进行无偏置的评估。

```
augmentedTrainingData = transform(trainingData,@augmentData);
```

多次读取同一图像，并显示增强的训练数据。

```
augmentedData = cell(4,1);
for k = 1:4
    data = read(augmentedTrainingData);
    augmentedData{k} = insertShape(data{1},'Rectangle',data{2});
    reset(augmentedTrainingData);
end
figure
montage(augmentedData,'BorderSize',10)
```



### 预处理训练数据

预处理增强的训练数据和验证数据以准备进行训练。

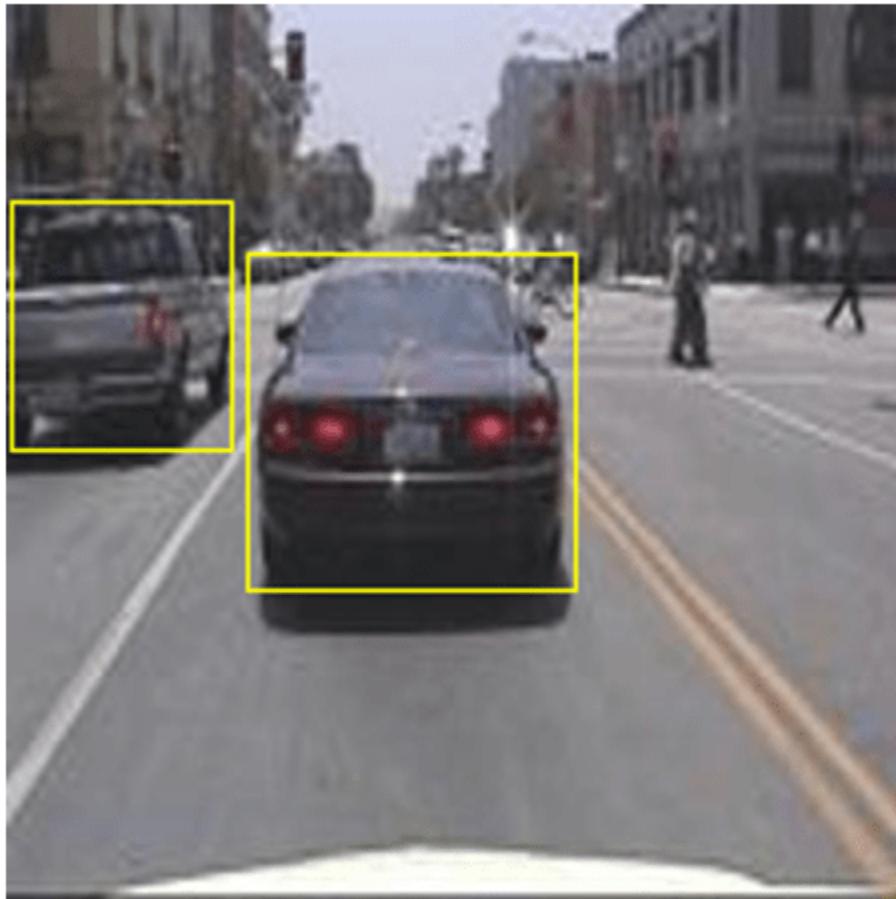
```
trainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));
validationData = transform(validationData,@(data)preprocessData(data,inputSize));
```

读取预处理的数据。

```
data = read(trainingData);
```

显示图像和边界框。

```
I = data{1};
bbox = data{2};
annotatedImage = insertShape(I,'Rectangle',bbox);
annotatedImage = imresize(annotatedImage,2);
figure
imshow(annotatedImage)
```



## 训练 Faster R-CNN

使用 `trainingOptions` 指定网络训练选项。将 '`ValidationData`' 设置为经过预处理的验证数据。将 '`CheckpointPath`' 设置为临时位置。这样可在训练过程中保存经过部分训练的检测器。如果由于停电或系统故障等原因导致训练中断，您可以从保存的检查点继续训练。

```
options = trainingOptions('sgdm',...
    'MaxEpochs',10,...
    'MiniBatchSize',2,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir,...
    'ValidationData',validationData);
```

如果 `doTraining` 为 `true`，则使用 `trainFasterRCNNObjectDetector` 训练 Faster R-CNN 目标检测器。否则，加载预训练的网络。

```
if doTraining
    % Train the Faster R-CNN detector.
    % * Adjust NegativeOverlapRange and PositiveOverlapRange to ensure
```

```
% that training samples tightly overlap with ground truth.
[detector, info] = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...
    'NegativeOverlapRange',[0 0.3], ...
    'PositiveOverlapRange',[0.6 1]);
else
    % Load pretrained detector for the example.
    pretrained = load('fasterRCNNResNet50EndToEndVehicleExample.mat');
    detector = pretrained.detector;
end
```

此示例在具有 12 GB 内存的 Nvidia(TM) Titan X GPU 上进行了验证。训练网络需要大约 20 分钟。具体训练时间因您使用的硬件而异。

对一个测试图像运行检测器以进行快速检查。确保将图像的大小调整为与训练图像相同。

```
I = imread(testDataTbl.imageFilename{3});
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);
```

显示结果。

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```



### 使用测试集评估检测器

基于大量图像评估经过训练的目标检测器以测量其性能。Computer Vision Toolbox™ 提供目标检测器评估函数，用于测量常见指标，如平均精确率 (evaluateDetectionPrecision) 和对数平均泄漏检率 (evaluateDetectionMissRate)。对于此示例，使用平均精确率指标来评估性能。平均准确率提供单一数字，该数字综合反映了检测器进行正确分类的能力（精确率）和检测器找到所有相关对象的能力（召回率）。

将应用于训练数据的同一预处理变换应用于测试数据。

```
testData = transform(testData,@(data)preprocessData(data,inputSize));
```

对所有测试图像运行检测器。

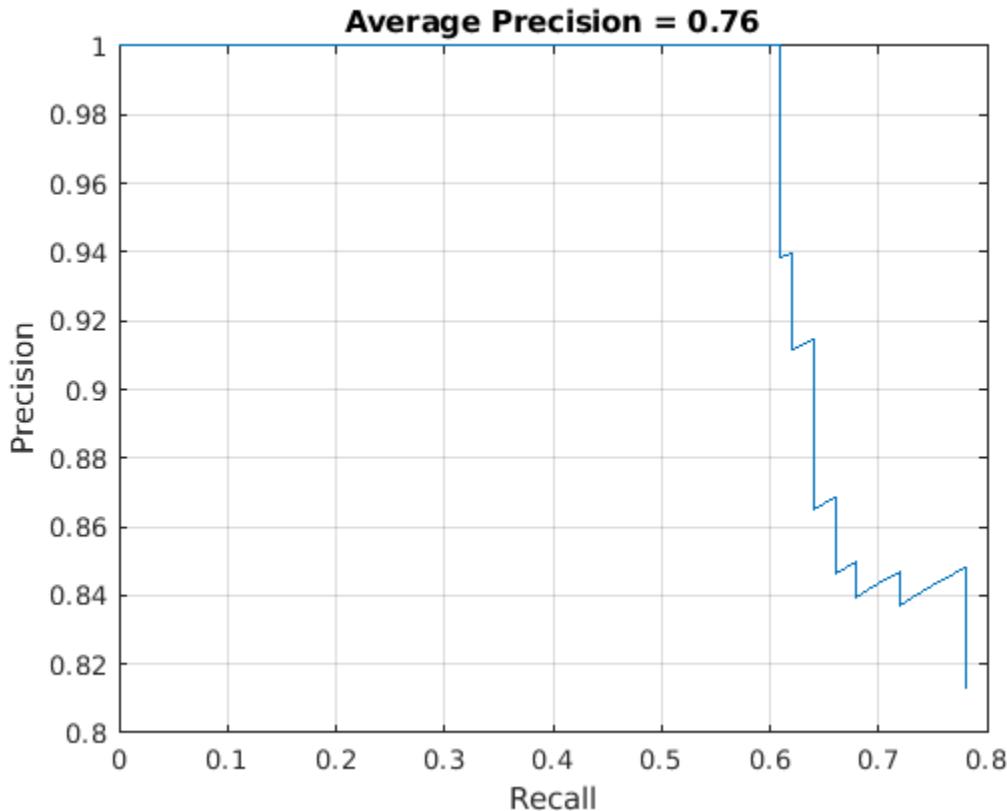
```
detectionResults = detect(detector,testData,'MinibatchSize',4);
```

使用平均精确率指标评估目标检测器。

```
[ap, recall, precision] = evaluateDetectionPrecision(detectionResults,testData);
```

精确率/召回率 (PR) 曲线强调检测器在不同召回水平下的精确程度。理想情况下，所有召回水平的精确率均为 1。使用更多数据有助于提高平均精确率，但可能需要更多训练时间。绘制 PR 曲线。

```
figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
```



## 支持函数

```
function data = augmentData(data)
% Randomly flip images and bounding boxes horizontally.
tform = randomAffine2d('XReflection',true);
sz = size(data{1});
rout = affineOutputView(sz,tform);
data{1} = imwarp(data{1},tform,'OutputView',rout);
```

```
% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2}, sz);

% Warp boxes.
data{2} = bboxwarp(data{2},tform,rout);
end

function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));

% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2}, sz);

% Resize boxes.
data{2} = bboxresize(data{2},scale);
end
```

## 参考资料

- [1] Ren, S., K. He, R. Gershick, and J. Sun."Faster R-CNN:Towards Real-Time Object Detection with Region Proposal Networks."IEEE Transactions of Pattern Analysis and Machine Intelligence.Vol. 39, Issue 6, June 2017, pp. 1137-1149.
- [2] Girshick, R., J. Donahue, T. Darrell, and J. Malik."Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation."Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition.Columbus, OH, June 2014, pp. 580-587.
- [3] Girshick, R."Fast R-CNN."Proceedings of the 2015 IEEE International Conference on Computer Vision.Santiago, Chile, Dec. 2015, pp. 1440-1448.
- [4] Zitnick, C. L., and P. Dollar."Edge Boxes:Locating Object Proposals from Edges."European Conference on Computer Vision.Zurich, Switzerland, Sept. 2014, pp. 391-405.
- [5] Uijlings, J. R. R., K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders."Selective Search for Object Recognition."International Journal of Computer Vision.Vol. 104, Number 2, Sept. 2013, pp. 154-171.

## 另请参阅

`rcnnObjectDetector | trainingOptions | trainNetwork | trainRCNNObjectDetector | fastRCNNObjectDetector | fasterRCNNObjectDetector | trainFastRCNNObjectDetector | trainFasterRCNNObjectDetector | detect | insertObjectAnnotation | evaluateDetectionMissRate | evaluateDetectionPrecision`

## 详细信息

- "Semantic Segmentation" (Computer Vision Toolbox)
- "Object Detection" (Computer Vision Toolbox)

# 图像处理示例

---

## 使用预训练的神经网络去除彩色图像中的噪声

此示例说明如何从 RGB 图像中去除高斯噪声。将图像分为单独的颜色通道，然后使用预训练的去噪神经网络 DnCNN 对每个通道去噪。

将一个彩色图像读入工作区中，并将数据转换为 **double**。显示原始彩色图像。

```
pristineRGB = imread('lighthouse.png');
pristineRGB = im2double(pristineRGB);
imshow(pristineRGB)
title('Pristine Image')
```

**Pristine Image**

将方差为 0.01 的零均值高斯白噪声添加到图像中。`imnoise` 分别向每个颜色通道添加噪声。显示含噪彩色图像。

```
noisyRGB = imnoise(pristineRGB,'gaussian',0,0.01);
imshow(noisyRGB)
title('Noisy Image')
```



将含噪 RGB 图像分为单独的颜色通道。

```
[noisyR,noisyG,noisyB] = imsplit(noisyRGB);
```

加载预训练的 DnCNN 网络。

```
net = denoisingNetwork('dncnn');
```

使用 DnCNN 网络去除每个颜色通道的噪声。

```
denoisedR = denoiseImage(noisyR,net);
denoisedG = denoiseImage(noisyG,net);
denoisedB = denoiseImage(noisyB,net);
```

合并去噪的颜色通道以形成去噪后的 RGB 图像。显示去噪后的彩色图像。

```
denoisedRGB = cat(3,denoisedR,denoisedG,denoisedB);
imshow(denoisedRGB)
title('Denoised Image')
```



计算含噪图像和去噪图像的峰值信噪比 (PSNR)。PSNR 越大，噪声相对信号越小，说明图像质量越高。

```
noisyPSNR = psnr(noisyRGB,pristineRGB);
fprintf('\n The PSNR value of the noisy image is %0.4f!',noisyPSNR);
```

The PSNR value of the noisy image is 20.6395.

```
denoisedPSNR = psnr(denoisedRGB,pristineRGB);
fprintf('\n The PSNR value of the denoised image is %0.4f.',denoisedPSNR);
```

The PSNR value of the denoised image is 29.6857.

计算含噪图像和去噪图像的结构相似性 (SSIM) 指数。SSIM 指数接近 1 表示与参考图像相当一致，图像质量更高。

```
noisySSIM = ssim(noisyRGB,pristineRGB);
fprintf('\n The SSIM value of the noisy image is %0.4f.',noisySSIM);
```

The SSIM value of the noisy image is 0.7393.

```
denoisedSSIM = ssim(denoisedRGB,pristineRGB);
fprintf('\n The SSIM value of the denoised image is %0.4f.',denoisedSSIM);
```

The SSIM value of the denoised image is 0.9507.

实际上，图像颜色通道经常具有相关噪声。为了去除相关的图像噪声，首先将 RGB 图像转换为具有亮度通道的颜色空间，例如 L\*a\*b\* 颜色空间。仅去除亮度通道上的噪声，然后将去噪图像转换回 RGB 颜色空间。

## 另请参阅

[denoisingNetwork](#) | [denoiseImage](#) | [rgb2lab](#) | [lab2rgb](#) | [psnr](#) | [ssim](#) | [imnoise](#)

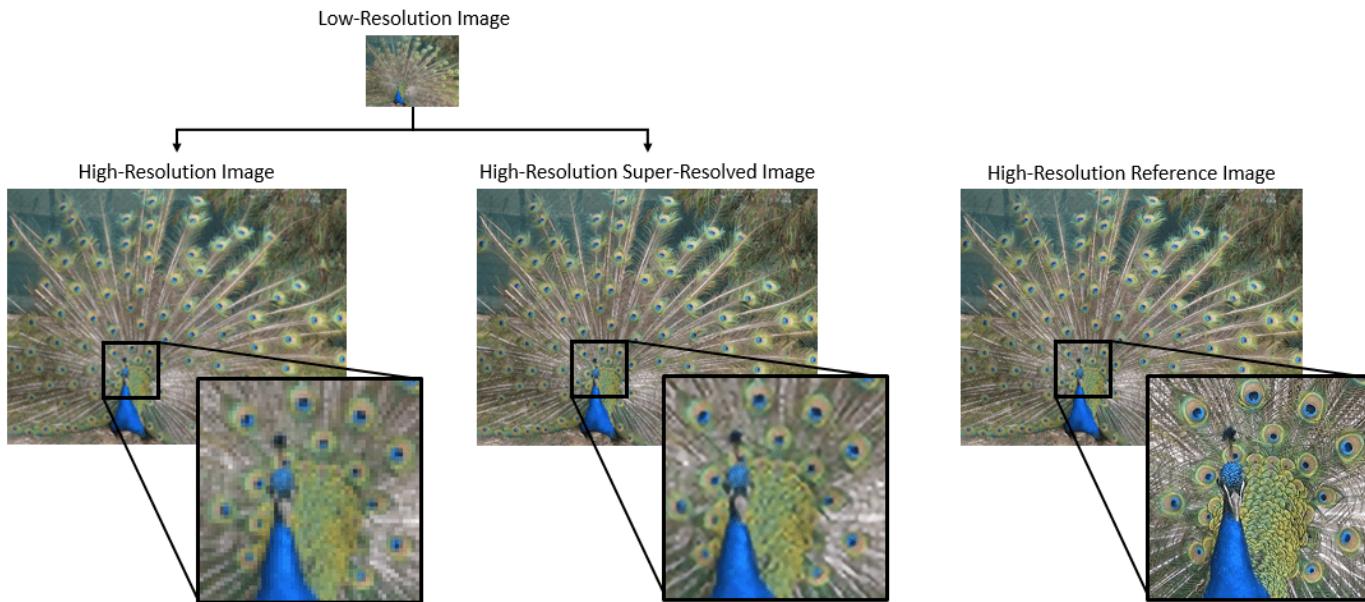
## 详细信息

- “Train and Apply Denoising Neural Networks” (Image Processing Toolbox)

## 使用深度学习执行单图像超分辨率

此示例说明如何使用超深超分辨率 (VDSR) 神经网络从低分辨率图像估计高分辨率图像。

超分辨率是从低分辨率图像创建高分辨率图像的过程。此示例以单图像超分辨率 (SISR) 为例，其目标是从一个低分辨率图像中恢复一个高分辨率图像。SISR 具有挑战性，因为高频图像成分通常无法从低分辨率图像中恢复。没有高频信息，所得高分辨率图像的质量是有限的。此外，SISR 是一个不适定问题，因为一个低分辨率图像可以产生若干种可能的高分辨率图像。



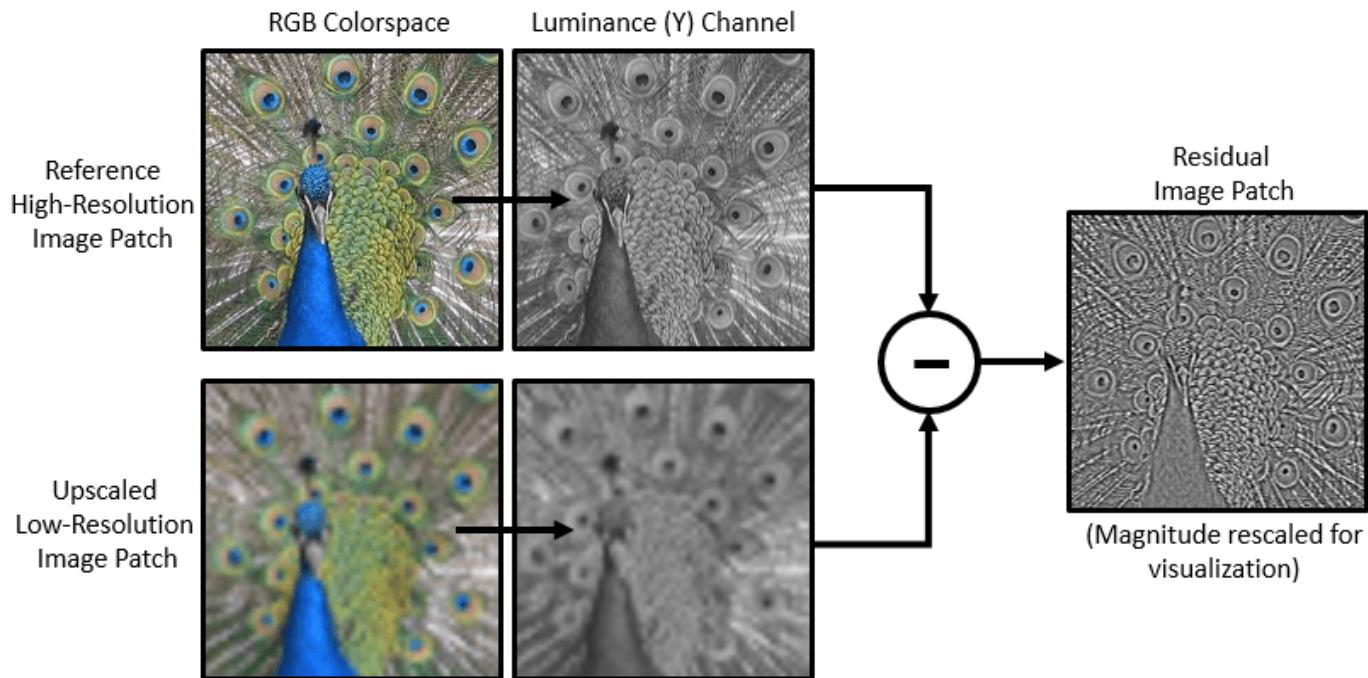
目前已提出包括深度学习算法在内的几种方法来执行 SISR。此示例探索了一种适用于 SISR 的深度学习算法，称为超深超分辨率 (VDSR) [1 (第 9-0 页)]。

### VDSR 网络

VDSR 是一种用于执行单图像超分辨率 [1 (第 9-0 页)] 的卷积神经网络架构。VDSR 网络学习低分辨率图像和高分辨率图像之间的映射。这种映射是有可能获得的，因为低分辨率和高分辨率图像具有相似的图像成分，主要区别在于高频细节。

VDSR 采用一种残差学习策略，这意味着网络会学习估计残差图像。在超分辨率中，残差图像是高分辨率参考图像和低分辨率图像之间的差异，低分辨率图像已使用双三次插值扩增以匹配参考图像的大小。残差图像包含关于图像高频细节的信息。

VDSR 网络基于彩色图像的亮度检测残差图像。图像的亮度通道 Y 使用红色、绿色和蓝色像素值的线性组合来表示每个像素的亮度。另一方面，图像的两个色度通道 Cb 和 Cr 使用红色、绿色和蓝色像素值的其他线性组合来表示色差信息。VDSR 仅使用亮度通道进行训练，因为人类对亮度变化的感知比对颜色变化更敏感。



如果  $Y_{\text{highres}}$  是高分辨率图像的亮度， $Y_{\text{lowres}}$  是使用双三次插值扩增的低分辨率图像的亮度，则 VDSR 网络的输入是  $Y_{\text{lowres}}$ ，网络学习基于训练数据预测  $Y_{\text{residual}} = Y_{\text{highres}} - Y_{\text{lowres}}$ 。

在 VDSR 网络学习估计残差图像后，您可以将估计的残差图像添加到上采样的低分辨率图像中，然后将图像转换回 RGB 颜色空间，从而重构高分辨率图像。

参考图像大小与低分辨率图像大小的关系可通过缩放因子来表示。随着缩放因子的增大，SISR 变得更加不适用，因为低分辨率图像会丢失有关高频图像成分的更多信息。VDSR 通过使用大感受野来解决此问题。此示例通过缩放增强来训练多缩放因子 VDSR 网络。缩放增强可以改进在较大缩放因子下的结果，因为网络可以利用较小缩放因子下的图像上下文。此外，VDSR 网络可以泛化至接受具有非整数缩放因子的图像。

### 下载训练和测试数据

下载包含 20,000 个静态自然图像的 IAPR TC-12 Benchmark [2 (第 9-0 页)]。该数据集包括人、动物、城市等的照片。数据文件的大小约为 1.8 GB。如果您不想下载训练数据集，则可以通过在命令行中键入 `load('trainedVDSR-Epoch-100-ScaleFactors-234.mat')` 来加载预训练的 VDSR 网络。然后，直接转至本示例中的使用 VDSR 网络执行单图像超分辨率 (第 9-0 页) 部分。

使用辅助函数 `downloadIAPRTC12Data` 下载数据。此函数作为支持文件包含在本示例中。

```
imagesDir = tempdir;
url = 'http://www-i6.informatik.rwth-aachen.de/imageclef/resources/iaprtc12.tgz';
downloadIAPRTC12Data(url,imagesDir);
```

此示例将使用 IAPR TC-12 Benchmark 数据的一个小型子集来训练网络。加载 imageCLEF 训练数据。所有图像均为 32 位 JPEG 彩色图像。

```
trainImagesDir = fullfile(imagesDir,'iaprtc12','images','02');
exts = {'jpg','bmp','png'};
pristineImages = imageDatastore(trainImagesDir,'FileExtensions',exts);
```

列出训练图像的数量。

```
numel(pristineImages.Files)
ans = 616
```

### 准备训练数据

要创建训练数据集，请生成由上采样图像和对应残差图像组成的图像对。

上采样图像以 MAT 文件形式存储在磁盘的 `upsampledDirName` 目录中。计算出的表示网络响应的残差图像以 MAT 文件形式存储在磁盘的 `residualDirName` 目录中。这些 MAT 文件以 `double` 数据类型存储，以便在训练网络时实现更高的精度。

```
upsampledDirName = [trainImagesDir filesep 'upsampledImages'];
residualDirName = [trainImagesDir filesep 'residualImages'];
```

使用辅助函数 `createVDSRTrainingSet` 预处理训练数据。此函数作为支持文件包含在本示例中。

该辅助函数对 `trainImages` 中的每个原始图像执行以下操作：

- 将图像转换为 YCbCr 颜色空间
- 以不同缩放因子缩小亮度 (Y) 通道，以创建低分辨率样本图像，然后使用双三次插值将图像大小调整为原始大小
- 计算原始图像和调整大小后的图像之间的差异。
- 将调整大小后的图像和残差图像保存到磁盘。

```
scaleFactors = [2 3 4];
createVDSRTrainingSet(pristineImages,scaleFactors,upsampledDirName,residualDirName);
```

### 定义训练集的预处理流程

在本示例中，网络输入是使用双三次插值进行上采样后的低分辨率图像。所需的网络响应是残差图像。从输入图像文件集合中创建名为 `upsampledImages` 的图像数据存储。从计算的残差图像文件集合中创建名为 `residualImages` 的图像数据存储。两个数据存储都需要使用辅助函数 `matRead` 从图像文件中读取图像数据。此函数作为支持文件包含在本示例中。

```
upsampledImages = imageDatastore(upsampledDirName,'FileExtensions','.mat','ReadFcn',@matRead);
residualImages = imageDatastore(residualDirName,'FileExtensions','.mat','ReadFcn',@matRead);
```

创建指定数据增强参数的 `imageDataAugmenter`。在训练期间使用数据增强来更改训练数据，这可以有效地增加可用的训练数据量。此处，增强器指定 90 度的随机旋转和 x 方向上的随机翻转。

```
augmenter = imageDataAugmenter(
    'RandRotation',@(0:randi([0,1],1)*90, ...
    'RandXReflection',true);
```

创建一个 `randomPatchExtractionDatastore` (Image Processing Toolbox) 以根据上采样图像和残差图像数据存储执行随机补片提取。补片提取是从单个较大图像中提取大量小图像补片或图块的过程。这种类型的数据增强经常用于图像到图像回归问题，其中许多网络架构可以基于非常小的输入图像大小进行训练。这意味着可以从原始训练集中的每个全大小图像中提取大量补片，这极大地增加了训练集的大小。

```
patchSize = [41 41];
patchesPerImage = 64;
dsTrain = randomPatchExtractionDatastore(upsampledImages,residualImages,patchSize, ...
    "DataAugmentation",augmenter,"PatchesPerImage",patchesPerImage);
```

生成的数据存储 `dsTrain` 在一轮训练的每次迭代中向网络提供小批量数据。预览从数据存储中读取的结果。

```
inputBatch = preview(dsTrain);
disp(inputBatch)
```

InputImage	ResponseImage
{41×41 double}	{41×41 double}

## 设置 VDSR 层

此示例使用 Deep Learning Toolbox™ 中的 41 个单独层定义 VDSR 网络，这些层包括：

- `imageInputLayer` - 图像输入层
- `convolution2dLayer` - 卷积神经网络的二维卷积层
- `reluLayer` - 修正线性单元 (ReLU) 层
- `regressionLayer` - 神经网络的回归输出层

第一个层，即 `imageInputLayer`，对图像补片进行操作。补片大小基于网络感受野，它是一个空间图像区域，影响网络中最顶层的响应。理想情况下，网络感受野与图像大小相同，这样感受野可以看到图像中的所有高级特征。在这种情况下，对于具有 D 个卷积层的网络，感受野是  $(2D+1) \times (2D+1)$ 。

VDSR 有 20 个卷积层，因此感受野和图像补片大小为  $41 \times 41$ 。图像输入层接受具有一个通道的图像，因为仅使用亮度通道训练 VDSR。

```
networkDepth = 20;
firstLayer = imageInputLayer([41 41 1], 'Name', 'InputLayer', 'Normalization', 'none');
```

图像输入层后跟一个二维卷积层，其中包含 64 个大小为  $3 \times 3$  的滤波器。小批量大小决定滤波器的数量。对每个卷积层的输入填零，使特征图与每次卷积后的输入始终大小相同。He 的方法 [3 (第 9-0 页)] 将权重初始化为随机值，以在神经元学习中引入不对称性。每个卷积层后跟一个 ReLU 层，该层在网络中引入非线性。

```
convLayer = convolution2dLayer(3, 64, 'Padding', 1, ...
    'WeightsInitializer', 'he', 'BiasInitializer', 'zeros', 'Name', 'Conv1');
```

指定一个 ReLU 层。

```
relLayer = reluLayer('Name', 'ReLU1');
```

中间各层包含 18 个交替的卷积层和修正线性单元层。每个卷积层包含 64 个大小为  $3 \times 3 \times 64$  的滤波器，每个滤波器对 64 个通道中的  $3 \times 3$  空间区域进行操作。如前所述，每个卷积层都后跟一个 ReLU 层。

```
middleLayers = [convLayer relLayer];
for layerNumber = 2:networkDepth-1
    convLayer = convolution2dLayer(3, 64, 'Padding', [1 1], ...
        'WeightsInitializer', 'he', 'BiasInitializer', 'zeros', ...
```

```
'Name',[Conv' num2str(layerNumber)]);

relLayer = reluLayer('Name',[ReLU' num2str(layerNumber)]);
middleLayers = [middleLayers convLayer relLayer];
end
```

倒数第二个层是一个卷积层，该层具有一个大小为  $3 \times 3 \times 64$  的滤波器，用于重构图像。

```
convLayer = convolution2dLayer(3,1,'Padding',[1 1], ...
    'WeightsInitializer','he','BiasInitializer','zeros',...
    'NumChannels',64,['Name',[Conv' num2str(networkDepth)]]);
```

最后一层是一个回归层，而不是 ReLU 层。该回归层计算残差图像和网络预测之间的均方误差。

```
finalLayers = [convLayer regressionLayer('Name','FinalRegressionLayer')];
```

串联所有层以形成 VDSR 网络。

```
layers = [firstLayer middleLayers finalLayers];
```

您也可以使用 `vdsrLayers` 辅助函数来创建 VDSR 层。此函数作为支持文件包含在本示例中。

```
layers = vdsrLayers;
```

### 指定训练选项

使用具有动量的随机梯度下降 (SGDM) 优化来训练网络。使用 `trainingOptions` 函数指定 SGDM 的超参数设置。学习率最初为 **0.1**，每 10 轮降低为原来的十分之一。进行 100 轮训练。

训练深度网络是很费时间的。通过指定高学习率可加快训练速度。然而，这可能会导致网络的梯度爆炸或不受控制地增长，阻碍网络训练成功。要将梯度保持在有意义的范围内，请通过将 `'GradientThreshold'` 指定为 **0.01** 来启用梯度裁剪，并指定 `'GradientThresholdMethod'` 使用梯度的 L2-范数。

```
maxEpochs = 100;
epochIntervals = 1;
initLearningRate = 0.1;
learningRateFactor = 0.1;
l2reg = 0.0001;
miniBatchSize = 64;
options = trainingOptions('sgdm',...
    'Momentum',0.9, ...
    'InitialLearnRate',initLearningRate, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',10, ...
    'LearnRateDropFactor',learningRateFactor, ...
    'L2Regularization',l2reg, ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThresholdMethod','l2norm', ...
    'GradientThreshold',0.01, ...
    'Plots','training-progress', ...
    'Verbose',false);
```

### 训练网络

默认情况下，该示例加载 VDSR 网络的一个预训练版本，该版本已对超分辨率图像进行缩放因子为 2、3 和 4 的训练。借助预训练网络，您无需等待训练完成，即可执行测试图像的超分辨率处理。

要训练 VDSR 网络，请将以下代码中的 **doTraining** 变量设置为 **true**。使用 **trainNetwork** 函数训练网络。

在 GPU 上（如果有）进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。在 NVIDIA Titan X 上训练大约需要 6 个小时。

```
doTraining = false;
if doTraining
    net = trainNetwork(dsTrain,layers,options);
    modelDateTime = string(datetime('now','Format','yyyy-MM-dd-HH-mm-ss'));
    save(strcat("trainedVDSR-",modelDateTime,"-Epoch-",num2str(maxEpochs),"ScaleFactors-234.mat"),'net');
else
    load('trainedVDSR-Epoch-100-ScaleFactors-234.mat');
end
```

## 使用 VDSR 网络执行单图像超分辨率

要使用 VDSR 网络执行单图像超分辨率 (SISR)，请按照此示例的后续步骤进行操作。该示例的后续部分显示如何：

- 基于高分辨率参考图像创建一个示例低分辨率图像。
- 使用双三次插值对该低分辨率图像执行 SISR，双三次插值是一种不依赖于深度学习的传统图像处理解决方案。
- 使用 VDSR 神经网络对该低分辨率图像执行 SISR。
- 以可视化方式比较使用双三次插值和 VDSR 重构的高分辨率图像。
- 量化超分辨率图像与高分辨率参考图像的相似性，以此评估前者的质量。

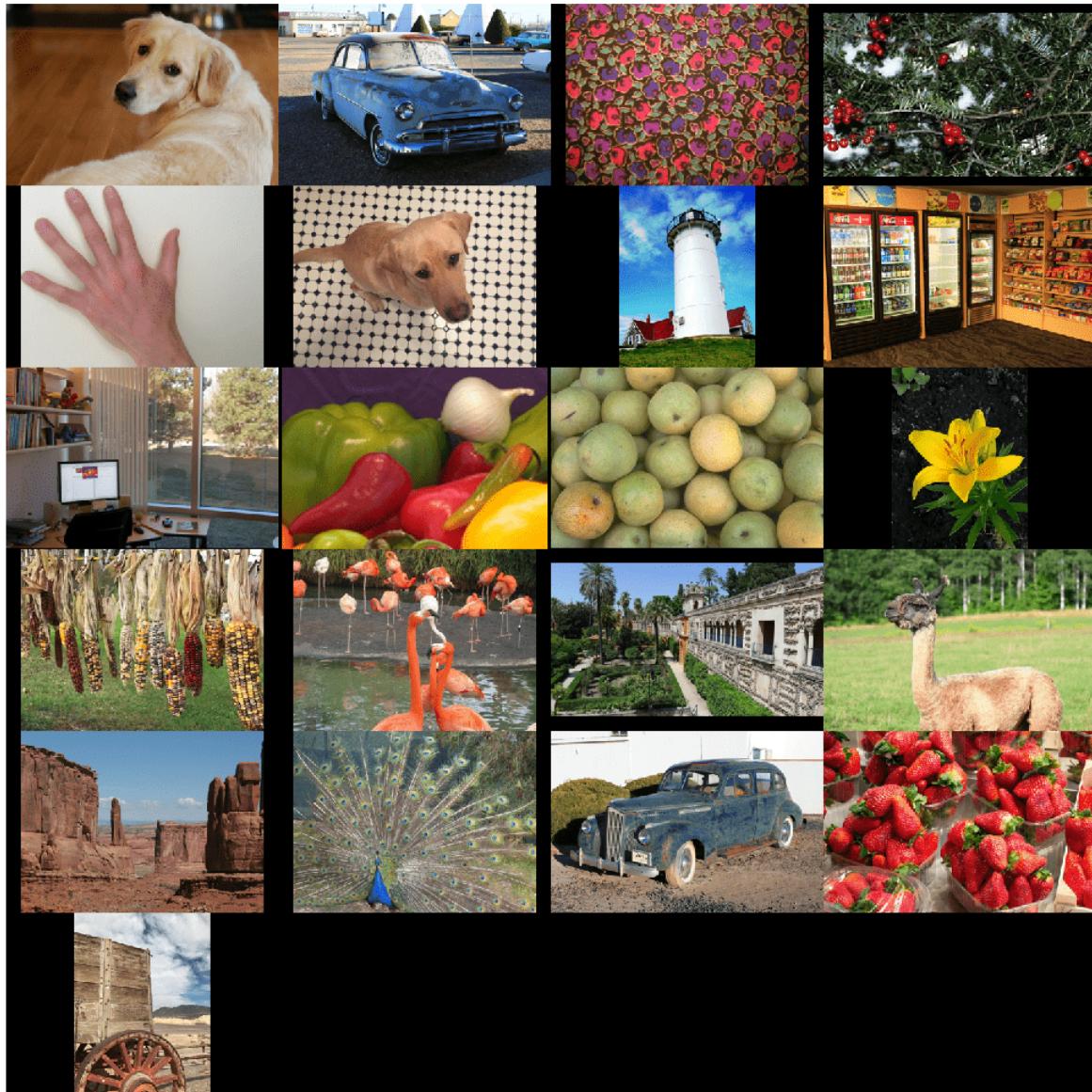
### 创建示例低分辨率图像

创建一个低分辨率图像，用于比较使用深度学习的超分辨率结果和使用双三次插值等传统图像处理方法的结果。测试数据集 **testImages** 包含 Image Processing Toolbox™ 中提供的 21 个未失真图像。将这些图像加载到 **imageDatastore** 中。

```
exts = {'jpg','png'};
fileNames = {'sherlock.jpg','car2.jpg','fabric.png','greens.jpg','hands1.jpg','kobi.png',...
    'lighthouse.png','micromarket.jpg','office_4.jpg','onion.png','pears.png','yellowlily.jpg',...
    'indiancorn.jpg','flamingos.jpg','sevilla.jpg','llama.jpg','parkavenue.jpg',...
    'peacock.jpg','car1.jpg','strawberries.jpg','wagon.jpg'};
filePath = [fullfile(matlabroot,'toolbox','images','imdata') filesep];
filePathNames = strcat(filePath,fileNames);
testImages = imageDatastore(filePathNames,'FileExtensions',exts);
```

将测试图像显示为蒙太奇。

```
montage(testImages)
```



选择其中一个图像作为超分辨率的参考图像。您也可以自选其他高分辨率图像作为参考图像。

```
indx = 1; % Index of image to read from the test image datastore  
Ireference = readimage(testImages,indx);  
Ireference = im2double(Ireference);  
imshow(Ireference)  
title('High-Resolution Reference Image')
```



使用 `imresize` (Image Processing Toolbox) 和缩放因子 0.25 创建高分辨率参考图像的一个低分辨率版本。图像的高频分量在缩减分辨率过程中丢失。

```
scaleFactor = 0.25;
Ilowres = imresize(Ireference,scaleFactor,'bicubic');
imshow(Ilowres)
title('Low-Resolution Image')
```



### 使用双三次插值提高图像分辨率

要提高图像分辨率，一种不借助于深度学习的标准方法是双三次插值。使用双三次插值扩增低分辨率图像，使所得高分辨率图像与参考图像大小相同。

```
[nrows,ncols,np] = size(Ireference);
Ibicubic = imresize(Ilowres,[nrows ncols],'bicubic');
imshow(Ibicubic)
title('High-Resolution Image Obtained Using Bicubic Interpolation')
```



### 使用预训练的 VDSR 网络提高图像分辨率

如前文所述，VDSR 只使用图像的亮度通道进行训练，因为人类对亮度变化的感知比对颜色变化更敏感。

使用 `rgb2ycbcr` (Image Processing Toolbox) 函数将低分辨率图像从 RGB 颜色空间转换为亮度 (`Iy`) 和色度 (`Icb` 和 `Icr`) 通道。

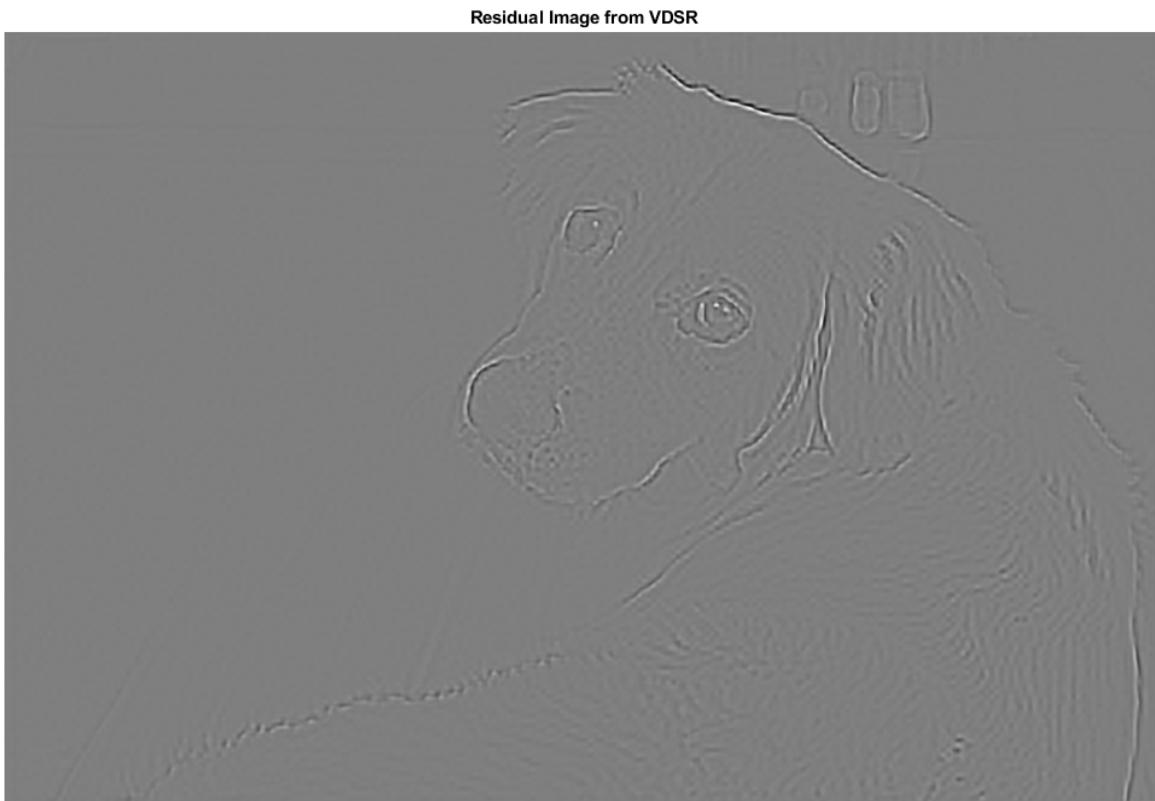
```
Iycbcr = rgb2ycbcr(Ilowres);
Iy = Iycbcr(:,:,1);
Icb = Iycbcr(:,:,2);
Icr = Iycbcr(:,:,3);
```

使用双三次插值扩增亮度通道和两个色度通道。上采样的色度通道 `Icb_bicubic` 和 `Icr_bicubic` 不需要进一步处理。

```
Iy_bicubic = imresize(Iy,[nrows ncols],'bicubic');
Icb_bicubic = imresize(Icb,[nrows ncols],'bicubic');
Icr_bicubic = imresize(Icr,[nrows ncols],'bicubic');
```

对扩增的亮度分量 `Iy_bicubic` 应用经过训练的 VDSR 网络。观察来自最终层（回归层）的 **activations**。网络的输出是所需的残差图像。

```
Iresidual = activations(net,Iy_bicubic,41);
Iresidual = double(Iresidual);
imshow(Iresidual,[])
title('Residual Image from VDSR')
```



将残差图像与扩增的亮度分量相加，得到高分辨率 VDSR 亮度分量。

```
Isr = Iy_bicubic + Iresidual;
```

将高分辨率 VDSR 亮度分量与扩增的颜色分量串联起来。使用 `ycbcr2rgb` (Image Processing Toolbox) 函数将图像转换为 RGB 颜色空间。结果为使用 VDSR 得到的最终高分辨率彩色图像。

```
Ivdsr = ycbcr2rgb(cat(3,Isr,Icb_bicubic,Icr_bicubic));
imshow(Ivdsr)
title('High-Resolution Image Obtained Using VDSR')
```



### 可视化和定量比较

为了在视觉上更好地理解高分辨率图像，请检查每个图像中的一个小区域。使用向量 `roi` 指定关注区域 (ROI)，格式为 [x y 宽度 高度]。其中的元素定义 ROI 左上角的 x 坐标和 y 坐标，以及它的宽度和高度。

```
roi = [320 30 480 400];
```

将高分辨率图像裁剪到该 ROI，并将结果显示为蒙太奇。与使用双三次插值创建的高分辨率图像相比，VDSR 图像具有更清晰的细节和更锐利的边缘。

```
montage({imcrop(Ibicubic,roi),imcrop(Ivdsr,roi)})  
title('High-Resolution Results Using Bicubic Interpolation (Left) vs. VDSR (Right)');
```



使用图像质量指标来定量比较使用双三次插值的高分辨率图像和 VDSR 图像。参考图像是准备低分辨率图像示例时所用的原始高分辨率图像 **Ireference**。

对照参考图像测量每个图像的峰值信噪比 (PSNR)。PSNR 值越大，通常表示图像质量越好。有关该指标的详细信息，请参阅 **psnr** (Image Processing Toolbox)。

```
bicubicPSNR = psnr(Ibicubic,Ireference)
```

```
bicubicPSNR = 38.4747
```

```
vdsrPSNR = psnr(Ivdsr,Ireference)
```

```
vdsrPSNR = 39.2346
```

测量每个图像的结构相似性指数 (SSIM)。SSIM 对照参考图像评估图像三个特性的视觉效果：亮度、对比度和结构。SSIM 值越接近 1，测试图像与参考图像越一致。有关该指标的详细信息，请参阅 **ssim** (Image Processing Toolbox)。

```
bicubicSSIM = ssim(Ibicubic,Ireference)
```

```
bicubicSSIM = 0.9861
```

```
vdsrSSIM = ssim(Ivdsr,Ireference)
```

```
vdsrSSIM = 0.9874
```

使用自然图像质量评价方法 (NIQE) 测量图像感知质量。NIQE 分数越小，表示感知质量越好。有关该指标的详细信息，请参阅 **niqe** (Image Processing Toolbox)。

```
bicubicNIQE = niqe(Ibicubic)
```

```
bicubicNIQE = 5.1721
```

```
vdsrNIQE = niqe(Ivdsr)
```

```
vdsrNIQE = 4.7611
```

分别计算缩放因子为 2、3 和 4 时整个测试图像集的平均 PSNR 和 SSIM。为简单起见，您可以使用辅助函数 `superResolutionMetrics` 来计算平均指标。此函数作为支持文件包含在本示例中。

```
scaleFactors = [2 3 4];  
superResolutionMetrics(net,testImages,scaleFactors);
```

Results for Scale factor 2

```
Average PSNR for Bicubic = 31.809683  
Average PSNR for VDSR = 31.921784  
Average SSIM for Bicubic = 0.938194  
Average SSIM for VDSR = 0.949404
```

Results for Scale factor 3

```
Average PSNR for Bicubic = 28.170441  
Average PSNR for VDSR = 28.563952  
Average SSIM for Bicubic = 0.884381  
Average SSIM for VDSR = 0.895830
```

Results for Scale factor 4

```
Average PSNR for Bicubic = 27.010839  
Average PSNR for VDSR = 27.837260  
Average SSIM for Bicubic = 0.861604  
Average SSIM for VDSR = 0.877132
```

对于每个缩放因子，VDSR 都具有比双三次插值更好的指标分数。

### 参考资料

- [1] Kim, J., J. K. Lee, and K. M. Lee."Accurate Image Super-Resolution Using Very Deep Convolutional Networks."Proceedings of the IEEE® Conference on Computer Vision and Pattern Recognition.2016, pp. 1646-1654.
- [2] Grubinger, M., P. Clough, H. Müller, and T. Deselaers."The IAPR TC-12 Benchmark:A New Evaluation Resource for Visual Information Systems."Proceedings of the Ontolimage 2006 Language Resources For Content-Based Image Retrieval.Genoa, Italy.Vol. 5, May 2006, p. 10.
- [3] He, K., X. Zhang, S. Ren, and J. Sun."Delving Deep into Rectifiers:Surpassing Human-Level Performance on ImageNet Classification."Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026-1034.

### 另请参阅

`trainingOptions` | `trainNetwork` | `transform` | `combine` | `activations` |  `imageDataAugmenter` | `imageDatastore`

### 详细信息

- “Datastores for Deep Learning”
- “预处理图像以进行深度学习”（第 18-2 页）
- “深度学习层列表”（第 1-18 页）

# 使用深度学习进行 JPEG 图像去块

此示例说明如何训练去噪卷积神经网络 (DnCNN)，然后使用该网络来减少图像中的 JPEG 压缩伪影。

图像压缩用于减少图像的内存占用。JPEG 图像格式采用一种流行且强大的压缩方法，它使用质量因子来指定压缩量。降低质量值会导致更高的压缩比和更小的内存占用，但会牺牲图像的视觉质量。

JPEG 压缩是有损的，这意味着压缩过程会导致图像丢失信息。对于 JPEG 图像，这种信息丢失表现为图像中的块伪影。如图所示，更多压缩会导致更多信息丢失和更强的伪影。具有高频成分的纹理区域（例如草和云）看起来模糊不清。锐边（例如屋顶和灯塔顶部护栏）会呈现振铃效应。



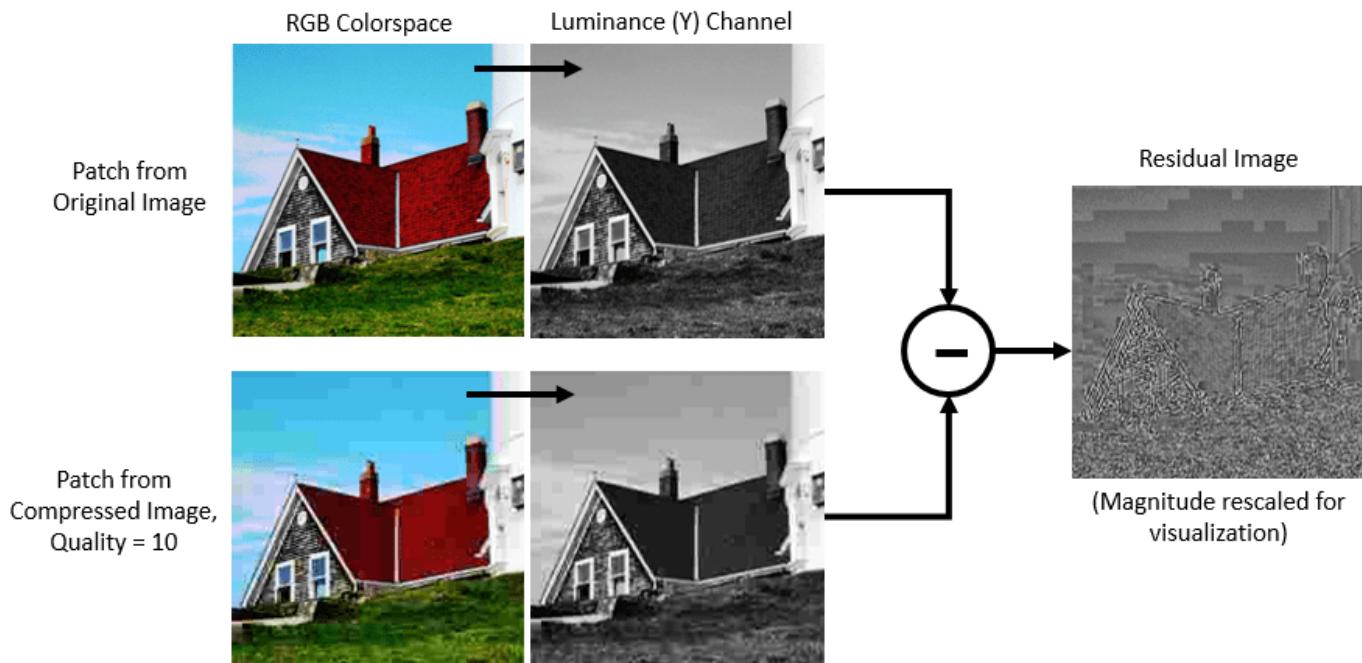
JPEG 去块是减少 JPEG 图像中压缩伪影影响的过程。有几种方法可以执行 JPEG 去块，其中一些较为高效的方法使用深度学习。此示例实现一种基于深度学习的方法，它尝试将 JPEG 压缩伪影的影响降到最低。

## DnCNN 网络

此示例使用一种内置深度前馈卷积神经网络，称为 **DnCNN**。该网络主要是为了去除图像中的噪声而设计的。不过，也可以训练 DnCNN 架构以去除 JPEG 压缩伪影或提高图像分辨率。

参考论文 [1 (第 9-0 页)] 采用残差学习策略，这意味着 DnCNN 网络学习估计残差图像。残差图像是原始图像和图像的失真副本之间的差异。残差图像包含关于图像失真的信息。在本示例中，失真显示为 JPEG 块伪影。

DnCNN 网络被训练用于通过彩色图像的亮度来检测残差图像。图像的亮度通道 Y 通过红色、绿色和蓝色像素值的线性组合来表示每个像素的亮度。另一方面，图像的两个色度通道 Cb 和 Cr 使用红色、绿色和蓝色像素值的其他线性组合来表示色差信息。DnCNN 仅使用亮度通道进行训练，因为人类对亮度变化的感知比对颜色变化更敏感。



如果  $Y_{\text{Original}}$  是原始图像的亮度，而  $Y_{\text{Compressed}}$  是包含 JPEG 压缩伪影的图像的亮度，则 DnCNN 网络的输入是  $Y_{\text{Compressed}}$  并且网络学习基于训练数据预测  $Y_{\text{Residual}} = Y_{\text{Compressed}} - Y_{\text{Original}}$ 。

一旦 DnCNN 网络学会如何估计残差图像，它就可以通过将残差图像添加到压缩的亮度通道、然后将图像转换回 RGB 颜色空间来重新构造压缩的 JPEG 图像的未失真版本。

### 下载训练数据

下载包含 20,000 个静态自然图像的 IAPR TC-12 Benchmark [2 (第 9-0 页)]。该数据集包括人、动物、城市等的照片。数据文件的大小约为 1.8 GB。如果您不想下载训练网络所需的训练数据集，可以通过在命令行中键入 `load('pretrainedJPEGDnCNN.mat')` 来加载预训练的 DnCNN 网络。然后，直接转至本示例中的使用 DnCNN 网络执行 JPEG 去块 (第 9-0 页) 部分。

使用辅助函数 `downloadIAPRTC12Data` 下载数据。此函数作为支持文件包含在本示例中。

```
imagesDir = tempdir;
url = "http://www-i6.informatik.rwth-aachen.de/imageclef/resources/iaprtc12.tgz";
downloadIAPRTC12Data(url,imagesDir);
```

此示例将使用 IAPR TC-12 Benchmark 数据的一个小型子集来训练网络。加载 imageCLEF 训练数据。所有图像均为 32 位 JPEG 彩色图像。

```
trainImagesDir = fullfile(imagesDir,'iaprtc12','images','00');
exts = {'jpg','bmp','png'};
imdsPristine = imageDatastore(trainImagesDir,'FileExtensions',exts);
```

列出训练图像的数量。

```
numel(imdsPristine.Files)
```

```
ans = 251
```

## 准备训练数据

要创建训练数据集，请读入原始图像，并以 JPEG 文件格式输出不同压缩级别的图像。

指定用于呈现图像压缩伪影的 JPEG 图像质量值。质量值必须在 [0, 100] 范围内。较小的质量值会导致更多压缩和更强的压缩伪影。使用更密集的小质量值来采样，以使训练数据具有广泛的压缩伪影。

```
JPEGQuality = [5:5:40 50 60 70 80];
```

压缩图像以 MAT 文件形式存储在磁盘的 `compressedImagesDir` 目录中。计算出的残差图像以 MAT 文件形式存储在磁盘的 `residualImagesDir` 目录中。这些 MAT 文件以 `double` 数据类型存储，以在训练网络时实现更高的精度。

```
compressedImagesDir = fullfile(imagesDir,'iaprtc12','JPEGDeblockingData','compressedImages');
residualImagesDir = fullfile(imagesDir,'iaprtc12','JPEGDeblockingData','residualImages');
```

使用辅助函数 `createJPEGDeblockingTrainingSet` 预处理训练数据。此函数作为支持文件包含在本示例中。

对于每个原始训练图像，辅助函数输出质量因子为 100 的图像副本用作参考图像，并且输出分别采用各个质量因子的图像副本用作网络输入。该函数以 `double` 数据类型计算参考图像和压缩图像的亮度 (Y) 通道，以更加精确地计算残差图像。压缩图像以 .MAT 文件形式存储在磁盘的 `compressedDirName` 目录中。计算出的残差图像以 .MAT 文件的形式存储在磁盘的 `residualDirName` 目录中。

```
[compressedDirName,residualDirName] = createJPEGDeblockingTrainingSet(imdsPristine,JPEGQuality);
```

## 为训练创建随机补片提取数据存储

使用随机补片提取数据存储将训练数据馈送给网络。此数据存储从包含网络输入和预期网络响应的两个图像数据存储中提取随机的对应补片。

在本示例中，网络输入是压缩图像。所需的网络响应是残差图像。从压缩图像文件集合中创建名为 `imdsCompressed` 的图像数据存储。从计算的残差图像文件集合中创建名为 `imdsResidual` 的图像数据存储。两个数据存储都需要使用辅助函数 `matRead` 从图像文件中读取图像数据。此函数作为支持文件包含在本示例中。

```
imdsCompressed = imageDatastore(compressedDirName,'FileExtensions','.mat','ReadFcn',@matRead);
imdsResidual = imageDatastore(residualDirName,'FileExtensions','.mat','ReadFcn',@matRead);
```

创建指定数据增强参数的 `imageDataAugmenter`。在训练期间使用数据增强来更改训练数据，这可以有效地增加可用的训练数据量。此处，增强器指定 90 度的随机旋转和 x 方向上的随机翻转。

```
augmenter = imageDataAugmenter( ...
    'RandRotation',@(randi([0,1],1)*90, ...
    'RandXReflection',true);
```

基于这两个图像数据存储创建 `randomPatchExtractionDatastore` (Image Processing Toolbox)。指定  $50 \times 50$  像素的补片大小。每个图像生成 128 个大小为  $50 \times 50$  像素的随机补片。指定小批量大小为 128。

```
patchSize = 50;
patchesPerImage = 128;
dsTrain = randomPatchExtractionDatastore(imdsCompressed,imdsResidual,patchSize, ...
    'PatchesPerImage',patchesPerImage, ...
    'DataAugmentation',augmenter);
dsTrain.MiniBatchSize = patchesPerImage;
```

随机补片提取数据存储 `dsTrain` 在一轮训练的迭代中向网络提供小批量数据。预览从数据存储中读取的结果。

```
inputBatch = preview(dsTrain);
disp(inputBatch)
```

InputImage	ResponseImage
{50×50 double}	{50×50 double}

## 设置 DnCNN 层

使用 `dnCNNLayers` (Image Processing Toolbox) 函数创建内置 DnCNN 网络的层。默认情况下，网络深度（卷积层个数）为 20。

```
layers = dnCNNLayers
```

```
layers =
1x59 Layer array with layers:
```

1 'InputLayer'	Image Input	50x50x1 images
2 'Conv1'	Convolution	64 3x3x1 convolutions with stride [1 1] and padding [1 1 1 1]
3 'ReLU1'	ReLU	ReLU
4 'Conv2'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
5 'BNorm2'	Batch Normalization	Batch normalization with 64 channels
6 'ReLU2'	ReLU	ReLU
7 'Conv3'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
8 'BNorm3'	Batch Normalization	Batch normalization with 64 channels
9 'ReLU3'	ReLU	ReLU
10 'Conv4'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
11 'BNorm4'	Batch Normalization	Batch normalization with 64 channels
12 'ReLU4'	ReLU	ReLU
13 'Conv5'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
14 'BNorm5'	Batch Normalization	Batch normalization with 64 channels
15 'ReLU5'	ReLU	ReLU
16 'Conv6'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
17 'BNorm6'	Batch Normalization	Batch normalization with 64 channels
18 'ReLU6'	ReLU	ReLU
19 'Conv7'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
20 'BNorm7'	Batch Normalization	Batch normalization with 64 channels
21 'ReLU7'	ReLU	ReLU
22 'Conv8'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
23 'BNorm8'	Batch Normalization	Batch normalization with 64 channels
24 'ReLU8'	ReLU	ReLU
25 'Conv9'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
26 'BNorm9'	Batch Normalization	Batch normalization with 64 channels
27 'ReLU9'	ReLU	ReLU
28 'Conv10'	Convolution	64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
29 'BNorm10'	Batch Normalization	Batch normalization with 64 channels
30 'ReLU10'	ReLU	ReLU

```

31 'Conv11'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
32 'BNorm11'        Batch Normalization  Batch normalization with 64 channels
33 'ReLU11'          ReLU             ReLU
34 'Conv12'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
35 'BNorm12'        Batch Normalization  Batch normalization with 64 channels
36 'ReLU12'          ReLU             ReLU
37 'Conv13'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
38 'BNorm13'        Batch Normalization  Batch normalization with 64 channels
39 'ReLU13'          ReLU             ReLU
40 'Conv14'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
41 'BNorm14'        Batch Normalization  Batch normalization with 64 channels
42 'ReLU14'          ReLU             ReLU
43 'Conv15'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
44 'BNorm15'        Batch Normalization  Batch normalization with 64 channels
45 'ReLU15'          ReLU             ReLU
46 'Conv16'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
47 'BNorm16'        Batch Normalization  Batch normalization with 64 channels
48 'ReLU16'          ReLU             ReLU
49 'Conv17'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
50 'BNorm17'        Batch Normalization  Batch normalization with 64 channels
51 'ReLU17'          ReLU             ReLU
52 'Conv18'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
53 'BNorm18'        Batch Normalization  Batch normalization with 64 channels
54 'ReLU18'          ReLU             ReLU
55 'Conv19'          Convolution      64 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
56 'BNorm19'        Batch Normalization  Batch normalization with 64 channels
57 'ReLU19'          ReLU             ReLU
58 'Conv20'          Convolution      1 3x3x64 convolutions with stride [1 1] and padding [1 1 1 1]
59 'FinalRegressionLayer' Regression Output  mean-squared-error

```

## 选择训练选项

使用具有动量的随机梯度下降 (SGDM) 优化来训练网络。使用 `trainingOptions` 函数指定 SGDM 的超参数设置。

训练深度网络是很费时间的。通过指定高学习率可加快训练速度。然而，这可能会导致网络的梯度爆炸或不受控制地增长，阻碍网络训练成功。要将梯度保持在有意义的范围内，请通过将 '`GradientThreshold`' 值设置为 **0.005** 来启用梯度裁剪，并指定 '`GradientThresholdMethod`' 使用梯度的绝对值。

```

maxEpochs = 30;
initLearningRate = 0.1;
l2reg = 0.0001;
batchSize = 64;

options = trainingOptions('sgdm', ...
    'Momentum',0.9, ...
    'InitialLearnRate',initLearningRate, ...
    'LearnRateSchedule','piecewise', ...
    'GradientThresholdMethod','absolute-value', ...
    'GradientThreshold',0.005, ...
    'L2Regularization',l2reg, ...
    'MiniBatchSize',batchSize, ...
    'MaxEpochs',maxEpochs, ...
    'Plots','training-progress', ...
    'Verbose',false);

```

## 训练网络

默认情况下，该示例加载一个预训练的 DnCNN 网络。该预训练网络可让您无需等待训练完成即可执行 JPEG 去块。

要训练网络，请将以下代码中的 `doTraining` 变量设置为 `true`。使用 `trainNetwork` 函数训练 DnCNN 网络。

在 GPU 上（如果有）进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。在 NVIDIA™ Titan X 上训练大约需要 40 个小时。

```
doTraining = false;
if doTraining
    modelDateTime = string(datetime('now','Format','yyyy-MM-dd-HH-mm-ss'));
    [net,info] = trainNetwork(dsTrain,layers,options);
    save(strcat("trainedJPEGDnCNN-",modelDateTime,"-Epoch-",num2str(maxEpochs),".mat"),'net');
else
    load('pretrainedJPEGDnCNN.mat');
end
```

现在，您可以使用 DnCNN 网络从图像中去除 JPEG 压缩伪影。

## 使用 DnCNN 网络执行 JPEG 去块

要使用 DnCNN 执行 JPEG 去块，请按照此示例的后续步骤进行操作。该示例的后续部分显示如何：

- 用三种不同质量级别的 JPEG 压缩伪影创建示例测试图像。
- 使用 DnCNN 网络去除压缩伪影。
- 以可视化方式比较去块之前和之后的图像。
- 对压缩和去块图像与未失真参考图像之间的相似性进行量化，以此评估压缩和去块图像的质量。

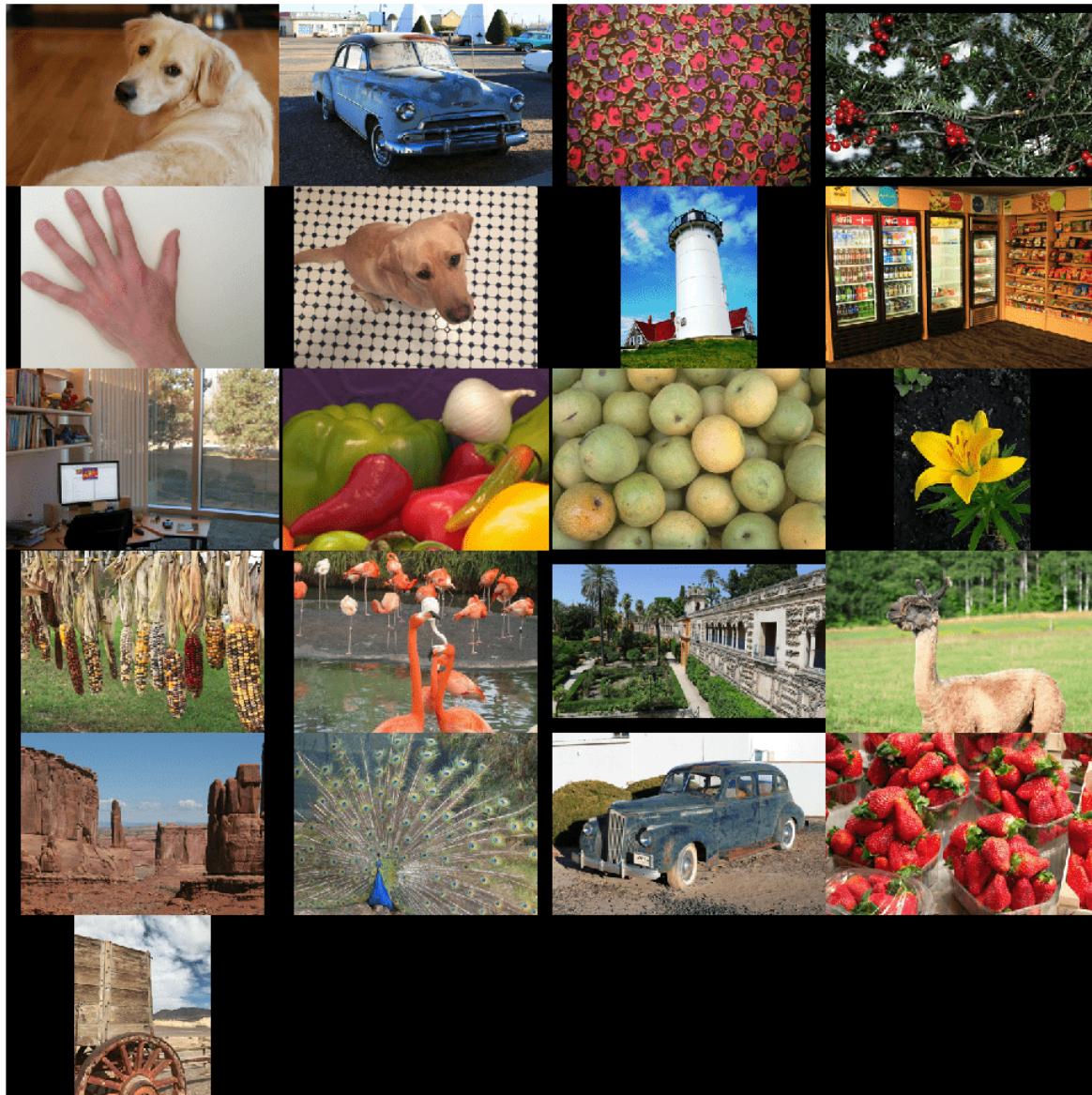
## 创建具有块伪影的示例图像

使用 DnCNN 网络创建示例图像以评估 JPEG 图像去块的结果。测试数据集 `testImages` 包含 Image Processing Toolbox™ 中提供的 21 个未失真图像。将这些图像加载到 `imageDatastore` 中。

```
exts = {'jpg','png'};
fileNames = {'sherlock.jpg','car2.jpg','fabric.png','greens.jpg','hands1.jpg','kobi.png',...
    'lighthouse.png','micromarket.jpg','office_4.jpg','onion.png','pears.png','yellowlily.jpg',...
    'indiancorn.jpg','flamingos.jpg','sevilla.jpg','llama.jpg','parkavenue.jpg',...
    'peacock.jpg','car1.jpg','strawberries.jpg','wagon.jpg'};
filePath = [fullfile(matlabroot,'toolbox','images','imdata') filesep];
filePathNames = strcat(filePath,fileNames);
testImages = imageDatastore(filePathNames,'FileExtensions',exts);
```

将测试图像显示为蒙太奇。

```
montage(testImages)
```



选择其中一个图像作为 JPEG 去块的参考图像。您也可以自选其他未压缩图像作为参考图像。

```
indx = 7; % Index of image to read from the test image datastore
Ireference = readimage(testImages,indx);
imshow(Ireference)
title('Uncompressed Reference Image')
```

Uncompressed Reference Image



使用 JPEG Quality 值 10、20 和 50 创建三个压缩测试图像。

```
imwrite(Ireference,fullfile(tempdir,'testQuality10.jpg'),'Quality',10);
imwrite(Ireference,fullfile(tempdir,'testQuality20.jpg'),'Quality',20);
imwrite(Ireference,fullfile(tempdir,'testQuality50.jpg'),'Quality',50);
```

## 预处理压缩图像

将图像的压缩版本读入工作区中。

```
I10 = imread(fullfile(tempdir,'testQuality10.jpg'));
I20 = imread(fullfile(tempdir,'testQuality20.jpg'));
I50 = imread(fullfile(tempdir,'testQuality50.jpg'));
```

将压缩图像显示为蒙太奇。

```
montage({I50,I20,I10}, 'Size',[1 3])
title('JPEG-Compressed Images with Quality Factor: 50, 20 and 10 (left to right)')
```



如前文所述，DnCNN 只使用图像的亮度通道进行训练，因为人类对亮度变化的感知比对颜色变化更敏感。使用 `rgb2ycbcr` (Image Processing Toolbox) 函数将 JPEG 压缩图像从 RGB 颜色空间转换为 YCbCr 颜色空间。

```
I10ycbcr = rgb2ycbcr(I10);
I20ycbcr = rgb2ycbcr(I20);
I50ycbcr = rgb2ycbcr(I50);
```

## 应用 DnCNN 网络

为了执行网络的前向传导，请使用 `denoiseImage` (Image Processing Toolbox) 函数。该函数使用完全相同的训练和测试过程对图像去噪。您可以将 JPEG 压缩伪影视为一种图像噪声。

```
I10y_predicted = denoiseImage(I10ycbcr(:,:,1),net);
I20y_predicted = denoiseImage(I20ycbcr(:,:,1),net);
I50y_predicted = denoiseImage(I50ycbcr(:,:,1),net);
```

色度通道不需要处理。将去块后的亮度通道与原始色度通道串联起来，以获得 YCbCr 颜色空间中的去块图像。

```
I10ycbcr_predicted = cat(3,I10y_predicted,I10ycbcr(:,:,2:3));
I20ycbcr_predicted = cat(3,I20y_predicted,I20ycbcr(:,:,2:3));
I50ycbcr_predicted = cat(3,I50y_predicted,I50ycbcr(:,:,2:3));
```

使用 `ycbcr2rgb` (Image Processing Toolbox) 函数将去块 YCbCr 图像转换为 RGB 颜色空间。

```
I10_predicted = ycbcr2rgb(I10ycbcr_predicted);
I20_predicted = ycbcr2rgb(I20ycbcr_predicted);
I50_predicted = ycbcr2rgb(I50ycbcr_predicted);
```

将去块图像显示为蒙太奇。

```
montage({I50_predicted,I20_predicted,I10_predicted},'Size',[1 3])
title('Deblocked Images with Quality Factor 50, 20 and 10 (Left to Right)')
```



为了在视觉上更好地理解这些改进，请检查每个图像中的一个小区域。使用向量 `roi` 指定关注区域 (ROI)，格式为 [x y 宽度 高度]。其中的元素定义 ROI 左上角的 x 坐标和 y 坐标，以及它的宽度和高度。

```
roi = [30 440 100 80];
```

将压缩图像裁切到该 ROI，并将结果显示为蒙太奇。

```
i10 = imcrop(I10,roi);
i20 = imcrop(I20,roi);
i50 = imcrop(I50,roi);
montage({i50 i20 i10},'Size',[1 3])
title('Patches from JPEG-Compressed Images with Quality Factor 50, 20 and 10 (Left to Right)')
```



将去块图像裁切到该 ROI，并将结果显示为蒙太奇。

```
i10predicted = imcrop(I10_predicted,roi);
i20predicted = imcrop(I20_predicted,roi);
i50predicted = imcrop(I50_predicted,roi);
montage({i50predicted,i20predicted,i10predicted},'Size',[1 3])
title('Patches from Deblocked Images with Quality Factor 50, 20 and 10 (Left to Right)')
```



## 定量比较

通过四个指标量化去块图像的质量。您可以使用 **displayJPEGResults** 辅助函数分别计算质量因子为 10、20 和 50 时压缩图像和去块图像的这些指标。此函数作为支持文件包含在本示例中。

- 结构相似性索引 (SSIM)。SSIM 对照参考图像评估图像三个特性的视觉效果：亮度、对比度和结构。SSIM 值越接近 1，测试图像与参考图像越一致。此处，参考图像是在 JPEG 压缩之前的未失真原始图像 **Ireference**。有关该指标的详细信息，请参阅 **ssim** (Image Processing Toolbox)。
- 峰值信噪比 (PSNR)。PSNR 值越大，信号较失真而言越强。有关该指标的详细信息，请参阅 **psnr** (Image Processing Toolbox)。
- 自然图像质量评价方法 (NIQE)。NIQE 使用基于自然场景训练的模型来测量图像感知质量。NIQE 分数越小，表示感知质量越好。有关该指标的详细信息，请参阅 **niqe** (Image Processing Toolbox)。
- 盲/无参考图像空间质量评价方法 (BRISQUE)。BRISQUE 使用基于具有图像失真的自然场景训练的模型来测量图像感知质量。BRISQUE 分数越小，表示感知质量越好。有关该指标的详细信息，请参阅 **brisque** (Image Processing Toolbox)。

```
displayJPEGResults(Ireference,I10,I20,I50,I10_predicted,I20_predicted,I50_predicted)
```

```
SSIM Comparison
=====
I10: 0.90624 I10_predicted: 0.91286
I20: 0.94904 I20_predicted: 0.95444
I50: 0.97238 I50_predicted: 0.97482

PSNR Comparison
=====
I10: 26.6046 I10_predicted: 27.0793
I20: 28.8015 I20_predicted: 29.3378
I50: 31.4512 I50_predicted: 31.8584

NIQE Comparison
=====
I10: 7.2194 I10_predicted: 3.9478
I20: 4.5158 I20_predicted: 3.0685
I50: 2.8874 I50_predicted: 2.4106
NOTE: Smaller NIQE score signifies better perceptual quality

BRISQUE Comparison
=====
I10: 52.372 I10_predicted: 38.9271
I20: 45.3772 I20_predicted: 30.8991
I50: 27.7093 I50_predicted: 24.3845
NOTE: Smaller BRISQUE score signifies better perceptual quality
```

### 参考资料

- [1] Zhang, K., W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser:Residual Learning of Deep CNN for Image Denoising."IEEE® Transactions on Image Processing.Feb 2017.
- [2] Grubinger, M., P. Clough, H. Müller, and T. Deselaers."The IAPR TC-12 Benchmark:A New Evaluation Resource for Visual Information Systems."Proceedings of the Ontolimage 2006 Language Resources For Content-Based Image Retrieval.Genoa, Italy.Vol. 5, May 2006, p. 10.

### 另请参阅

**rgb2ycbcr | ycbcr2rgb | dnCNNLayers | denoiseImage | trainingOptions | trainNetwork | randomPatchExtractionDatastore**

### 详细信息

- “预处理图像以进行深度学习” (第 18-2 页)
- “Datastores for Deep Learning”
- “深度学习层列表” (第 1-18 页)

## 使用深度学习进行图像处理算子逼近

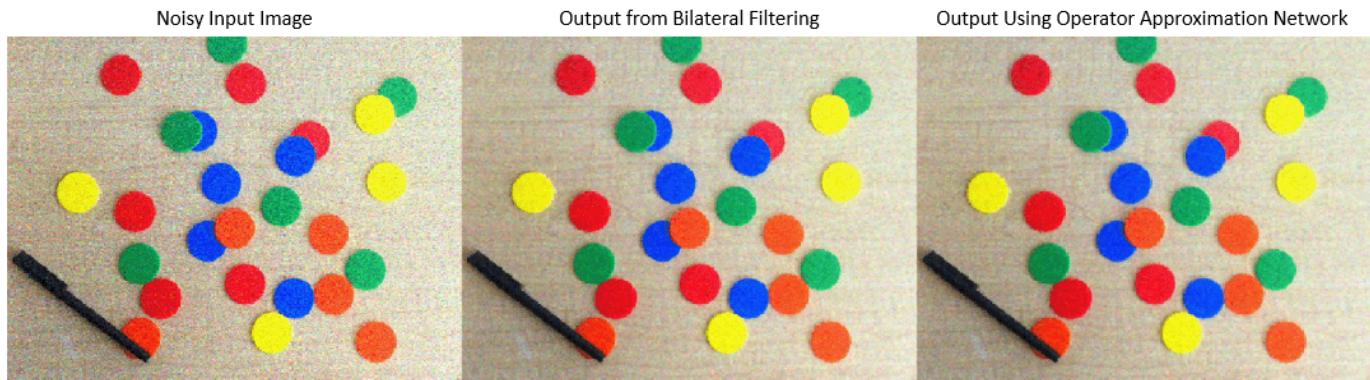
此示例说明如何使用多尺度上下文聚合网络 (CAN) 逼近图像滤波运算。

算子逼近寻找处理图像的替代方法，使得结果类似于常规图像处理运算或流程的输出。算子逼近的目标通常是减少处理图像所需的时间。

人们提出多种经典方法和深度学习方法来执行算子逼近。一些经典方法可提高单一算法的效率，但无法推广到其他运算。另一种常见技巧是将算子应用于图像的低分辨率副本以逼近各种运算，但高频成分的丢失会限制逼近的准确度。

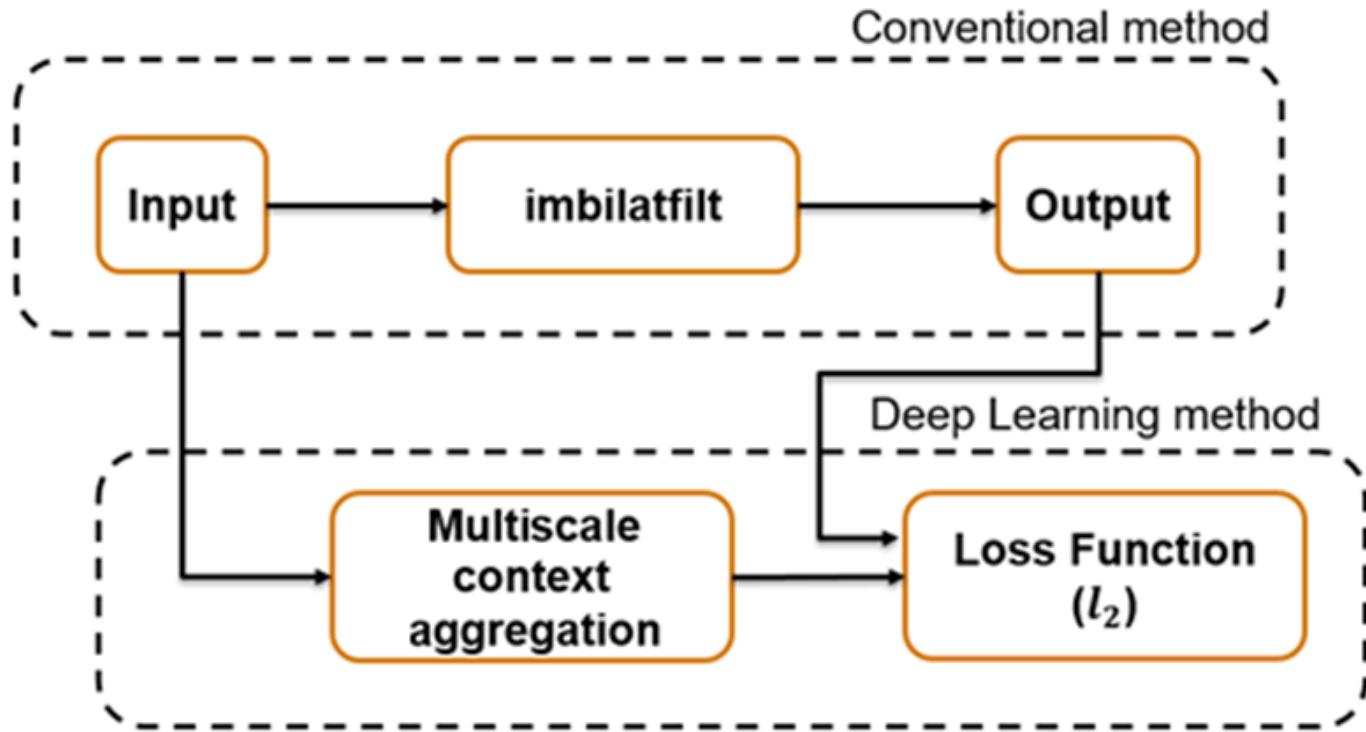
深度学习解决方案能够逼近更普遍且更复杂的运算。例如，Q. Chen [1 (第 9-0 页)] 提出的多尺度上下文聚合网络 (CAN) 可以逼近多尺度色调映射、照片样式转换、非局部去雾和铅笔绘图。多尺度 CAN 基于全分辨率图像进行训练，以提高处理高频细节的准确度。网络在经过训练后，可以绕过常规的处理运算而直接处理图像。

此示例展示如何训练多尺度 CAN 来逼近双边图像滤波运算，此运算在保持边缘锐度的同时降低图像噪声。该示例说明了完整的训练和推断工作流，包括创建训练数据存储、选择训练选项、训练网络以及使用网络处理测试图像的过程。



### 算子逼近网络

对多尺度 CAN 进行训练，以最小化图像处理运算的常规输出和使用多尺度上下文聚合处理输入图像后的网络响应之间的  $l_2$  损失。多尺度上下文聚合从整个图像中寻找关于每个像素的信息，而不是将搜索范围限制在像素周围的小邻域。



为了帮助网络学习全局图像属性，多尺度 CAN 架构有很大的感受野。第一层和最后一层具有相同的大小，因为算子不应更改图像的大小。连续的各中间层通过指数级增加的缩放因子而扩张（因此 CAN 具有“多尺度”的特点）。扩张使网络能够在各种空间频率下寻找空间分离的特征，而不会降低图像的分辨率。在每个卷积层后，网络使用自适应归一化来平衡批量归一化和恒等映射对逼近算子的影响。

### 下载训练和测试数据

下载包含 20,000 个静态自然图像的 IAPR TC-12 Benchmark [2 (第 9-0 页)]。该数据集包括人、动物、城市等的照片。数据文件的大小约为 1.8 GB。如果您不想下载训练网络所需的训练数据集，可以通过在命令行中键入 `load('trainedOperatorLearning-Epoch-181.mat');` 来加载预训练的 CAN。然后，直接转至本示例中的使用多尺度 CAN 执行双边滤波逼近 (第 9-0 页) 部分。

```
imagesDir = tempdir;
url_1 = 'http://www-i6.informatik.rwth-aachen.de/imageclef/resources/iaprtc12.tgz';
downloadIAPRTC12Data(url_1,imagesDir);
```

此示例使用 IAPRTC-12 Benchmark 数据的小型子集来训练网络。

```
trainImagesDir = fullfile(imagesDir,'iaprtc12','images','39');
exts = {'jpg','bmp','png'};
pristineImages = imageDatastore(trainImagesDir,'FileExtensions',exts);
```

列出训练图像的数量。

```
numel(pristineImages.Files)
ans = 916
```

### 准备训练数据

要创建训练数据集，请读入原始图像，并输出经过双边滤波的图像。滤波后的图像存储在由 `preprocessDataDir` 指定的磁盘目录中。

```
preprocessDataDir = [trainImagesDir filesep 'preprocessedDataset'];
```

使用辅助函数 `bilateralFilterDataset` 预处理训练数据。此函数作为支持文件包含在本示例中。

该辅助函数对 `inputImages` 中的每个原始图像执行以下操作：

- 计算双边滤波的平滑度。对滤波后的图像进行平滑处理可以降低图像噪声。
- 使用 `imbilatfilt` (Image Processing Toolbox) 执行双边滤波。
- 使用 `imwrite` 将滤波后的图像保存到磁盘。

```
bilateralFilterDataset(pristineImages,preprocessDataDir);
```

### 为训练定义随机补片提取数据存储

使用随机补片提取数据存储将训练数据馈送给网络。此数据存储从包含网络输入和预期网络响应的两个图像数据存储中提取随机的对应补片。

在此示例中，网络输入是 `pristineImages` 中的原始图像。所需的网络响应是经过双边滤波处理的图像。从双边滤波后的图像文件集合中创建名为 `bilatFilteredImages` 的图像数据存储。

```
bilatFilteredImages = imageDatastore(preprocessDataDir,'FileExtensions',exts);
```

基于这两个图像数据存储创建一个 `randomPatchExtractionDatastore` (Image Processing Toolbox)。指定  $256 \times 256$  像素的补片大小。指定 '`PatchesPerImage`' 以在训练期间从每对图像中提取一个随机定位的补片。指定小批量大小为一。

```
miniBatchSize = 1;
patchSize = [256 256];
dsTrain = randomPatchExtractionDatastore(pristineImages,bilatFilteredImages,patchSize, ...
    'PatchesPerImage',1);
dsTrain.MiniBatchSize = miniBatchSize;
```

`randomPatchExtractionDatastore` 在一轮训练的每次迭代中向网络提供小批量数据。对数据存储执行读取操作以探查数据。

```
inputBatch = read(dsTrain);
disp(inputBatch)
```

InputImage	ResponseImage
{ $256 \times 256 \times 3$ uint8}	{ $256 \times 256 \times 3$ uint8}

### 设置多尺度 CAN 层

此示例使用 Deep Learning Toolbox™ 中的层定义多尺度 CAN，这些层包括：

- `imageInputLayer` - 图像输入层
- `convolution2dLayer` - 卷积神经网络的二维卷积层
- `batchNormalizationLayer` - 批量归一化层
- `leakyReluLayer` - 泄漏修正线性单元层
- `regressionLayer` - 神经网络的回归输出层

添加两个自定义尺度层来实现一个自适应批量归一化层。这些层作为支持文件包含在本示例中。

- **adaptiveNormalizationMu** - 调整批量归一化分支强度的尺度层
- **adaptiveNormalizationLambda** - 调整恒等分支强度的尺度层

第一个层，即 **imageInputLayer**，对图像补片进行操作。补片大小基于网络感受野，它是一个空间图像区域，影响网络中最顶层的响应。理想情况下，网络感受野与图像大小相同，这样它就可以看到图像中的所有高级特征。对于双边滤波器，逼近图像补片大小固定为  $256 \times 256$ 。

```
networkDepth = 10;
numberOfFilters = 32;
firstLayer = imageInputLayer([256 256 3], 'Name', 'InputLayer', 'Normalization', 'none');
```

图像输入层后跟一个二维卷积层，其中包含 32 个大小为  $3 \times 3$  的滤波器。对每个卷积层的输入填零，使特征图与每次卷积后的输入始终大小相同。将权重初始化为单位矩阵。

```
Wgts = zeros(3,3,3,numberOfFilters);
for ii = 1:3
    Wgts(2,2,ii,ii) = 1;
end
convolutionLayer = convolution2dLayer(3,numberOfFilters, 'Padding', 1, ...
    'Weights', Wgts, 'Name', 'Conv1');
```

每个卷积层后跟一个批量归一化层，以及一个用于调整批量归一化分支强度的自适应归一化缩放层。稍后，此示例还将相应创建用于调整恒等分支强度的自适应归一化尺度层。现在，在 **adaptiveNormalizationMu** 层之后指定一个相加层。最后，为负输入指定标量乘数为 0.2 的泄漏 ReLU 层。

```
batchNorm = batchNormalizationLayer('Name', 'BN1');
adaptiveMu = adaptiveNormalizationMu(numberOfFilters, 'Mu1');
addLayer = additionLayer(2, 'Name', 'add1');
leakyrelLayer = leakyReluLayer(0.2, 'Name', 'Leaky1');
```

按照相同的模式指定网络的中间层。连续卷积层具有一个随网络加深而指数级放大的扩张因子。

```
middleLayers = [convolutionLayer batchNorm adaptiveMu addLayer leakyrelLayer];

Wgts = zeros(3,3,numberOfFilters,numberOfFilters);
for ii = 1:numberOfFilters
    Wgts(2,2,ii,ii) = 1;
end

for layerNumber = 2:networkDepth-2
    dilationFactor = 2^(layerNumber-1);
    padding = dilationFactor;
    conv2dLayer = convolution2dLayer(3,numberOfFilters, ...
        'Padding', padding, 'DilationFactor', dilationFactor, ...
        'Weights', Wgts, 'Name', ['Conv' num2str(layerNumber)]);
    batchNorm = batchNormalizationLayer('Name', ['BN' num2str(layerNumber)]);
    adaptiveMu = adaptiveNormalizationMu(numberOfFilters, ['Mu' num2str(layerNumber)]);
    addLayer = additionLayer(2, 'Name', ['add' num2str(layerNumber)]);
    leakyrelLayer = leakyReluLayer(0.2, 'Name', ['Leaky' num2str(layerNumber)]);
    middleLayers = [middleLayers conv2dLayer batchNorm adaptiveMu addLayer leakyrelLayer];
end
```

不要对倒数第二个卷积层应用扩张系数。

```
conv2dLayer = convolution2dLayer(3,numberOfFilters, ...
    'Padding', 1, 'Weights', Wgts, 'Name', 'Conv9');
```

```

batchNorm = batchNormalizationLayer('Name','AN9');
adaptiveMu = adaptiveNormalizationMu(numberOfFilters,'Mu9');
addLayer = additionLayer(2,'Name','add9');
leakyrelLayer = leakyReluLayer(0.2,'Name','Leaky9');
middleLayers = [middleLayers conv2dLayer batchNorm adaptiveMu addLayer leakyrelLayer];

```

最后一个卷积层有一个大小为  $1 \times 1 \times 32 \times 3$  的滤波器，用于重新构造图像。

```

Wgts = sqrt(2/(9*numberOfFilters))*randn(1,1,numberOfFilters,3);
conv2dLayer = convolution2dLayer(1,3,'NumChannels',numberOfFilters, ...
    'Weights',Wgts,'Name','Conv10');

```

最后一层是一个回归层，而不是泄漏 ReLU 层。该回归层计算双边滤波后的图像和网络预测之间的均方误差。

```

finalLayers = [conv2dLayer
    regressionLayer('Name','FinalRegressionLayer')
];

```

串联所有层。

```

layers = [firstLayer middleLayers finalLayers'];
lgraph = layerGraph(layers);

```

创建跳跃连接，用作自适应归一化方程恒等分支。将跳跃连接与相加层相连。

```

skipConv1 = adaptiveNormalizationLambda(numberOfFilters,'Lambda1');
skipConv2 = adaptiveNormalizationLambda(numberOfFilters,'Lambda2');
skipConv3 = adaptiveNormalizationLambda(numberOfFilters,'Lambda3');
skipConv4 = adaptiveNormalizationLambda(numberOfFilters,'Lambda4');
skipConv5 = adaptiveNormalizationLambda(numberOfFilters,'Lambda5');
skipConv6 = adaptiveNormalizationLambda(numberOfFilters,'Lambda6');
skipConv7 = adaptiveNormalizationLambda(numberOfFilters,'Lambda7');
skipConv8 = adaptiveNormalizationLambda(numberOfFilters,'Lambda8');
skipConv9 = adaptiveNormalizationLambda(numberOfFilters,'Lambda9');

lgraph = addLayers(lgraph,skipConv1);
lgraph = connectLayers(lgraph,'Conv1','Lambda1');
lgraph = connectLayers(lgraph,'Lambda1','add1/in2');

lgraph = addLayers(lgraph,skipConv2);
lgraph = connectLayers(lgraph,'Conv2','Lambda2');
lgraph = connectLayers(lgraph,'Lambda2','add2/in2');

lgraph = addLayers(lgraph,skipConv3);
lgraph = connectLayers(lgraph,'Conv3','Lambda3');
lgraph = connectLayers(lgraph,'Lambda3','add3/in2');

lgraph = addLayers(lgraph,skipConv4);
lgraph = connectLayers(lgraph,'Conv4','Lambda4');
lgraph = connectLayers(lgraph,'Lambda4','add4/in2');

lgraph = addLayers(lgraph,skipConv5);
lgraph = connectLayers(lgraph,'Conv5','Lambda5');
lgraph = connectLayers(lgraph,'Lambda5','add5/in2');

lgraph = addLayers(lgraph,skipConv6);

```

```

lgraph = connectLayers(lgraph,'Conv6','Lambda6');
lgraph = connectLayers(lgraph,'Lambda6','add6/in2');

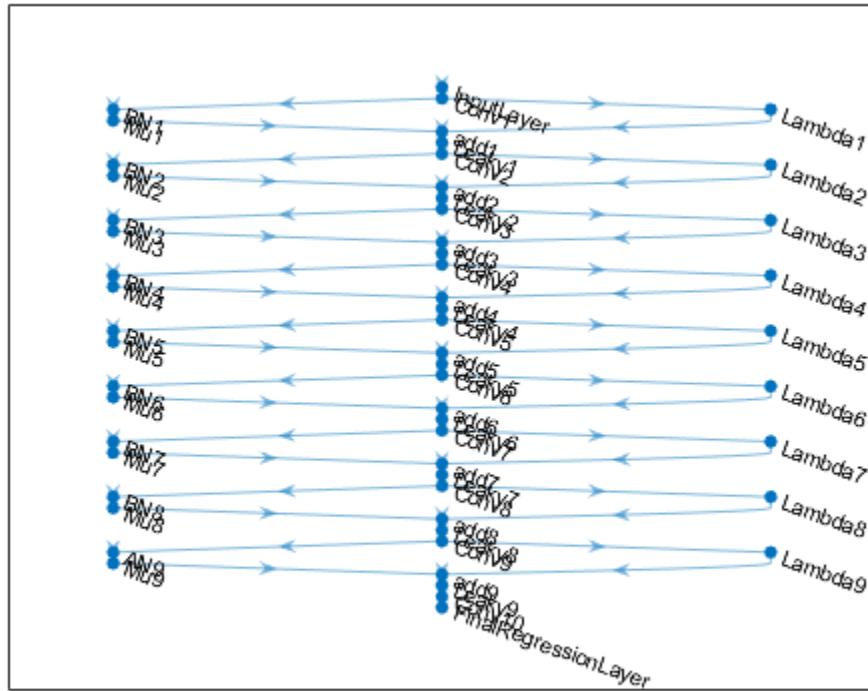
lgraph = addLayers(lgraph,skipConv7);
lgraph = connectLayers(lgraph,'Conv7','Lambda7');
lgraph = connectLayers(lgraph,'Lambda7','add7/in2');

lgraph = addLayers(lgraph,skipConv8);
lgraph = connectLayers(lgraph,'Conv8','Lambda8');
lgraph = connectLayers(lgraph,'Lambda8','add8/in2');

lgraph = addLayers(lgraph,skipConv9);
lgraph = connectLayers(lgraph,'Conv9','Lambda9');
lgraph = connectLayers(lgraph,'Lambda9','add9/in2');

绘制层次图。
plot(lgraph)

```



### 指定训练选项

使用 Adam 优化器训练网络。使用 `trainingOptions` 函数指定超参数设置。对 “Momentum” 使用默认值 0.9，对 “L2Regularization” 使用 0.0001（权重衰减）。指定 0.0001 的恒定学习率。进行 181 轮训练。

```

maxEpochs = 181;
initLearningRate = 0.0001;
miniBatchSize = 1;

```

```

options = trainingOptions('adam', ...
    'InitialLearnRate',initLearningRate, ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'Plots','training-progress', ...
    'Verbose',false);

```

## 训练网络

默认情况下，该示例加载一个逼近双边滤波器的预训练多尺度 CAN。预训练网络可让您无需等待训练完成即可执行双边滤波逼近。

要训练网络，请将以下代码中的 `doTraining` 变量设置为 `true`。使用 `trainNetwork` 函数训练多尺度 CAN。

在 GPU 上（如果有）进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。在 NVIDIA™ Titan X 上训练大约需要 15 个小时。

```

doTraining = false;
if doTraining
    modelDateTime = string(datetime('now','Format','yyyy-MM-dd-HH-mm-ss'));
    net = trainNetwork(dsTrain,lgraph,options);
    save(strcat("trainedOperatorLearning-",modelDateTime,"-Epoch-",num2str(maxEpochs),".mat"),'net');
else
    load('trainedOperatorLearning-Epoch-181.mat');
end

```

## 使用多尺度 CAN 执行双边滤波逼近

要使用训练用于逼近双边滤波器的多尺度 CAN 网络处理图像，请按照此示例的后续步骤操作。该示例的后续部分显示如何：

- 基于参考图像创建一个含噪示例输入图像。
- 使用 `imbilatfilt` (Image Processing Toolbox) 函数对含噪图像执行常规双边滤波。
- 使用 CAN 对含噪图像执行双边滤波逼近。
- 以可视化方式比较算子逼近和常规双边滤波的去噪图像。
- 量化这些图像与原始参考图像的相似性，以此评估去噪图像的质量。

## 创建示例含噪图像

创建一个示例含噪图像，它将用于比较算子逼近和常规双边滤波的结果。测试数据集 `testImages` 包含 Image Processing Toolbox™ 中提供的 21 个原始图像。将这些图像加载到 `imageDatastore` 中。

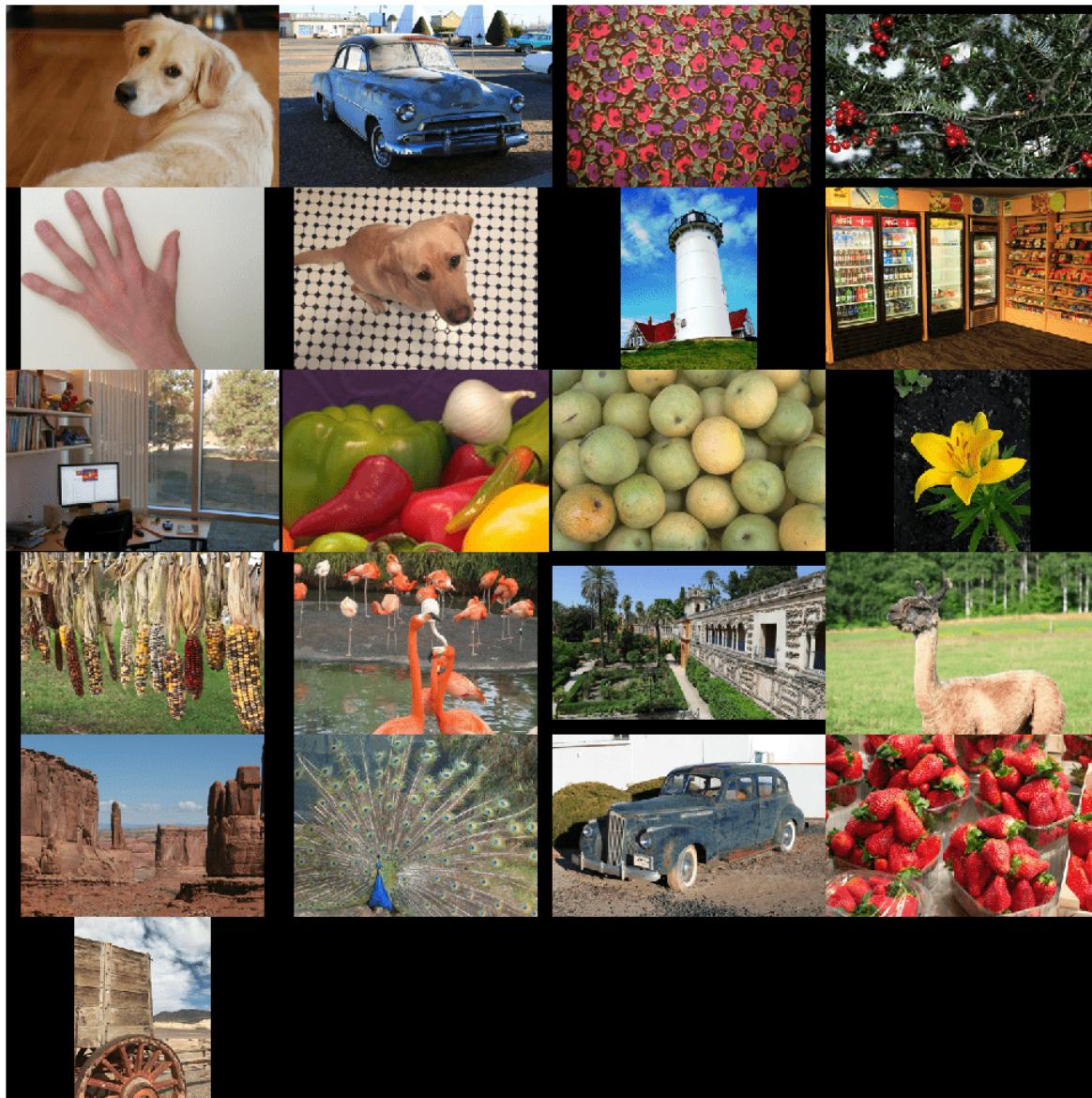
```

exts = {'jpg','png'};
fileNames = {'sherlock.jpg','car2.jpg','fabric.png','greens.jpg','hands1.jpg','kobi.png',...
    'lighthouse.png','micromarket.jpg','office_4.jpg','onion.png','pears.png','yellowlily.jpg',...
    'indiancorn.jpg','flamingos.jpg','sevilla.jpg','llama.jpg','parkavenue.jpg',...
    'peacock.jpg','car1.jpg','strawberries.jpg','wagon.jpg'};
filePath = [fullfile(matlabroot,'toolbox','images','imdata') filesep];
filePathNames = strcat(filePath,fileNames);
testImages = imageDatastore(filePathNames,'FileExtensions',exts);

```

将测试图像显示为蒙太奇。

```
montage(testImages)
```



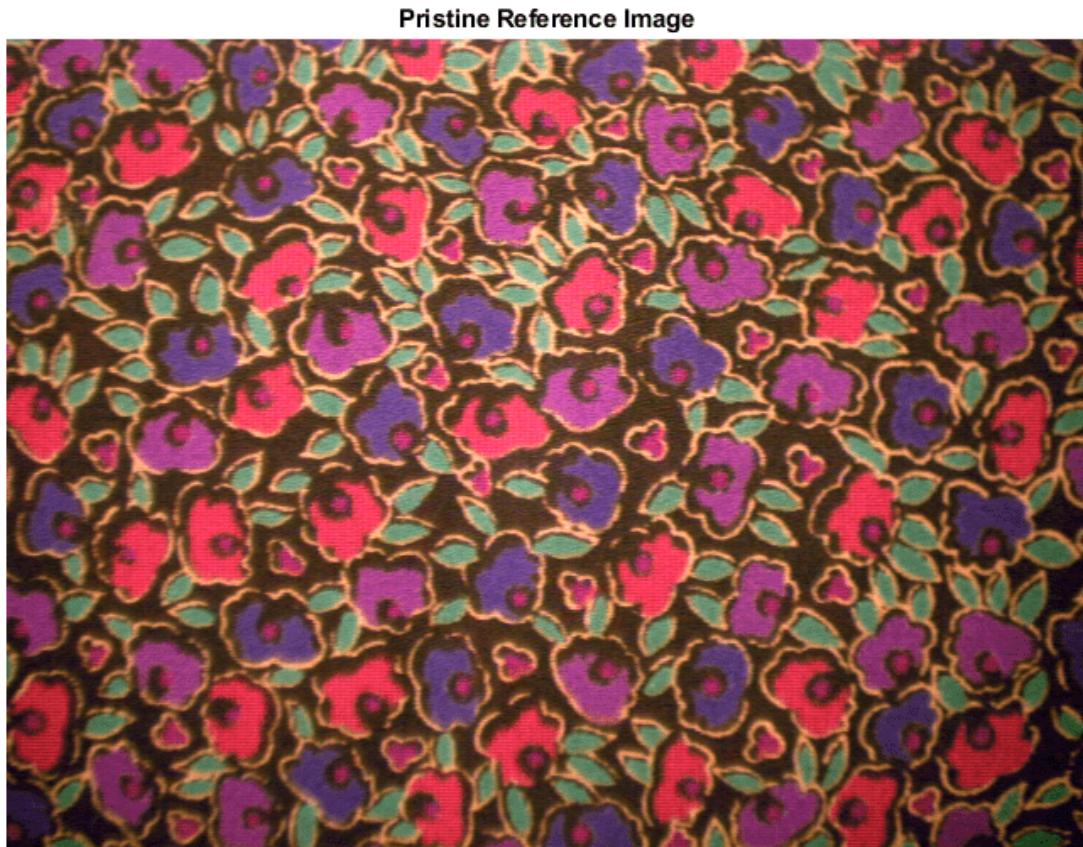
选择其中一个图像作为双边滤波的参考图像。将图像转换为数据类型 `uint8`。

```
indx = 3; % Index of image to read from the test image datastore
Ireference = readimage(testImages,indx);
Ireference = im2uint8(Ireference);
```

您也可以自选其他图像作为参考图像。请注意，测试图像的大小必须至少为  $256 \times 256$ 。如果测试图像小于  $256 \times 256$ ，则使用 `imresize` (Image Processing Toolbox) 函数增加图像大小。网络还要求测试图像为 RGB。如果测试图像为灰阶图，则使用 `cat` 函数沿第三个维度串联原始图像的三个副本，以将图像转换为 RGB。

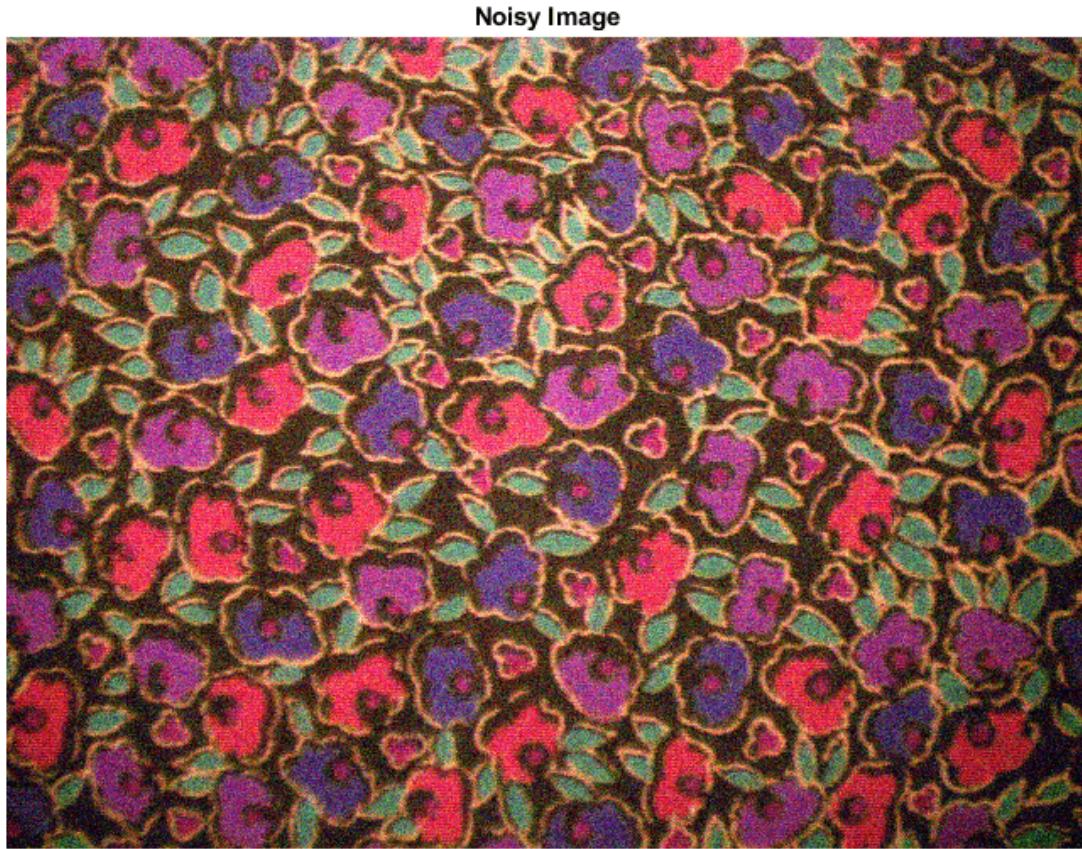
显示参考图像。

```
imshow(Ireference)  
title('Pristine Reference Image')
```



使用 `imnoise` (Image Processing Toolbox) 函数将方差为 0.00001 的零均值高斯白噪声添加到参考图像中。

```
Inoisy = imnoise(Ireference,'gaussian',0.00001);  
imshow(Inoisy)  
title('Noisy Image')
```



### 使用双边滤波对图像进行滤波

常规的双边滤波是在保持边缘锐度的同时降低图像噪声的标准方法。使用 `imbilatfilt` (Image Processing Toolbox) 函数对含噪图像应用双边滤波器。指定与像素值方差相同的平滑度。

```
degreeOfSmoothing = var(double(Inoisy(:)));
Ibilat = imbilatfilt(Inoisy,degreeOfSmoothing);
imshow(Ibilat)
title('Denoised Image Obtained Using Bilateral Filtering')
```

Denoised Image Obtained Using Bilateral Filtering



### 使用经过训练的网络处理图像

将经过训练的网络应用于归一化的输入图像，并观察来自最后一层（回归层）的 **activations**。网络的输出是所需的去噪图像。

```
Iapprox = activations(net,Inoisy,'FinalRegressionLayer');
```

Image Processing Toolbox™ 要求浮点图像的像素值在 [0, 1] 范围内。使用 **rescale** 函数将像素值缩放到此范围，然后将图像转换为 **uint8**。

```
Iapprox = rescale(Iapprox);
Iapprox = im2uint8(Iapprox);
imshow(Iapprox)
title('Denoised Image Obtained Using Multiscale CAN')
```

Denoised Image Obtained Using Multiscale CAN



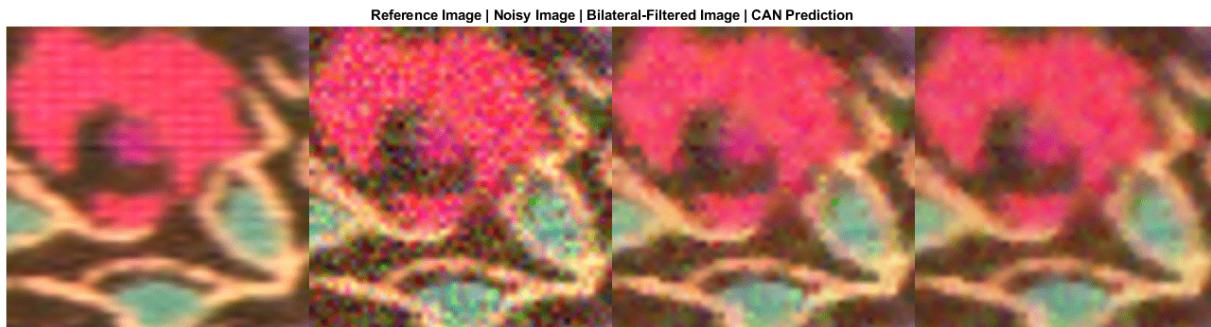
### 可视化和定量比较

为了在视觉上更好地理解去噪后的图像，请检查每个图像中的一个小区域。使用向量 **roi** 指定关注区域 (ROI)，格式为 [x y 宽度 高度]。其中的元素定义 ROI 左上角的 x 坐标和 y 坐标，以及它的宽度和高度。

```
roi = [300 30 50 50];
```

将图像裁剪到该 ROI，并将结果显示为蒙太奇。

```
montage({imcrop(Ireference,roi),imcrop(Inoisy,roi), ...
    imcrop(Ibilat,roi),imcrop(Iapprox,roi)}, ...
    'Size',[1 4]);
title('Reference Image | Noisy Image | Bilateral-Filtered Image | CAN Prediction');
```



与常规的双边滤波相比，CAN 去除了更多噪声。这两种方法都保留了边缘锐度。

使用图像质量指标来定量比较含噪输入图像、双边滤波图像和算子逼近图像。参考图像是在添加噪声之前的原始参考图像 `Ireference`。

对照参考图像测量每个图像的峰值信噪比 (PSNR)。PSNR 值越大，通常表示图像质量越好。有关该指标的详细信息，请参阅 `psnr` (Image Processing Toolbox)。

```
noisyPSNR = psnr(Inoisy,Ireference);
bilatPSNR = psnr(Ibilat,Ireference);
approxPSNR = psnr(Iapprox,Ireference);
disp(['PSNR of: Noisy Image / Bilateral-Filtered Image / Operator Approximated Image = ', ...
    num2str([noisyPSNR bilatPSNR approxPSNR])])
```

```
PSNR of: Noisy Image / Bilateral-Filtered Image / Operator Approximated Image = 20.2857    25.7978    26.2011
```

测量每个图像的结构相似性指数 (SSIM)。SSIM 对照参考图像评估图像三个特性的视觉效果：亮度、对比度和结构。SSIM 值越接近 1，测试图像与参考图像越一致。有关该指标的详细信息，请参阅 `ssim` (Image Processing Toolbox)。

```
noisySSIM = ssim(Inoisy,Ireference);
bilatSSIM = ssim(Ibilat,Ireference);
approxSSIM = ssim(Iapprox,Ireference);
disp(['SSIM of: Noisy Image / Bilateral-Filtered Image / Operator Approximated Image = ', ...
    num2str([noisySSIM bilatSSIM approxSSIM])])
```

```
SSIM of: Noisy Image / Bilateral-Filtered Image / Operator Approximated Image = 0.76251    0.91576    0.92663
```

使用自然图像质量评价方法 (NIQE) 测量图像感知质量。NIQE 分数越小，表示感知质量越好。有关该指标的详细信息，请参阅 `niqe` (Image Processing Toolbox)。

```
noisyNIQE = niqe(Inoisy);
bilatNIQE = niqe(Ibilat);
approxNIQE = niqe(Iapprox);
disp(['NIQE score of: Noisy Image / Bilateral-Filtered Image / Operator Approximated Image = ', ...
    num2str([noisyNIQE bilatNIQE approxNIQE])])
```

```
NIQE score of: Noisy Image / Bilateral-Filtered Image / Operator Approximated Image = 12.1865    7.22606    6.1
```

与常规的双边滤波相比，算子逼近可产生更好的指标分数。

### 参考资料

- [1] Chen, Q. J. Xu, and V. Koltun."Fast Image Processing with Fully-Convolutional Networks."In Proceedings of the 2017 IEEE Conference on Computer Vision.Venice, Italy, Oct. 2017, pp. 2516-2525.
- [2] Grubinger, M., P. Clough, H. Müller, and T. Deselaers."The IAPR TC-12 Benchmark:A New Evaluation Resource for Visual Information Systems."Proceedings of the OntolImage 2006 Language Resources For Content-Based Image Retrieval.Genoa, Italy.Vol. 5, May 2006, p. 10.

### 另请参阅

`randomPatchExtractionDatastore` | `trainNetwork` | `trainingOptions` | `layerGraph` |  
`activations` | `imbilatfilt` | `imageDatastore`

### 详细信息

- “预处理图像以进行深度学习” (第 18-2 页)
- “Datastores for Deep Learning”
- “深度学习层列表” (第 1-18 页)

# 使用深度学习的神经样式迁移

此示例说明如何使用预训练的 VGG-19 网络 [1] (第 9-0 页) 将一个图像的样式外观应用于另一个图像的场景内容。

## 加载数据

加载样式图像和内容图像。此示例使用梵高的名作《星夜》作为样式图像，使用一个灯塔照片作为内容图像。

```
styleImage = im2double(imread('starryNight.jpg'));
contentImage = imread('lighthouse.png');
```

以蒙太奇方式显示样式图像和内容图像。

```
imshow(imtile({styleImage,contentImage},'BackgroundColor','w'));
```



## 加载特征提取网络

在此示例中，您使用一个经过修改的预训练 VGG-19 深度神经网络在不同层提取内容图像和样式图像的特征。这些多层特征用于计算各自的内容和样式损失。该网络使用组合损失生成样式化的迁移图像。

要获得一个预训练 VGG-19 网络，请安装 `vgg19`。如果没有安装所需的支持包，软件会提供下载链接。

```
net = vgg19;
```

要使 VGG-19 网络适合特征提取，请从网络中删除所有全连接层。

```
lastFeatureLayerIdx = 38;
layers = net.Layers;
layers = layers(1:lastFeatureLayerIdx);
```

VGG-19 网络的最大池化层会造成衰落效应。要降低衰落效应并增加梯度流，请将所有最大池化层替换为平均池化层 [1] (第 9-0 页)。

```
for l = 1:lastFeatureLayerIdx
    layer = layers(l);
```

```

if isa(layer,'nnet.cnn.layer.MaxPooling2DLayer')
    layers(l) = averagePooling2dLayer(layer.PoolSize,'Stride',layer.Stride,'Name',layer.Name);
end
end

```

使用修改后的层创建一个层次图。

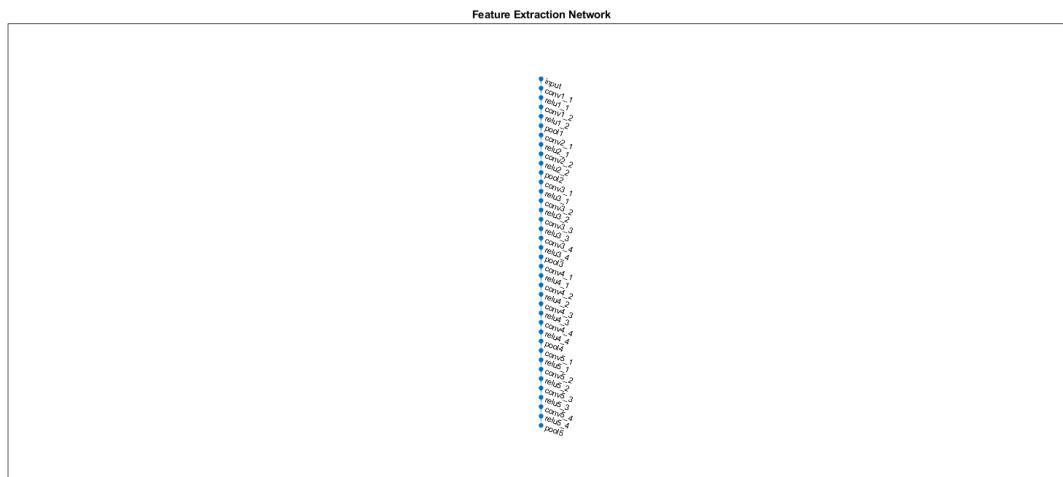
```
lgraph = layerGraph(layers);
```

在图中可视化特征提取网络。

```

plot(lgraph)
title('Feature Extraction Network')

```



要使用自定义训练循环训练网络并支持自动微分，请将层次图转换为 **dlnetwork** 对象。

```
dlnet = dlnetwork(lgraph);
```

### 预处理数据

将样式图像和内容图像的大小调整为较小的大小，以加快处理速度。

```

imageSize = [384,512];
styleImg = imresize(styleImage,imageSize);
contentImg = imresize(contentImage,imageSize);

```

预训练的 VGG-19 网络对减去了通道均值的图像执行分类。从图像输入层获取通道均值，图像输入层是网络中的第一层。

```

imgInputLayer = lgraph.Layers(1);
meanVggNet = imgInputLayer.Mean(1,1,:);

```

通道均值适用于像素值在 [0, 255] 范围内的浮点数据类型的图像。将样式图像和内容图像转换为范围在 [0, 255] 范围内的数据类型 **single**。然后，从样式图像和内容图像中减去通道均值。

```

styleImg = rescale(single(styleImg),0,255) - meanVggNet;
contentImg = rescale(single(contentImg),0,255) - meanVggNet;

```

## 初始化迁移图像

迁移图像是作为样式迁移结果的输出图像。您可以使用样式图像、内容图像或任何随机图像来初始化迁移图像。使用样式图像或内容图像初始化会使样式迁移处理发生偏置，并产生与输入图像更相似的迁移图像。相反，使用白噪声初始化可以消除偏置，但需要更长时间才能收敛于样式化图像。为了实现更好的样式化和更快的收敛，此示例将输出迁移图像初始化为内容图像和白噪声图像的加权组合。

```
noiseRatio = 0.7;
randImage = randi([-20,20],[imageSize 3]);
transferImage = noiseRatio.*randImage + (1-noiseRatio).*contentImg;
```

## 定义损失函数和样式迁移参数

### 内容损失

内容损失的目标是使迁移图像的特征与内容图像的特征相匹配。内容损失计算的是每个内容特征层的内容图像特征与迁移图像特征之间的均方差 [1]（第 9-0 页）。 $\hat{Y}$  是迁移图像的预测特征图， $Y$  是内容图像的预测特征图。 $W_c^l$  是  $l^{th}$  层的内容层权重。 $H, W, C$  分别是特征图的高度、宽度和通道。

$$L_{content} = \sum_l W_c^l \times \frac{1}{HWC} \sum_{i,j} (\hat{Y}_{i,j}^l - Y_{i,j}^l)^2$$

指定内容特征提取层的名称。从这些层提取的特征用于计算内容损失。在 VGG-19 网络中，使用更深层的特征比使用浅层的特征更高效。因此，将内容特征提取层指定为第四个卷积层。

```
styleTransferOptions.contentFeatureLayerNames = {'conv4_2'};
```

指定内容特征提取层的权重。

```
styleTransferOptions.contentFeatureLayerWeights = 1;
```

### 样式损失

样式损失的目标是使迁移图像的纹理与样式图像的纹理相匹配。图像的样式表示为一个 Gram 矩阵。因此，样式损失计算的是样式图像的 Gram 矩阵与迁移图像的 Gram 矩阵之间的均方差 [1]（第 9-0 页）。 $Z$  和  $\hat{Z}$  分别是样式图像和迁移图像的预测特征图。 $G_Z$  和  $G_{\hat{Z}}$  分别是样式特征和迁移特征的 Gram 矩阵。 $W_s^l$  是  $l^{th}$  样式层的样式层权重。

$$G_{\hat{Z}} = \sum_{i,j} \hat{Z}_{i,j} \times \hat{Z}_{j,i}$$

$$G_Z = \sum_{i,j} Z_{i,j} \times Z_{j,i}$$

$$L_{style} = \sum_l W_s^l \times \frac{1}{(2HWC)^2} \sum (G_{\hat{Z}}^l - G_Z^l)^2$$

指定样式特征提取层的名称。从这些层中提取的特征用于计算样式损失。

```
styleTransferOptions.styleFeatureLayerNames = {'conv1_1','conv2_1','conv3_1','conv4_1','conv5_1'};
```

指定样式特征提取层的权重。为简单样式图像指定较小的权重，为复杂样式图像增大权重。

```
styleTransferOptions.styleFeatureLayerWeights = [0.5,1.0,1.5,3.0,4.0];
```

## 总损失

总损失是内容损失和样式损失的加权组合。 $\alpha$  和  $\beta$  分别是内容损失和样式损失的权重因子。

$$L_{total} = \alpha \times L_{content} + \beta \times L_{style}$$

为内容损失和样式损失指定权重因子 **alpha** 和 **beta**。**alpha** 与 **beta** 的比率应在 1e-3 或 1e-4 左右 [1] (第 9-0 页)。

```
styleTransferOptions.alpha = 1;
styleTransferOptions.beta = 1e3;
```

## 指定训练选项

进行 2500 次迭代的训练。

```
numIterations = 2500;
```

指定 Adam 优化的选项。将学习率设置为 2 以加快收敛速度。您可以通过观察输出图像和损失来尝试调整学习率。用 [] 初始化尾部平均梯度和尾部平均梯度平方衰减率。

```
learningRate = 2;
trailingAvg = [];
trailingAvgSq = [];
```

## 训练网络

将样式图像、内容图像和迁移图像转换为基础类型为 **single** 和维度标签为 'SSC' 的 **dlarray** 对象。

```
dlStyle = dlarray(styleImg,'SSC');
dlContent = dlarray(contentImg,'SSC');
dlTransfer = dlarray(transferImage,'SSC');
```

在 GPU 上 (如果有) 进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。对于 GPU 训练，需将数据转换为 **gpuArray**。

```
if canUseGPU
    dlContent = gpuArray(dlContent);
    dlStyle = gpuArray(dlStyle);
    dlTransfer = gpuArray(dlTransfer);
end
```

从内容图像中提取内容特征。

```
numContentFeatureLayers = numel(styleTransferOptions.contentFeatureLayerNames);
contentFeatures = cell(1,numContentFeatureLayers);
[contentFeatures{:}] = forward(dlnet,dlContent,'Outputs',styleTransferOptions.contentFeatureLayerNames);
```

从样式图像中提采样式特征。

```
numStyleFeatureLayers = numel(styleTransferOptions.styleFeatureLayerNames);
styleFeatures = cell(1,numStyleFeatureLayers);
[styleFeatures{:}] = forward(dlnet,dlStyle,'Outputs',styleTransferOptions.styleFeatureLayerNames);
```

使用自定义训练循环训练模型。对于每次迭代：

- 使用内容图像、样式图像和迁移图像的特征来计算内容损失和样式损失。要计算损失和梯度，请使用辅助函数 **imageGradients** (在此示例的支持函数 (第 9-0 页) 部分中定义)。

- 使用 `adamupdate` 函数更新迁移图像。
- 选择最佳样式迁移图像作为最终输出图像。

```
figure
```

```
minimumLoss = inf;
```

```
for iteration = 1:numIterations
```

```
    % Evaluate the transfer image gradients and state using dlfeval and the  
    % imageGradients function listed at the end of the example.
```

```
[grad,losses] = dlfeval(@imageGradients,dlnet,dlTransfer,contentFeatures,styleFeatures,styleTransferOptions  
[dlTransfer,trailingAvg,trailingAvgSq] = adamupdate(dlTransfer,grad,trailingAvg,trailingAvgSq,iteration,lea
```

```
if losses.totalLoss < minimumLoss
```

```
    minimumLoss = losses.totalLoss;
```

```
    dlOutput = dlTransfer;
```

```
end
```

```
% Display the transfer image on the first iteration and after every 50
```

```
% iterations. The postprocessing steps are described in the "Postprocess
```

```
% Transfer Image for Display" section of this example.
```

```
if mod(iteration,50) == 0 || (iteration == 1)
```

```
transferImage = gather(extractdata(dlTransfer));
```

```
transferImage = transferImage + meanVggNet;
```

```
transferImage = uint8(transferImage);
```

```
transferImage = imresize(transferImage,size(contentImage,[1 2]));
```

```
image(transferImage)
```

```
title(['Transfer Image After Iteration ',num2str(iteration)])
```

```
axis off image
```

```
drawnow
```

```
end
```

```
end
```

Transfer Image After Iteration 2500



### 后处理迁移图像以用于显示

获取更新后的迁移图像。

```
transferImage = gather(extractdata(dlOutput));
```

为迁移图像加上网络训练后的均值。

```
transferImage = transferImage + meanVggNet;
```

某些像素值可能会超过内容和样式图像的原始范围 [0, 255]。通过将数据类型转换为 uint8，您可以将值裁剪到 [0, 255] 的范围之内。

```
transferImage = uint8(transferImage);
```

将迁移图像的大小调整为内容图像的原始大小。

```
transferImage = imresize(transferImage,size(contentImage,[1 2]));
```

以蒙太奇方式显示内容图像、迁移图像和样式图像。

```
imshow(imtile({contentImage,transferImage,styleImage}, ...
    'GridSize',[1 3],'BackgroundColor','w'));
```



## 支持函数

### 计算图像损失和梯度

`imageGradients` 辅助函数使用内容图像、样式图像和迁移图像的特征来返回损失和梯度。

```
function [gradients,losses] = imageGradients(dlnet,dITransfer,contentFeatures,styleFeatures,params)

    % Initialize transfer image feature containers.
    numContentFeatureLayers = numel(params.contentFeatureLayerNames);
    numStyleFeatureLayers = numel(params.styleFeatureLayerNames);

    transferContentFeatures = cell(1,numContentFeatureLayers);
    transferStyleFeatures = cell(1,numStyleFeatureLayers);

    % Extract content features of transfer image.
    [transferContentFeatures{:}] = forward(dlnet,dITransfer,'Outputs',params.contentFeatureLayerNames);

    % Extract style features of transfer image.
    [transferStyleFeatures{:}] = forward(dlnet,dITransfer,'Outputs',params.styleFeatureLayerNames);

    % Compute content loss.
    cLoss = contentLoss(transferContentFeatures,contentFeatures,params.contentFeatureLayerWeights);

    % Compute style loss.
    sLoss = styleLoss(transferStyleFeatures,styleFeatures,params.styleFeatureLayerWeights);

    % Compute final loss as weighted combination of content and style loss.
    loss = (params.alpha * cLoss) + (params.beta * sLoss);

    % Calculate gradient with respect to transfer image.
    gradients = dlgradient(loss,dITransfer);

    % Extract various losses.
    losses.totalLoss = gather(extractdata(loss));
```

```

losses.contentLoss = gather(extractdata(cLoss));
losses.styleLoss = gather(extractdata(sLoss));

end

```

### 计算内容损失

**contentLoss** 辅助函数计算内容图像特征和迁移图像特征之间的加权均方差。

```

function loss = contentLoss(transferContentFeatures,contentFeatures,contentWeights)

loss = 0;
for i=1:numel(contentFeatures)
    temp = 0.5 .* mean((transferContentFeatures{1,i} - contentFeatures{1,i}).^2,'all');
    loss = loss + (contentWeights(i)*temp);
end
end

```

### 计算样式损失

**styleLoss** 辅助函数计算样式图像特征的 Gram 矩阵和迁移图像特征的 Gram 矩阵之间的加权均方差。

```

function loss = styleLoss(transferStyleFeatures,styleFeatures,styleWeights)

loss = 0;
for i=1:numel(styleFeatures)

    tsf = transferStyleFeatures{1,i};
    sf = styleFeatures{1,i};
    [h,w,c] = size(sf);

    gramStyle = computeGramMatrix(sf);
    gramTransfer = computeGramMatrix(tsf);
    sLoss = mean((gramTransfer - gramStyle).^2,'all') / ((h*w*c)^2);

    loss = loss + (styleWeights(i)*sLoss);
end
end

```

### 计算 Gram 矩阵

**styleLoss** 辅助函数使用 **computeGramMatrix** 辅助函数来计算特征图的 Gram 矩阵。

```

function gramMatrix = computeGramMatrix(featureMap)
[H,W,C] = size(featureMap);
reshapedFeatures = reshape(featureMap,H*W,C);
gramMatrix = reshapedFeatures' * reshapedFeatures;
end

```

### 参考资料

[1] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge."A Neural Algorithm of Artistic Style."Preprint, submitted September 2, 2015. <https://arxiv.org/abs/1508.06576>

### 另请参阅

[vgg19](#) | [trainNetwork](#) | [trainingOptions](#) | [dlarray](#)

## 详细信息

- “定义自定义训练循环、损失函数和网络” (第 17-33 页)
- “Specify Training Options in Custom Training Loop”
- “Train Network Using Custom Training Loop”
- “List of Functions with dlarray Support”
- “深度学习层列表” (第 1-18 页)



# 自动驾驶示例

---

## 训练深度学习车辆检测器

此示例说明如何使用深度学习来训练基于视觉的车辆检测器。

### 概述

使用计算机视觉进行车辆检测是对跟踪自主车辆周围车辆的重要环节。许多自动驾驶应用都要求具备车辆检测和车辆跟踪能力，例如前向碰撞警告、自适应巡航控制和自动车道保持。Automated Driving Toolbox™ 提供预训练的车辆检测器 (**vehicleDetectorFasterRCNN** (Automated Driving Toolbox) 和 **vehicleDetectorACF** (Automated Driving Toolbox)) 以实现快速原型。然而，预训练的模型不一定适合每个应用，可能需要您从头开始训练。此示例说明如何使用深度学习从头开始训练车辆检测器。

深度学习是一种功能强大的机器学习方法，可用于训练稳健的目标检测器。目标检测深度学习有多种方法，包括 Faster R-CNN 和 you only look once (YOLO) v2。此示例使用 **trainFasterRCNNObjectDetector** 函数训练 Faster R-CNN 车辆检测器。有关详细信息，请参阅“Object Detection” (Computer Vision Toolbox)。

### 下载预训练的检测器

下载预训练的检测器，避免在训练上花费时间。如果要训练检测器，请将 **doTrainingAndEval** 变量设置为 **true**。

```
doTrainingAndEval = false;
if ~doTrainingAndEval && ~exist('fasterRCNNResNet50EndToEndVehicleExample.mat','file')
    disp('Downloading pretrained detector (118 MB)...');
    pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/fasterRCNNResNet50EndToEndVehicleExample.mat';
    websave('fasterRCNNResNet50EndToEndVehicleExample.mat',pretrainedURL);
end
```

### 加载数据集

此示例使用包含 295 个图像的小型标注数据集。其中许多图像来自加州理工学院的 Caltech Cars 1999 和 2001 数据集，可在加州理工学院计算视觉网站上获得，该网站由 Pietro Perona 创建并经许可使用。每个图像包含一个或两个带标签的车辆实例。您可以借助小型数据集探索 Faster R-CNN 的训练过程，但在实际应用中，您需要更多带标签的图像来训练稳健的检测器。解压缩车辆图像并加载车辆真实值数据。

```
unzip vehicleDatasetImages.zip
data = load('vehicleDatasetGroundTruth.mat');
vehicleDataset = data.vehicleDataset;
```

车辆数据存储在一个包含两列的表中，其中第一列包含图像文件路径，第二列包含车辆边界框。

将数据集分成两部分：一个是用于训练检测器的训练集，一个是用于评估检测器的测试集。选择 60% 的数据进行训练。其余数据用于评估。

```
rng(0)
shuffledIdx = randperm(height(vehicleDataset));
idx = floor(0.6 * height(vehicleDataset));
trainingDataTbl = vehicleDataset(shuffledIdx(1:idx),:);
testDataTbl = vehicleDataset(shuffledIdx(idx+1:end),:);
```

使用 **imageDatastore** 和 **boxLabelDatastore** 创建数据存储，以便在训练和评估期间加载图像和标签数据。

```
imdsTrain = imageDatastore(trainingDataTbl{,:}'imageFilename');
bldsTrain = boxLabelDatastore(trainingDataTbl{,:}'vehicle');
```

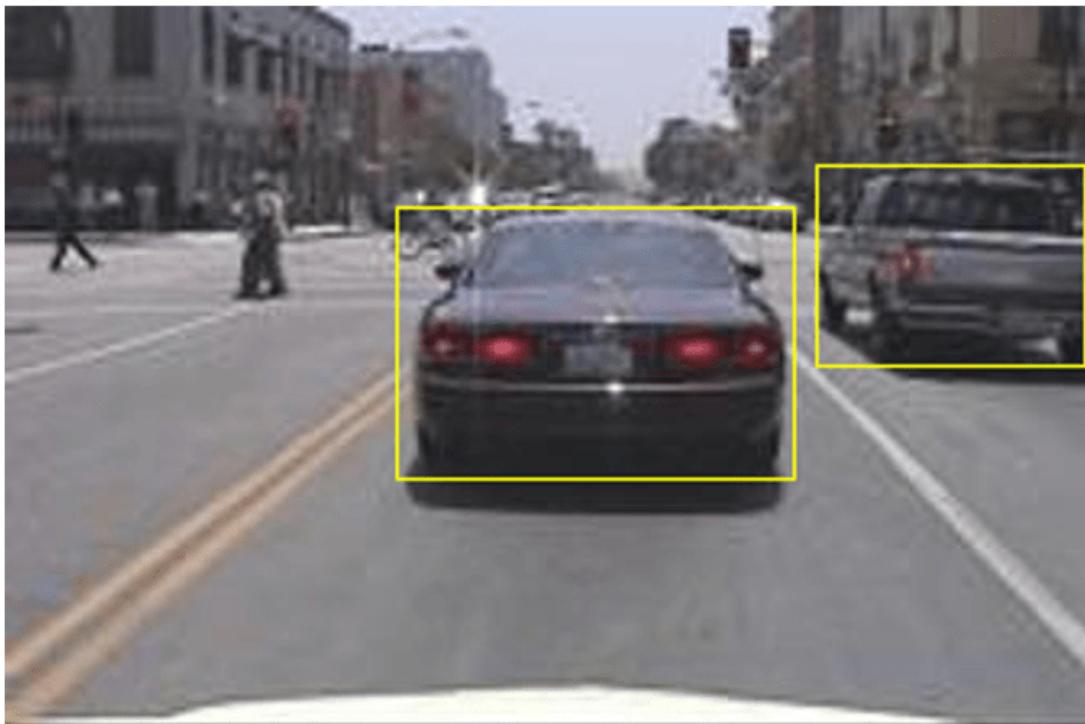
```
imdsTest = imageDatastore(testDataTbl{,:'imageFilename'});  
bldsTest = boxLabelDatastore(testDataTbl{,:'vehicle'});
```

组合图像和边界框标签数据存储。

```
trainingData = combine(imdsTrain,bldsTrain);  
testData = combine(imdsTest,bldsTest);
```

显示其中一个训练图像和边界框标签。

```
data = read(trainingData);  
I = data{1};  
bbox = data{2};  
annotatedImage = insertShape(I,'Rectangle',bbox);  
annotatedImage = imresize(annotatedImage,2);  
figure  
imshow(annotatedImage)
```



### 创建 Faster R-CNN 检测网络

Faster R-CNN 目标检测网络由一个特征提取网络后跟两个子网络组成。特征提取网络通常是一个预训练的 CNN，如 ResNet-50 或 Inception v3。特征提取网络之后的第一个子网络是区域提议网络 (RPN)，该网络经训练用于生成目标提议，即图像中可能存在目标的区域。对第二个子网络进行训练来预测每个目标提议的实际类。

特征提取网络通常是一个预训练的 CNN（有关详细信息，请参阅“预训练的深度神经网络”（第 1-8 页））。此示例使用 ResNet-50 进行特征提取。根据应用要求，也可以使用其他预训练网络，如 MobileNet v2 或 ResNet-18。

使用 `fasterRCNNLayers` 自动根据预训练的特征提取网络创建 Faster R-CNN 网络。`fasterRCNNLayers` 要求您指定几个用于参数化 Faster R-CNN 网络的输入：

- 网络输入大小
- 锚框
- 特征提取网络

首先，指定网络输入大小。选择网络输入大小时，请考虑运行网络本身所需的最低大小、训练图像的大小以及基于所选大小处理数据所产生的计算成本。如果可行，请选择接近训练图像大小且大于网络所需输入大小的网络输入大小。为了降低运行示例的计算成本，请指定网络输入大小为 [224 224 3]，这是运行网络所需的最低大小。

```
inputSize = [224 224 3];
```

请注意，此示例中使用的训练图像大于  $224 \times 224$ ，并且大小不同，因此您必须在训练前的预处理步骤中调整图像的大小。

接下来，使用 `estimateAnchorBoxes` 根据训练数据中目标的大小来估计锚框。考虑到训练前会对图像大小进行调整，用来估计锚框的训练数据的大小也要调整。使用 `transform` 预处理训练数据，然后定义锚框数量并估计锚框。

```
preprocessedTrainingData = transform(trainingData, @(data)preprocessData(data,inputSize));
numAnchors = 4;
anchorBoxes = estimateAnchorBoxes(preprocessedTrainingData,numAnchors)

anchorBoxes = 4×2
96   91
68   65
150  125
38   29
```

有关选择锚框的详细信息，请参阅“Estimate Anchor Boxes From Training Data”（Computer Vision Toolbox）（Computer Vision Toolbox™）和“Anchor Boxes for Object Detection”（Computer Vision Toolbox）。

现在，使用 `resnet50` 加载预训练的 ResNet-50 模型。

```
featureExtractionNetwork = resnet50;
```

选择 '`activation_40_relu`' 作为特征提取层。此特征提取层输出以 16 为因子的下采样特征图。该下采样量是空间分辨率和提取特征强度之间一个很好的折中，因为在网络更深层提取的特征能够对更强的图像特征进行编码，但以空间分辨率为代价。选择最佳特征提取层需要依靠经验分析。您可以使用 `analyzeNetwork` 查找网络中其他潜在特征提取层的名称。

```
featureLayer = 'activation_40_relu';
```

定义要检测的类的数量。

```
numClasses = width(vehicleDataset)-1;
```

创建 Faster R-CNN 目标检测网络。

```
lgraph = fasterRCNNLayers(inputSize,numClasses,anchorBoxes,featureExtractionNetwork,featureLayer);
```

您可以使用 `analyzeNetwork` 或 Deep Learning Toolbox™ 中的深度网络设计器来可视化网络。

如果需要对 Faster R-CNN 网络架构进行更多控制，请使用深度网络设计器手动设计 Faster R-CNN 检测网络。有关详细信息，请参阅 “Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN” (Computer Vision Toolbox)。

### 数据增强

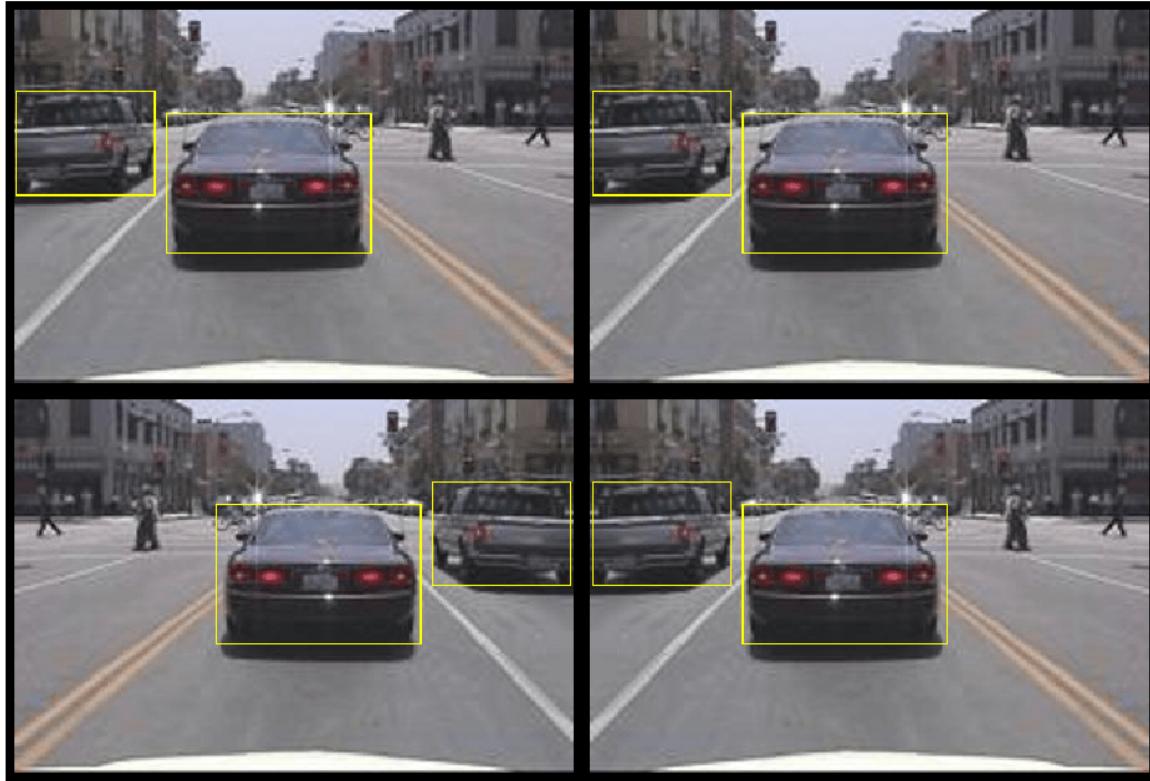
数据增强可通过在训练期间随机变换原始数据来提高网络准确度。通过使用数据增强，您可以为训练数据添加更多变化，但又不必增加带标签的训练样本的数量。

使用 `transform` 通过随机水平翻转图像和相关边界框标签来增强训练数据。请注意，数据增强不适用于测试数据。理想情况下，测试数据代表原始数据并且保持不变，以便进行无偏置的评估。

```
augmentedTrainingData = transform(trainingData,@augmentData);
```

多次读取同一图像，并显示增强的训练数据。

```
augmentedData = cell(4,1);
for k = 1:4
    data = read(augmentedTrainingData);
    augmentedData{k} = insertShape(data{1},'Rectangle',data{2});
    reset(augmentedTrainingData);
end
figure
montage(augmentedData,'BorderSize',10)
```



### 预处理训练数据

预处理增强的训练数据以准备进行训练。

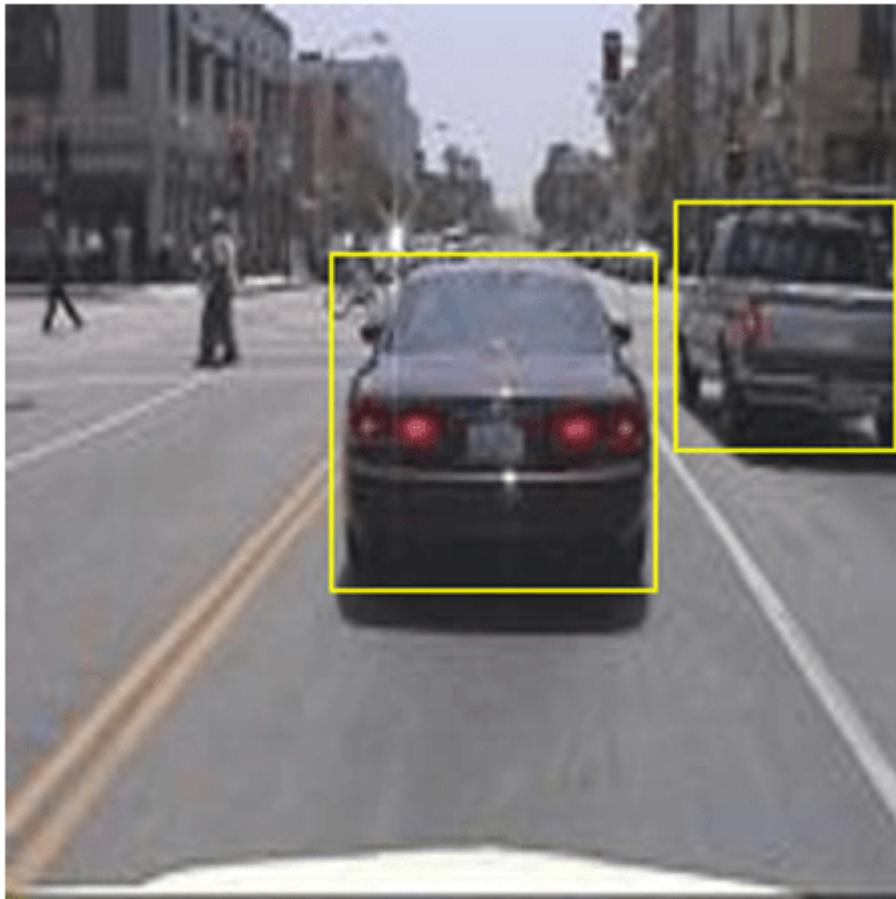
```
trainingData = transform(augmentedTrainingData,@(data)preprocessData(data,inputSize));
```

读取预处理的数据。

```
data = read(trainingData);
```

显示图像和边界框。

```
I = data{1};  
bbox = data{2};  
annotatedImage = insertShape(I,'Rectangle',bbox);  
annotatedImage = imresize(annotatedImage,2);  
figure  
imshow(annotatedImage)
```



## 训练 Faster R-CNN

使用 `trainingOptions` 指定网络训练选项。将 '`CheckpointPath`' 设置为临时位置。这样可在训练过程中保存经过部分训练的检测器。如果由于停电或系统故障等原因导致训练中断，您可以从保存的检查点继续训练。

```
options = trainingOptions('sgdm',...
    'MaxEpochs',7,...
    'MiniBatchSize',1,...
    'InitialLearnRate',1e-3,...
    'CheckpointPath',tempdir);
```

如果 `doTrainingAndEval` 为 `true`，则使用 `trainFasterRCNNObjectDetector` 训练 Faster R-CNN 目标检测器。否则，加载预训练的网络。

```
if doTrainingAndEval
    % Train the Faster R-CNN detector.
    % * Adjust NegativeOverlapRange and PositiveOverlapRange to ensure
    %   that training samples tightly overlap with ground truth.
```

```
[detector, info] = trainFasterRCNNObjectDetector(trainingData,lgraph,options, ...
    'NegativeOverlapRange',[0 0.3], ...
    'PositiveOverlapRange',[0.6 1]);
else
    % Load pretrained detector for the example.
    pretrained = load('fasterRCNNResNet50EndToEndVehicleExample.mat');
    detector = pretrained.detector;
end
```

此示例在具有 12 GB 内存的 Nvidia(TM) Titan X GPU 上进行了验证。训练网络需要大约 20 分钟。具体训练时间因您使用的硬件而异。

对一个测试图像运行检测器以进行快速检查。确保将图像的大小调整为与训练图像相同。

```
I = imread(testDataTbl.imageFilename{1});
I = imresize(I,inputSize(1:2));
[bboxes,scores] = detect(detector,I);
```

显示结果。

```
I = insertObjectAnnotation(I,'rectangle',bboxes,scores);
figure
imshow(I)
```



### 使用测试集评估检测器

基于大量图像评估经过训练的目标检测器以测量其性能。Computer Vision Toolbox™ 提供目标检测器评估函数，用于测量常见指标，如平均精确率 (`evaluateDetectionPrecision`) 和对数平均泄漏检率 (`evaluateDetectionMissRate`)。对于此示例，使用平均精确率指标来评估性能。平均准确率提供单一数字，该数字综合反映了检测器进行正确分类的能力（精确率）和检测器找到所有相关对象的能力（召回率）。

将应用于训练数据的同一预处理变换应用于测试数据。

```
testData = transform(testData,@(data)preprocessData(data,inputSize));
```

对所有测试图像运行检测器。

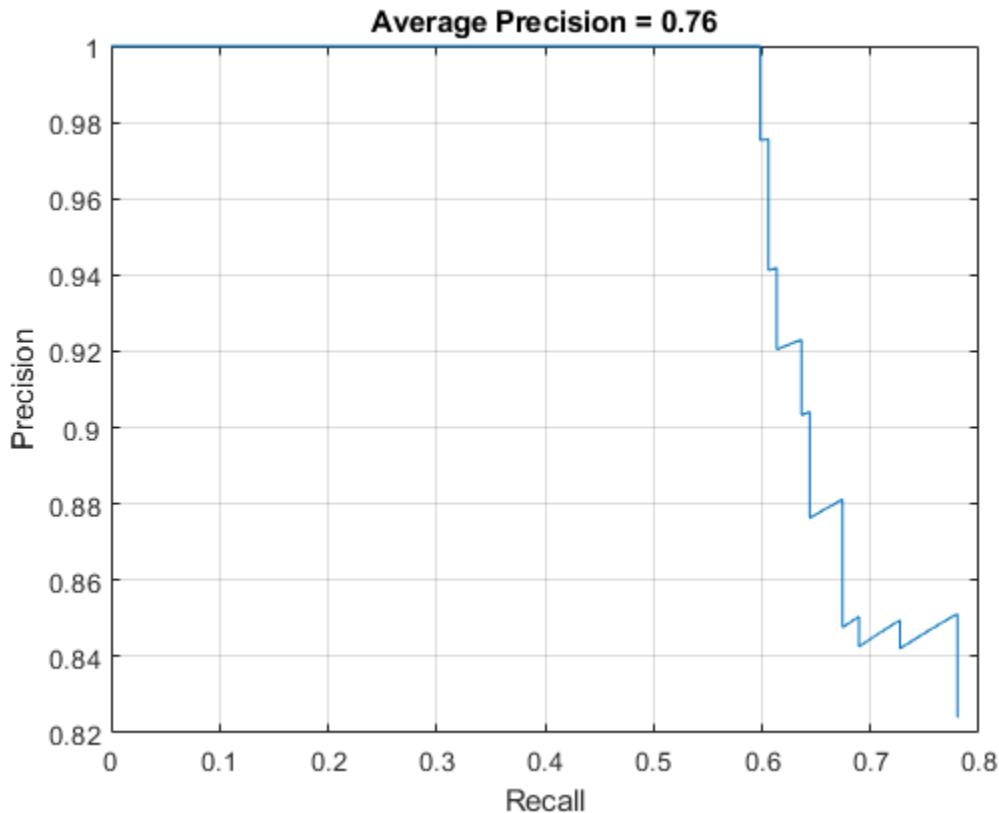
```
if doTrainingAndEval
    detectionResults = detect(detector,testData,'MinibatchSize',4);
else
    % Load pretrained detector for the example.
    pretrained = load('fasterRCNNResNet50EndToEndVehicleExample.mat');
    detectionResults = pretrained.detectionResults;
end
```

使用平均精确率指标评估目标检测器。

```
[ap, recall, precision] = evaluateDetectionPrecision(detectionResults,testData);
```

精确率/召回率 (PR) 曲线强调检测器在不同召回水平下的精确程度。理想情况下，所有召回水平的精确率均为 1。使用更多数据有助于提高平均精确率，但可能需要更多训练时间。绘制 PR 曲线。

```
figure
plot(recall,precision)
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f', ap))
```



## 支持函数

```
function data = augmentData(data)
% Randomly flip images and bounding boxes horizontally.
```

```

tform = randomAffine2d('XReflection',true);
sz = size(data{1},[1 2]);
rout = affineOutputView(sz, tform);
data{1} = imwarp(data{1},tform,'OutputView',rout);

% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2}, sz);

% Warp boxes.
data{2} = bboxwarp(data{2},tform,rout);
end

function data = preprocessData(data,targetSize)
% Resize image and bounding boxes to targetSize.
sz = size(data{1},[1 2]);
scale = targetSize(1:2)./sz;
data{1} = imresize(data{1},targetSize(1:2));

% Sanitize box data, if needed.
data{2} = helperSanitizeBoxes(data{2}, sz);

% Resize boxes.
data{2} = bboxresize(data{2},scale);
end

```

## 参考资料

- [1] Ren, S., K. He, R. Girshick, and J. Sun."Faster R-CNN:Towards Real-Time Object Detection with Region Proposal Networks."IEEE Transactions of Pattern Analysis and Machine Intelligence.Vol. 39, Issue 6, June 2017, pp. 1137-1149.
- [2] Girshick, R., J. Donahue, T. Darrell, and J. Malik."Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation."Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition.Columbus, OH, June 2014, pp. 580-587.
- [3] Girshick, R."Fast R-CNN."Proceedings of the 2015 IEEE International Conference on Computer Vision.Santiago, Chile, Dec. 2015, pp. 1440-1448.
- [4] Zitnick, C. L., and P. Dollar."Edge Boxes:Locating Object Proposals from Edges."European Conference on Computer Vision.Zurich, Switzerland, Sept. 2014, pp. 391-405.
- [5] Uijlings, J. R. R., K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders."Selective Search for Object Recognition."International Journal of Computer Vision.Vol. 104, Number 2, Sept. 2013, pp. 154-171.

## 另请参阅

### 函数

**trainRCNNOBJECTDETECTOR | trainFastRCNNOBJECTDETECTOR |  
trainFasterRCNNOBJECTDETECTOR**

## 详细信息

- "Getting Started with R-CNN, Fast R-CNN, and Faster R-CNN" (Computer Vision Toolbox)

- “使用 Faster R-CNN 深度学习进行目标检测” (第 8-78 页)
- “使用 R-CNN 深度学习训练目标检测器” (第 8-66 页)

## 使用单目相机和语义分割创建占据栅格

此示例说明如何使用语义分割和深度学习来估计一辆车周围的可用空间和创建占据栅格。然后，您将使用该占据栅格创建车辆代价地图，该代价地图可用于规划路径。

### 关于可用空间估计

自由空间估计用于识别自主车辆周围可以行驶而不会碰到任何障碍物的区域，这些障碍物包括行人、路沿和其他车辆等。车辆可以使用各种传感器来估计可用空间，如雷达、激光雷达或相机。此示例主要演示使用语义分割基于图像传感器估计可用空间。

在本示例中，您将学习如何：

- 使用语义图像分割来估计可用空间。
- 基于可用空间估计创建占据栅格。
- 在鸟瞰图上可视化占据栅格。
- 基于占据栅格创建车辆代价地图。
- 检查世界坐标系中的位置被占据还是自由的。

### 下载预训练网络

此示例使用预训练的语义分割网络，该网络可以将像素分为 11 个不同类，包括 **Road**、**Pedestrian**、**Car** 和 **Sky**。通过将分类为 **Road** 的图像像素定义为可用空间，可以估计图像中的可用空间。所有其他类都定义为非可用空间或障碍物。

训练该网络的完整过程如“Semantic Segmentation Using Deep Learning”(Computer Vision Toolbox)示例中所示。下载预训练的网络。

```
% Download the pretrained network.
pretrainedURL = 'https://www.mathworks.com/supportfiles/vision/data/segnetVGG16CamVid.mat';
pretrainedFolder = fullfile(tempdir,'pretrainedSegNet');
pretrainedSegNet = fullfile(pretrainedFolder,'segnetVGG16CamVid.mat');
if ~exist(pretrainedFolder,'dir')
    mkdir(pretrainedFolder);
    disp('Downloading pretrained SegNet (107 MB)...');
    websave(pretrainedSegNet,pretrainedURL);
    disp('Download complete.');
end

% % Load the network.
data = load(pretrainedSegNet);
net = data.net;
```

注意：数据的下载时间取决于您的 Internet 连接。下载完成之前，上面使用的命令会阻止 MATLAB。您也可以使用 Web 浏览器先将数据集下载到本地磁盘。这种情况下，要使用从 Web 下载的文件，请将上面代码中的 **pretrainedFolder** 变量更改为下载文件所在的位置。

### 估计可用空间

通过使用下载的语义分割网络处理图像来估计可用空间。网络返回图像中每个图像像素的分类。分类为 **Road** 的图像像素会被识别为可用空间。

本示例中使用的图像是 CamVid 数据集 [1] 中一个图像序列的单帧。本示例中所示的过程可应用于一系列帧，以估计车辆行驶时的可用空间。然而，由于本示例中使用了超深度卷积神经网络架构（具有 VGG-16 编码器的 SegNet），因此处理每帧大约需要 1 秒。因此，为了方便起见，此处仅处理单个帧。

```
% Read the image.
I = imread('seq05vd_snap_shot.jpg');

% Segment the image.
[C,scores,allScores] = semanticseg(I,net);

% Overlay free space onto the image.
B = labeloverlay(I,C,'IncludedLabels','Road');

% Display free space and image.
figure
imshow(B)
```

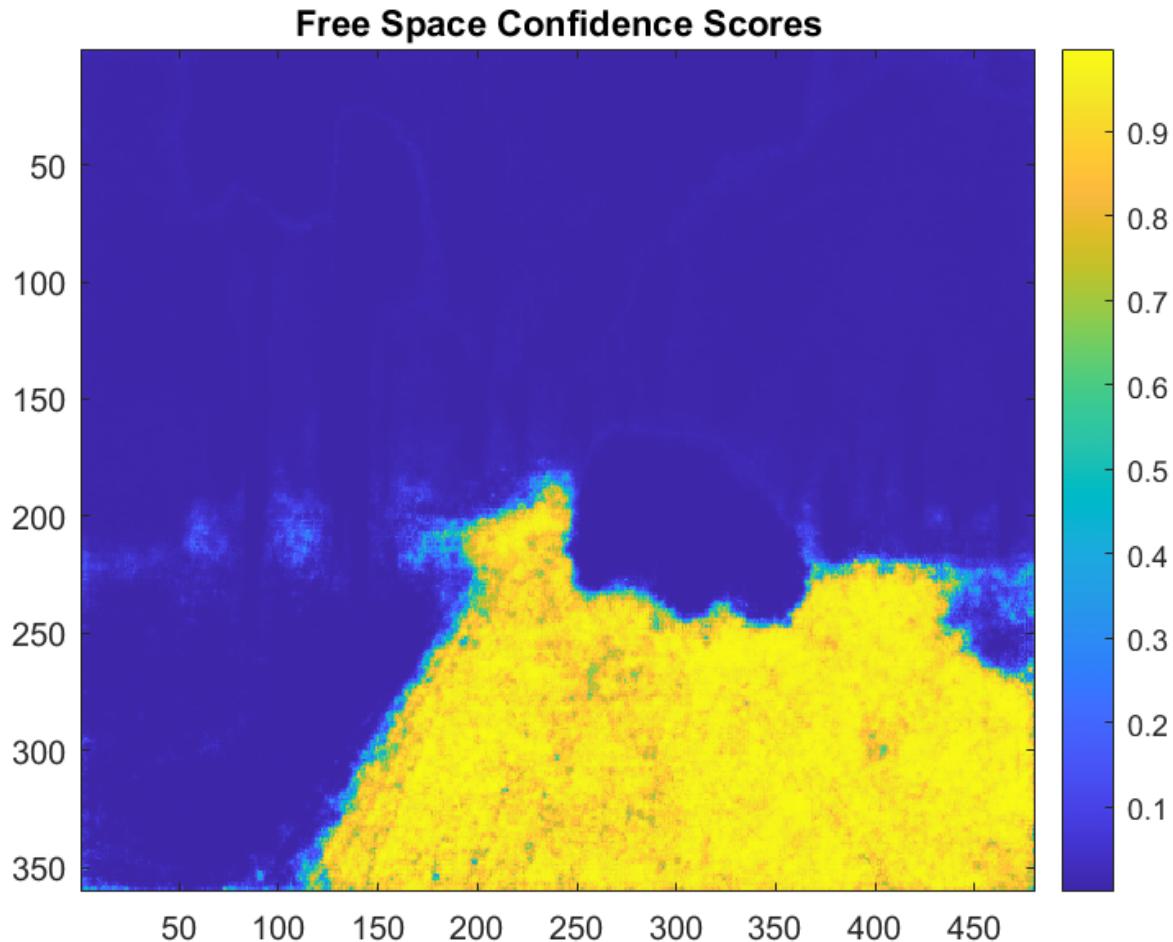


要了解可用空间估计的置信度，请显示每个像素的 Road 类的输出分数。置信度值可用于向下游算法告知估计的有效性。例如，即使网络将某像素分类为 Road，出于安全原因，置信度分数也可能低到足以忽略该分类。

```
% Use the network's output score for Road as the free space confidence.
roadClassIdx = 4;
freeSpaceConfidence = allScores(:, :, roadClassIdx);

% Display the free space confidence.
figure
```

```
imagesc(freeSpaceConfidence)
title('Free Space Confidence Scores')
colorbar
```



虽然 Road 像素的初始分割结果显示道路上的大多数像素被正确分类，但对分数进行可视化后，可以了解分类器关于这些分类的置信度的更多细节。例如，当您靠近汽车的边界时，置信度会下降。

### 创建鸟瞰图图像

自由空间估计是在图像空间中生成的。为了便于生成适用于导航的占据栅格地图，需要将自由空间估计变换为车辆坐标系。这可以通过将自由空间估计变换为鸟瞰图来实现。

要创建鸟瞰图图像，首先定义相机传感器配置。本示例末尾列出的支持函数 `camvidMonoCameraSensor` 可返回 `monoCamera` (Automated Driving Toolbox) 对象，该对象表示用于收集 CamVid[1] 数据的单目相机。配置 `monoCamera` (Automated Driving Toolbox) 时需要相机的内参和外参，这些参数是使用 CamVid 数据集中提供的数据进行估计的。为了估计相机的内参，该函数使用了 CamVid 棋盘校准图像和 Camera Calibrator (Computer Vision Toolbox) App。相机外参估计（例如高度和俯仰）来自于 CamVid 数据集的作者估计的外部参数数据。

```
% Create monoCamera for CamVid data.
sensor = camvidMonoCameraSensor();
```

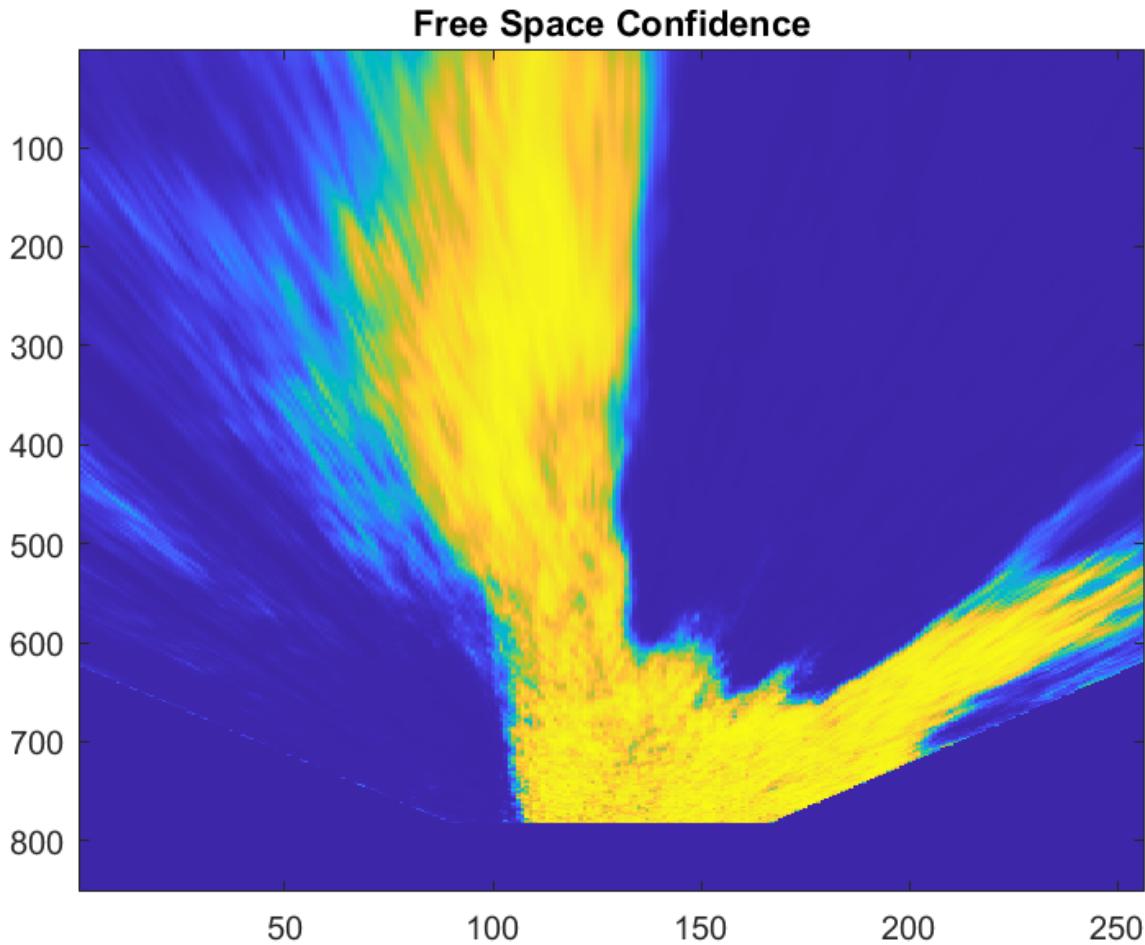
基于给定的相机设置，`birdsEyeView` (Automated Driving Toolbox) 对象将原始图像变换为鸟瞰图。此对象允许您指定要使用车辆坐标系变换的区域。请注意，车辆坐标单位是由 `monoCamera` (Automated Driving Toolbox) 对象根据相机台座高度单位确定的，本例中以米为单位。如果高度以毫米为单位指定，则仿真的其他部分也将使用毫米为单位。

```
% Define bird's-eye-view transformation parameters.  
distAheadOfSensor = 20; % in meters, as previously specified in monoCamera height input  
spaceToOneSide = 3; % look 3 meters to the right and left  
bottomOffset = 0;  
outView = [bottomOffset, distAheadOfSensor, -spaceToOneSide, spaceToOneSide];  
  
outImageSize = [NaN, 256]; % output image width in pixels; height is chosen automatically to preserve units per  
birdsEyeConfig = birdsEyeView(sensor,outView,outImageSize);  
  
为图像和自由空间置信度生成鸟瞰图。  
  
% Resize image and free space estimate to size of CamVid sensor.  
imageSize = sensor.Intrinsics.ImageSize;  
I = imresize(I,imageSize);  
freeSpaceConfidence = imresize(freeSpaceConfidence,imageSize);  
  
% Transform image and free space confidence scores into bird's-eye view.  
imageBEV = transformImage(birdsEyeConfig,I);  
freeSpaceBEV = transformImage(birdsEyeConfig,freeSpaceConfidence);  
  
% Display image frame in bird's-eye view.  
figure  
imshow(imageBEV)
```



将图像变换为鸟瞰图，并生成自由空间置信度。

```
figure
imagesc(freeSpaceBEV)
title('Free Space Confidence')
```



远离传感器的区域由于像素更少而更模糊，因此需要更多插值。

### 基于自由空间估计创建占据栅格图

占据栅格用于在车辆坐标系中将车辆的周围环境表示为离散网格，并用于路径规划。占据栅格中的每个单元都有一个代表该单元格占据概率的值。估计的自由空间可用于填充占据栅格的值。

使用自由空间估计值填充占据栅格的过程如下：

- 1 以车辆坐标系定义占据栅格的大小。
- 2 为每个栅格单元生成一组  $(X, Y)$  点。这些点使用车辆坐标系。
- 3 使用 `vehicleToImage` (Automated Driving Toolbox) 变换将点从车辆坐标空间  $(X, Y)$  变换为鸟瞰图图像坐标空间  $(x, y)$ 。

- 4 使用 `griddedInterpolant` 抽取  $(x,y)$  位置处的自由空间置信度值，通过插值法来插补不完全位于图像像素中心的自由空间置信度值。
- 5 使用该栅格单元中所有  $(x,y)$  点的平均自由空间置信度值来填充占据栅格单元。

为简洁起见，上面所述过程使用支持函数 `createOccupancyGridFromFreeSpaceEstimate` 实现，该函数列在此示例的末尾。根据鸟瞰图配置定义占据栅格的大小，并通过调用 `createOccupancyGridFromFreeSpaceEstimate` 创建占据栅格。

```
% Define dimensions and resolution of the occupancy grid.
gridX = distAheadOfSensor;
gridY = 2 * spaceToOneSide;
cellSize = 0.25; % in meters to match units used by CamVid sensor

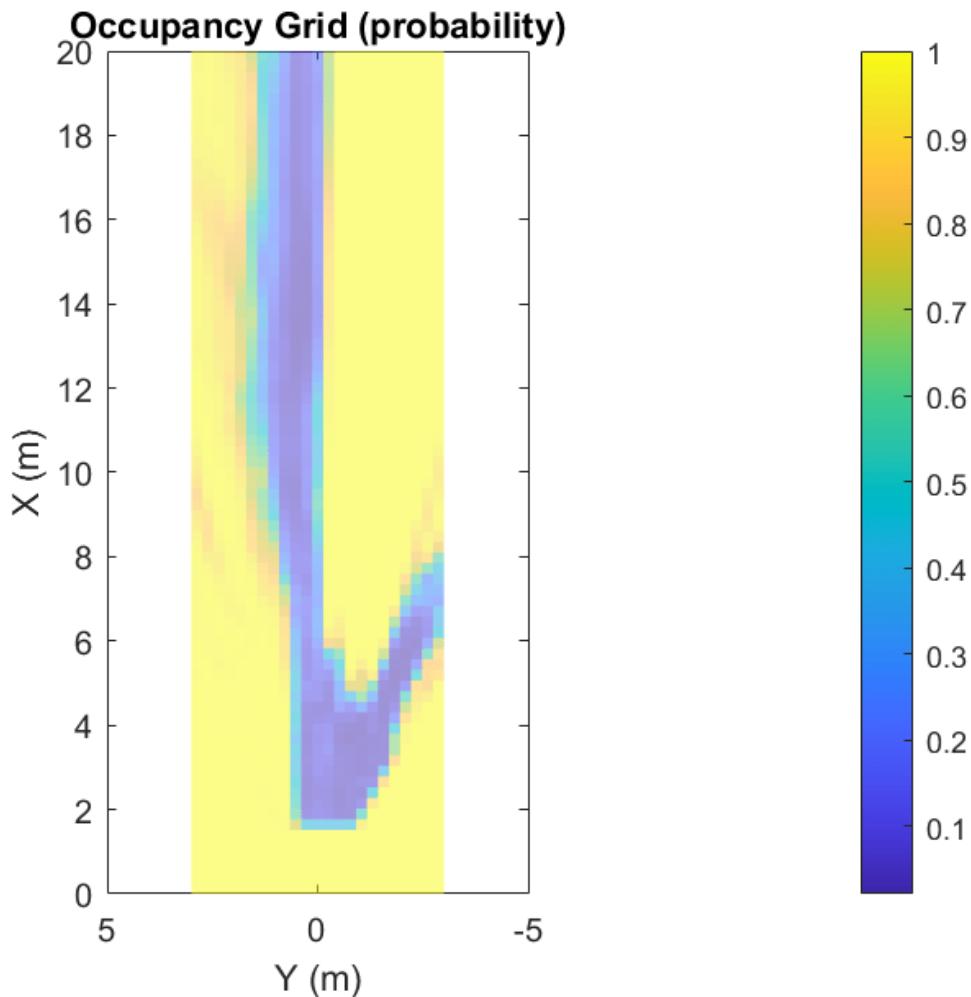
% Create the occupancy grid from the free space estimate.
occupancyGrid = createOccupancyGridFromFreeSpaceEstimate(...%
    freeSpaceBEV, birdsEyeConfig, gridX, gridY, cellSize);
```

使用 `birdsEyePlot` (Automated Driving Toolbox) 可可视化占据栅格。创建一个 `birdsEyePlot` (Automated Driving Toolbox)，并使用 `pcolor` 在最上层添加占据栅格。

```
% Create bird's-eye plot.
bep = birdsEyePlot('XLimits',[0 distAheadOfSensor], 'YLimits', [-5 5]);

% Add occupancy grid to bird's-eye plot.
hold on
[numCellsY,numCellsX] = size(occupancyGrid);
X = linspace(0, gridX, numCellsX);
Y = linspace(-gridY/2, gridY/2, numCellsY);
h = pcolor(X,Y,occupancyGrid);
title('Occupancy Grid (probability)')
colorbar
delete(legend)

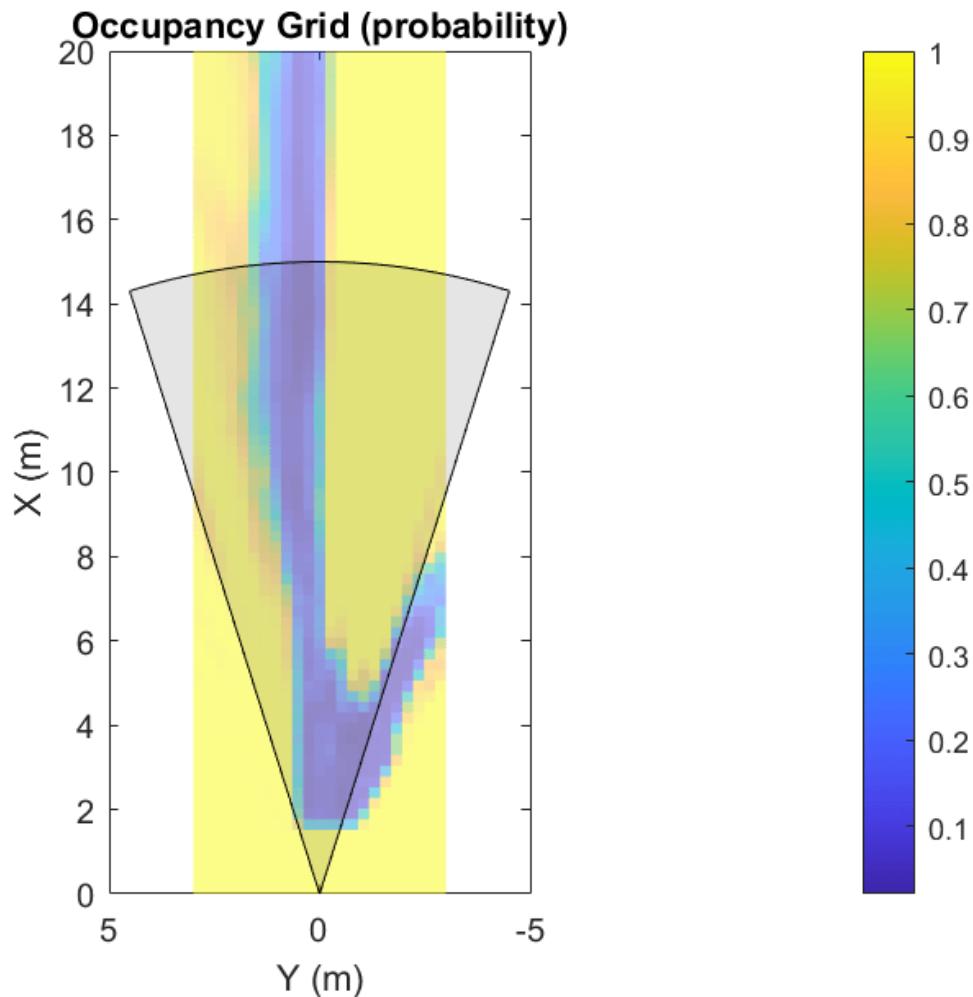
% Make the occupancy grid visualization transparent and remove grid lines.
h.FaceAlpha = 0.5;
h.LineStyle = 'none';
```



鸟瞰图还可以显示来自多个传感器的数据。例如，使用 `coverageAreaPlotter` (Automated Driving Toolbox) 添加雷达覆盖区域。

```
% Add coverage area to plot.
caPlotter = coverageAreaPlotter(bep, 'DisplayName', 'Coverage Area');

% Update it with a field of view of 35 degrees and a range of 60 meters
mountPosition = [0 0];
range = 15;
orientation = 0;
fieldOfView = 35;
plotCoverageArea(caPlotter, mountPosition, range, orientation, fieldOfView);
hold off
```



显示来自多个传感器的数据对于诊断和调试自动驾驶车辆作出的决策非常有用。

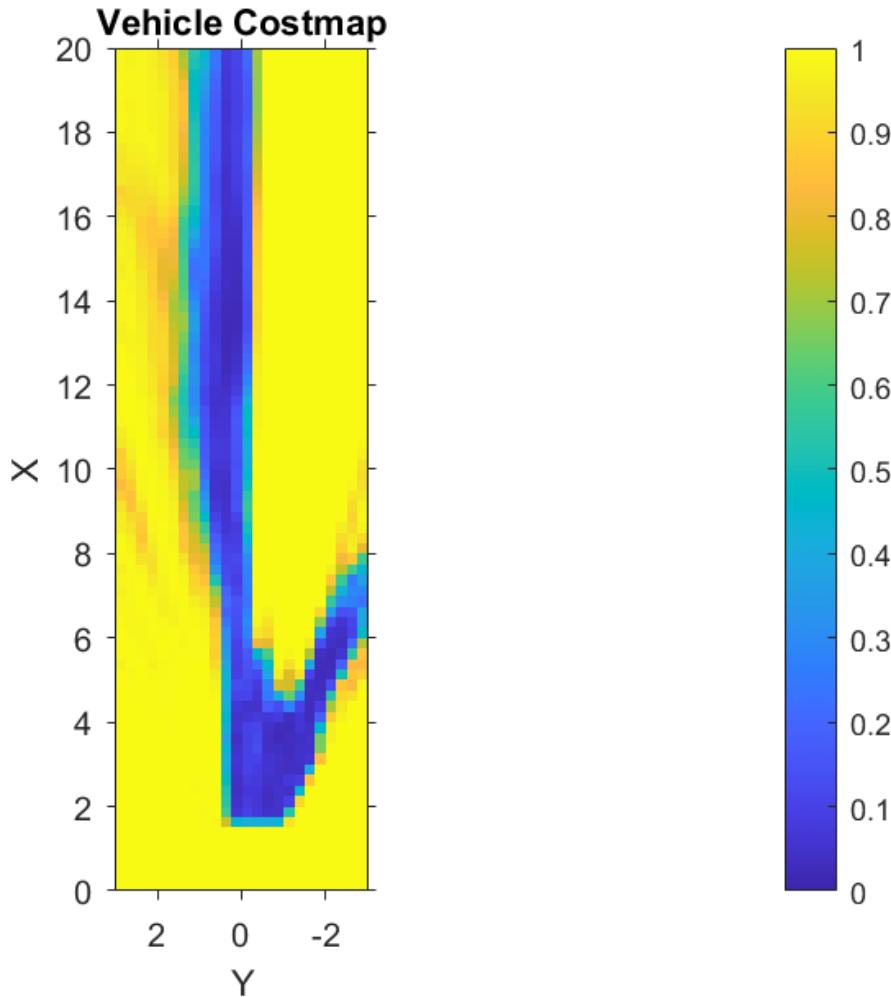
### 使用占据栅格创建车辆代价地图

`vehicleCostmap` (Automated Driving Toolbox) 能够检查车辆或世界坐标系中的某位置是占据的还是自由的。这种检查对于任何路径规划或决策算法均为必需的。使用生成的 `occupancyGrid` 创建 `vehicleCostmap` (Automated Driving Toolbox)。

```
% Create the costmap.
costmap = vehicleCostmap(flipud(occupancyGrid), ...
    'CellSize',cellSize, ...
    'MapLocation',[0,-spaceToOneSide]);
costmap.CollisionChecker.InflationRadius = 0;

% Display the costmap.
figure
plot(costmap,'Inflation','off')
colormap(parula)
colorbar
title('Vehicle Costmap')
```

```
% Orient the costmap so that it lines up with the vehicle coordinate
% system, where the X-axis points in front of the ego vehicle and the
% Y-axis points to the left.
view(gca,-90,90)
```



为了说明如何使用 `vehicleCostmap` (Automated Driving Toolbox), 使用世界坐标系创建一组位置。这些位置表示车辆可以通过的路径。

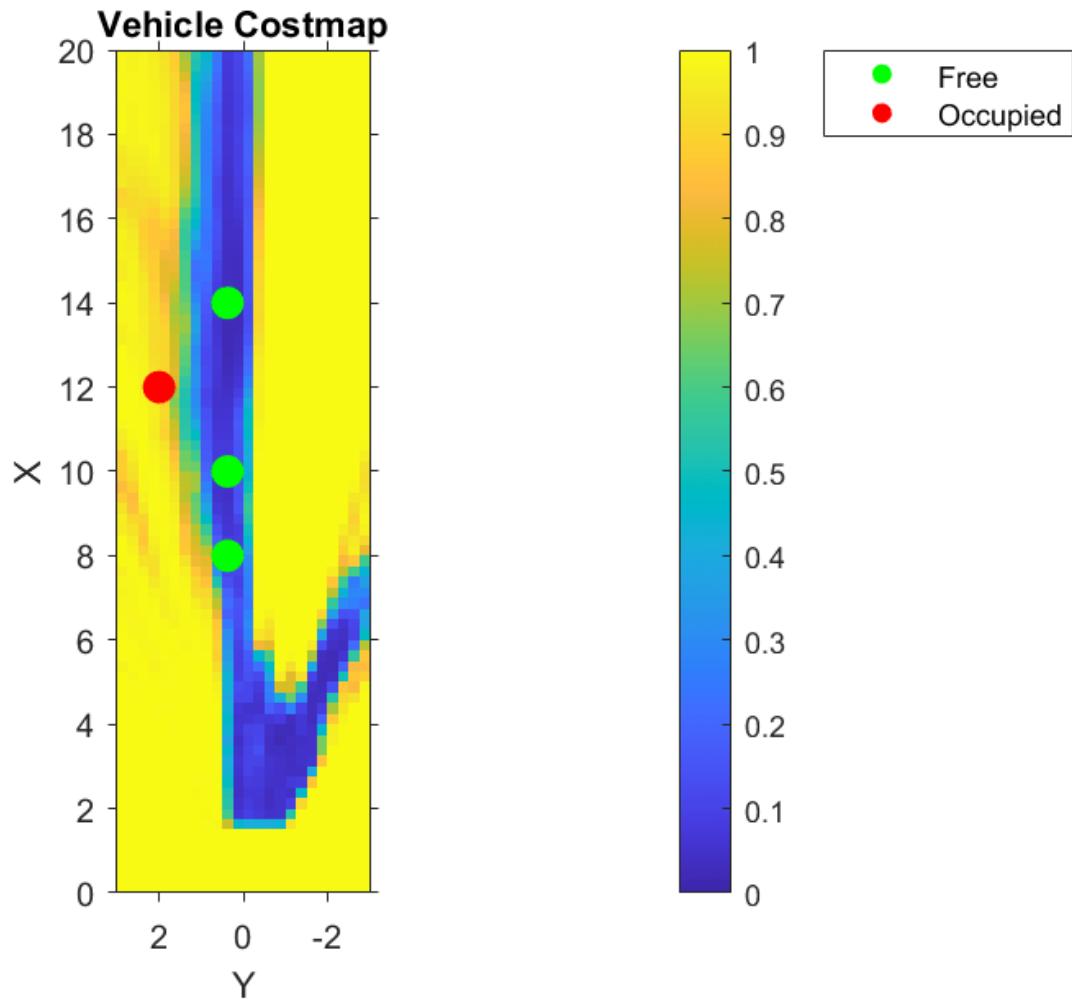
```
% Create a set of locations in vehicle coordinates.
candidateLocations = [
    8 0.375
    10 0.375
    12 2
    14 0.375
];
```

使用 `checkOccupied` (Automated Driving Toolbox) 检查每个位置是占据的还是自由的。根据结果, 可能无法采用某条潜在的路径, 因为它与 `costmap` 中定义的障碍物相冲突。

```
% Check if locations are occupied.
isOccupied = checkOccupied(costmap,candidateLocations);
```

```
% Partition locations into free and occupied for visualization purposes.
occupiedLocations = candidateLocations(isOccupied,:);
freeLocations = candidateLocations(~isOccupied,:);

% Display free and occupied points on top of costmap.
hold on
markerSize = 100;
scatter(freeLocations(:,1),freeLocations(:,2),markerSize,'g','filled')
scatter(occupiedLocations(:,1),occupiedLocations(:,2),markerSize,'r','filled');
legend(["Free" "Occupied"])
hold off
```



上面使用的 `occupancyGrid`、`vehicleCostmap` (Automated Driving Toolbox) 和 `checkOccupied` (Automated Driving Toolbox) 说明了 `pathPlannerRRT` (Automated Driving Toolbox) 等路径规划程序使用的基本操作。您可以在“Automated Parking Valet” (Automated Driving Toolbox)示例中了解有关路径规划的详细信息。

## 参考资料

[1] Brostow, Gabriel J., Julien Fauqueur, and Roberto Cipolla."Semantic Object Classes in Video:A high-definition ground truth database."Pattern Recognition Letters.Vol. 30, Issue 2, 2009, pp. 88-97.

## 支持函数

```

function sensor = camvidMonoCameraSensor()
% Return a monoCamera camera configuration based on data from the CamVid
% data set[1].
%
% The cameraCalibrator app was used to calibrate the camera using the
% calibration images provided in CamVid:
%
% http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/CalibrationSeq_and_Files_0010YU.zip
%
% Calibration pattern grid size is 28 mm.
%
% Camera pitch is computed from camera pose matrices [R t] stored here:
%
% http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/EgoBoost_trax_matFiles.zip

% References
% -----
% [1] Brostow, Gabriel J., Julien Fauqueur, and Roberto Cipolla. "Semantic Object
% Classes in Video: A high-definition ground truth database." _Pattern Recognition
% Letters_. Vol. 30, Issue 2, 2009, pp. 88-97.

calibrationData = load('camera_params_camvid.mat');

% Describe camera configuration.
focalLength = calibrationData.cameraParams.FocalLength;
principalPoint = calibrationData.cameraParams.PrincipalPoint;
imageSize = calibrationData.cameraParams.ImageSize;

% Camera height estimated based on camera setup pictured in [1].
height = 0.5; % height in meters from the ground

% Camera pitch was computed using camera extrinsics provided in data set.
pitch = 0; % pitch of the camera, towards the ground, in degrees

camIntrinsics = cameraIntrinsics(focalLength,principalPoint,imageSize);
sensor = monoCamera(camIntrinsics,height,'Pitch',pitch);
end

function occupancyGrid = createOccupancyGridFromFreeSpaceEstimate(...  

    freeSpaceBEV,birdsEyeConfig,gridX,gridY,cellSize)
% Return an occupancy grid that contains the occupancy probability over
% a uniform 2-D grid.

% Number of cells in occupancy grid.
numCellsX = ceil(gridX / cellSize);
numCellsY = ceil(gridY / cellSize);

% Generate a set of (X,Y) points for each grid cell. These points are in
% the vehicle's coordinate system. Start by defining the edges of each grid
% cell.

```

```
% Define the edges of each grid cell in vehicle coordinates.
XEdges = linspace(0,gridX,numCellsX);
YEdges = linspace(-gridY/2,gridY/2,numCellsY);

% Next, specify the number of sample points to generate along each
% dimension within a grid cell. Use these to compute the step size in the
% X and Y direction. The step size will be used to shift the edge values of
% each grid to produce points that cover the entire area of a grid cell at
% the desired resolution.

% Sample 20 points from each grid cell. Sampling more points may produce
% smoother estimates at the cost of additional computation.
numSamplePoints = 20;

% Step size needed to sample number of desired points.
XStep = (XEdges(2)-XEdges(1)) / (numSamplePoints-1);
YStep = (YEdges(2)-YEdges(1)) / (numSamplePoints-1);

% Finally, slide the set of points across both dimensions of the grid
% cells. Sample the occupancy probability along the way using
% griddedInterpolant.

% Create griddedInterpolant for sampling occupancy probability. Use 1
% minus the free space confidence to represent the probability of occupancy.
occupancyProb = 1 - freeSpaceBEV;
sz = size(occupancyProb);
[y,x] = ndgrid(1:sz(1),1:sz(2));
F = griddedInterpolant(y,x,occupancyProb);

% Initialize the occupancy grid to zero.
occupancyGrid = zeros(numCellsY*numCellsX,1);

% Slide the set of points XEdges and YEdges across both dimensions of the
% grid cell.
for j = 1:numSamplePoints

    % Increment sample points in the X-direction
    X = XEdges + (j-1)*XStep;

    for i = 1:numSamplePoints

        % Increment sample points in the Y-direction
        Y = YEdges + (i-1)*YStep;

        % Generate a grid of sample points in bird's-eye-view vehicle coordinates
        [XGrid,YGrid] = meshgrid(X,Y);

        % Transform grid of sample points to image coordinates
        xy = vehicleToImage(birdsEyeConfig,[XGrid(:) YGrid(:)]);

        % Clip sample points to lie within image boundaries
        xy = max(xy,1);
        xq = min(xy(:,1),sz(2));
        yq = min(xy(:,2),sz(1));

        % Sample occupancy probabilities using griddedInterpolant and keep
        % a running sum.
```

```
occupancyGrid = occupancyGrid + F(yq,xq);
end

% Determine mean occupancy probability.
occupancyGrid = occupancyGrid / numSamplePoints^2;
occupancyGrid = reshape(occupancyGrid,numCellsY,numCellsX);
end
```



# 激光雷达示例

---



# 信号处理示例

---

## 使用深度学习进行波形分割

此示例说明如何使用递归深度学习网络和时频分析来分割人体心电图 (ECG) 信号。

### 简介

人类的心电图活动可以作为远离基线信号的幅值序列来测量。对于单个正常心跳周期，ECG 信号可分为以下几种心跳形态 [1 (第 12-0 页)]：

- P 波 - 表示心房去极化的 QRS 复波前的小偏移
- QRS 复波 - 心跳的最大幅值部分
- T 波 - 表示心室复极化的 QRS 复波后的小偏移

ECG 波形的这些区域的分割可作为基础测量数据用于评估人类心脏整体健康和异常状况 [2 (第 12-0 页)]。手动注释 ECG 信号的每个区域可能是一项乏味且耗时的任务。信号处理和深度学习方法有助于简化注释并自动对关注区域进行注释。

此示例使用来自公开可用的 QT 数据库的 ECG 信号 [3 (第 12-0 页)] [4 (第 12-0 页)]。这些数据由总共 105 名患者大约 15 分钟的 ECG 录音片段组成，采样率为 250 Hz。为了获得每个录音片段，检查人员将两个电极放置在患者胸部的不同位置，以产生双通道信号。该数据库提供由自动专家系统生成的信号区域标签 [2 (第 12-0 页)]。此示例旨在使用深度学习解决方案根据样本所在的区域为每个 ECG 信号样本提供标签。这种对信号中的关注区域加标签的过程通常称为波形分割。

为了训练深度神经网络来对信号区域分类，可以使用长短期记忆 (LSTM) 网络。此示例说明如何使用信号预处理方法和时频分析来提高 LSTM 分割性能。具体而言，该示例使用傅里叶同步压缩变换来表示 ECG 信号的非平稳行为。

### 下载并准备数据

105 个双通道 ECG 信号的每个通道由自动专家系统独立标注并独立处理，总共 210 个 ECG 信号，它们与区域标签一起存储在 210 个 MAT 文件中。这些文件可在以下位置获得：<https://www.mathworks.com/supportfiles/SPT/data/QTDatabaseECGData.zip>。

将数据文件下载到您的临时目录中，临时目录的位置由 MATLAB® 的 `tempdir` 命令指定。如果要将数据文件放在不同于 `tempdir` 的文件夹中，请在后续说明中更改目录名称。

```
% Download the data
dataURL = 'https://www.mathworks.com/supportfiles/SPT/data/QTDatabaseECGData1.zip';
datasetFolder = fullfile(tempdir,'QTDatabase');
zipFile = fullfile(tempdir,'QTDatabaseECGData.zip');
if ~exist(datasetFolder,'dir')
    websave(zipFile,dataURL);
    unzip(zipFile,tempdir);
end
```

`unzip` 操作会在您的临时目录中创建 `QTDatabaseECGData` 文件夹，其中包含 210 个 MAT 文件。每个文件在变量 `ecgSignal` 中包含一个 ECG 信号，在变量 `signalRegionLabels` 中包含一个区域标签表。每个文件还在变量 `Fs` 中包含信号的采样率。在此示例中，所有信号的采样率均为 250 Hz。

创建一个信号数据存储来访问文件中的数据。此示例假设数据集已存储在临时目录中的 `QTDatabaseECGData` 文件夹下。如果不是这样，请更改下面代码中数据的路径。使用 `SignalVariableNames` 参数指定要从每个文件中读取的信号变量名称。

```
sds = signalDatastore(datasetFolder,'SignalVariableNames',['ecgSignal','signalRegionLabels'])
```

```
sds =
  signalDatastore with properties:

    Files:{%
      '/tmp/QTDataset/ecg1.mat';
      '/tmp/QTDataset/ecg10.mat';
      '/tmp/QTDataset/ecg100.mat'
      ... and 207 more
    }
    AlternateFileSystemRoots: [0×0 string]
    ReadSize: 1
    SignalVariableNames: ["ecgSignal" "signalRegionLabels"]
```

每次调用 **read** 函数时，数据存储都会返回一个包含 ECG 信号和区域标签表的二元素元胞数组。使用数据存储的 **preview** 函数，可以看到第一个文件的内容是长度为 225000 个采样的 ECG 信号和一个包含 3385 个区域标签的表。

```
data = preview(sds)
```

```
data=2×1 cell array
{225000×1 double}
{ 3385×2 table }
```

查看区域标签表的前几行，注意每行都包含区域范围索引和区域类值（P、T 或 QRS）。

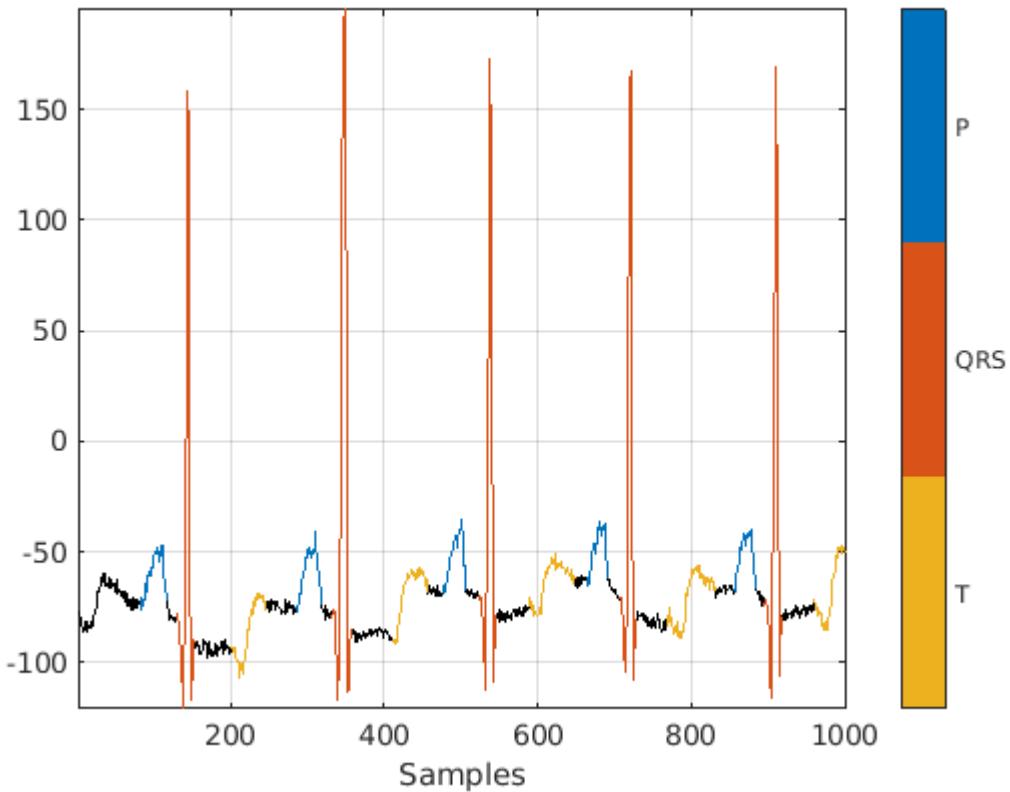
```
head(data{2})
```

```
ans=8×2 table
  ROILimits    Value
```

83	117	P
130	153	QRS
201	246	T
285	319	P
332	357	QRS
412	457	T
477	507	P
524	547	QRS

使用 **signalMask** 对象可视化前 1000 个采样的标签。

```
M = signalMask(data{2});
plotsigroi(M,data{1}(1:1000))
```



通常的机器学习分类过程如下：

- 1 将数据库分成训练数据集和测试数据集。
- 2 使用训练数据集训练网络。
- 3 使用经过训练的网络对测试数据集进行预测。

用 70% 的数据对网络进行训练，用其余的 30% 对网络进行测试。

为了获得可重现的结果，请重置随机数生成器。使用 **dividerand** 函数获得随机索引来对文件进行乱序处理，使用 **signalDatastore** 的 **subset** 函数将数据分成训练数据存储和测试数据存储。

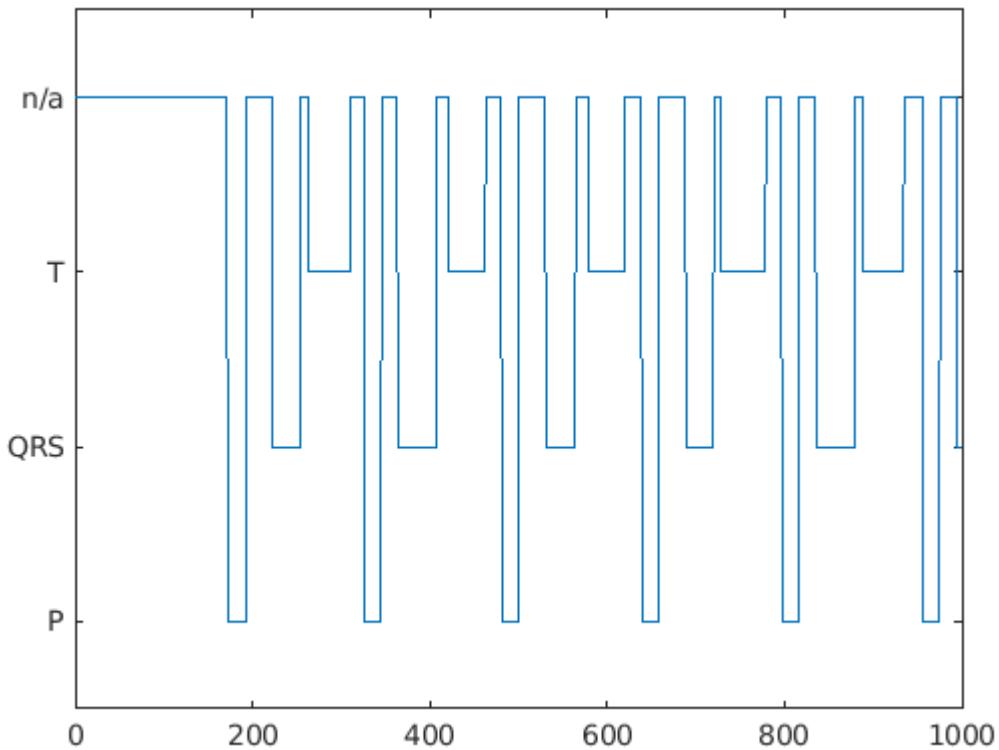
```
rng default
[trainIdx,~,testIdx] = dividerand(numel(sds.Files),0.7,0.3);
trainDs = subset(sds,trainIdx);
testDs = subset(sds,testIdx);
```

在此分割问题中，LSTM 网络的输入是 ECG 信号，输出是与输入信号长度相同的标签序列或标签掩膜。网络任务将用信号采样点所属区域的名称来标注每个信号采样点。因此，有必要将数据集中的区域标签变换为包含针对每个信号采样点的各个标签的序列。使用变换后的数据存储和 **getmask** 辅助函数来变换区域标签。**getmask** 函数添加一个标签类别 "n/a"，用于标注不属于任何关注区域的样本。

```
type getmask.m

function outputCell = getmask(inputCell)
%GETMASK Convert region labels to a mask of labels of size equal to the
%size of the input ECG signal.
```

```
%  
% inputCell is a two-element cell array containing an ECG signal vector  
% and a table of region labels.  
%  
% outputCell is a two-element cell array containing the ECG signal vector  
% and a categorical label vector mask of the same length as the signal.  
  
% Copyright 2020 The MathWorks, Inc.  
  
sig = inputCell{1};  
roiTable = inputCell{2};  
L = length(sig);  
M = signalMask(roiTable);  
  
% Get categorical mask and give priority to QRS regions when there is overlap  
mask = catmask(M,L,'OverlapAction','prioritizeByList','PriorityList',[2 1 3]);  
  
% Set missing values to "n/a"  
mask(ismissing(mask)) = "n/a";  
  
outputCell = {sig,mask};  
end  
  
预览变换后的数据存储，观察它是否返回长度相等的一个信号向量和一个标签向量。绘制分类封装向量的前 1000 个元素。  
  
trainDs = transform(trainDs, @getmask);  
testDs = transform(testDs, @getmask);  
  
transformedData = preview(trainDs)  
  
transformedData=1×2 cell array  
 {224993×1 double} {224993×1 categorical}  
  
plot(transformedData{2}(1:1000))
```



将非常长的输入信号传入 LSTM 网络会导致估计性能下降和内存使用量过多。为了避免这些影响，请使用变换后的数据存储和 `resizeData` 辅助函数来拆分 ECG 信号及其对应的标签掩膜。辅助函数创建尽可能多的包含 5000 个采样的信号段，并丢弃其余采样。变换后的数据存储的输出预览显示，第一个 ECG 信号及其标签掩膜分成了若干包含 5000 个采样的信号段。请注意，变换后的数据存储的预览仅显示 8 个元素，它们是在我们调用数据存储 `read` 函数时会生成的包含  $\text{floor}(224993/5000) = 44$  个元素的元胞数组的前 8 个元素。

```
trainDs = transform(trainDs,@resizeData);
testDs = transform(testDs,@resizeData);
preview(trainDs)

ans=8×2 cell array
{1×5000 double} {1×5000 categorical}
```

### 选择训练网络或下载预训练网络

此示例的下列各节比较三种不同的 LSTM 网络训练方法。由于数据集很大，每个网络的训练过程可能需要几分钟。如果您的机器同时有 GPU 和 Parallel Computing Toolbox™，则 MATLAB 会自动使用 GPU 以加快训练速度。否则将使用 CPU。

您可以跳过训练步骤，使用以下选择器下载预训练网络。如果您要像示例中那样训练网络，请选择“Train Networks”。如果您要跳过训练步骤，请选择“Download Networks”，一个文件（其中包含所有三个预训练网络 - **rawNet**、**filteredNet** 和 **fsstNet**-）将下载到您的临时目录中，其位置由 MATLAB® 的 **tempdir** 命令指定。如果要将下载的文件放在不同于 **tempdir** 的文件夹中，请在后续说明中更改目录名称。

```
actionFlag = Train networks; % Train networks
if actionFlag == "Download networks"
    % Download the pre-trained networks
    dataURL = 'https://ssd.mathworks.com/supportfiles/SPT/data/QTDatabaseECGSegmentationNetworks.zip'; %#ok
    modelsFolder = fullfile(tempdir,'QTDatabaseECGSegmentationNetworks');
    modelsFile = fullfile(modelsFolder,'trainedNetworks.mat');
    zipFile = fullfile(tempdir,'QTDatabaseECGSegmentationNetworks.zip');
    if ~exist(modelsFolder,'dir')
        websave(zipFile,dataURL);
        unzip(zipFile,fullfile(tempdir,'QTDatabaseECGSegmentationNetworks'));
    end
    load(modelsFile)
end
```

下载的网络和新训练的网络之间的结果可能略有不同，因为网络是使用随机初始权重训练的。

### 将原始 ECG 信号直接输入 LSTM 网络中

首先，使用来自训练数据集的原始 ECG 信号训练 LSTM 网络。

在训练前定义网络架构。指定大小为 1 的 **sequenceInputLayer**，以接受一维时序。用 'sequence' 输出模式指定一个 LSTM 层，以便为信号中的每个采样提供分类。使用 200 个隐含节点以获得最佳性能。指定输出大小为 4 的 **fullyConnectedLayer**，对每个波形类指定一个层。添加一个 **softmaxLayer** 和一个 **classificationLayer** 以输出估计的标签。

```
layers = [ ...
    sequenceInputLayer(1)
    lstmLayer(200,'OutputMode','sequence')
    fullyConnectedLayer(4)
    softmaxLayer
    classificationLayer];
```

为训练过程选择选项，以确保获得良好的网络性能。有关每个参数的说明，请参阅 **trainingOptions** 文档。

```
options = trainingOptions('adam',...
    'MaxEpochs',10, ...
    'MiniBatchSize',50, ...
    'InitialLearnRate',0.01, ...
    'LearnRateDropPeriod',3, ...
    'LearnRateSchedule','piecewise', ...
    'GradientThreshold',1, ...
    'Plots','training-progress',...
    'shuffle','every-epoch',...
    'Verbose',0, ...
    'DispatchInBackground',true);
```

由于整个训练数据集可放入内存，因此，如果 Parallel Computing Toolbox™ 可用，可以使用数据存储的 **tall** 函数以并行方式变换数据，然后将其收集到工作区中。神经网络训练是迭代进行的。在每次迭代中，数据存储从文件中读取数据，变换数据，然后更新网络系数。如果数据可放入计算机的内存中，将数据导

入工作区可以加快训练速度，因为数据只需读取和变换一次。请注意，如果数据无法放入内存，您必须将数据存储传递给训练函数，并且在每轮训练中执行变换。

为训练集和测试集创建 tall 数组。根据您的系统，MATLAB 创建的并行池中的工作进程数量可能会有所不同。

```
tallTrainSet = tall(trainDs);
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 8).
```

```
tallTestSet = tall(testDs);
```

现在调用 tall 数组的 **gather** 函数来计算整个数据集上的变换，并获得具有训练和测试信号及标签的元胞数组。

```
trainData = gather(tallTrainSet);
```

```
Evaluating tall expression using the Parallel Pool 'local':
```

```
- Pass 1 of 1: Completed in 11 sec
```

```
Evaluation completed in 12 sec
```

```
trainData(1,:)
```

```
ans=1×2 cell array
    {1×5000 double}  {1×5000 categorical}
```

```
 testData = gather(tallTestSet);
```

```
Evaluating tall expression using the Parallel Pool 'local':
```

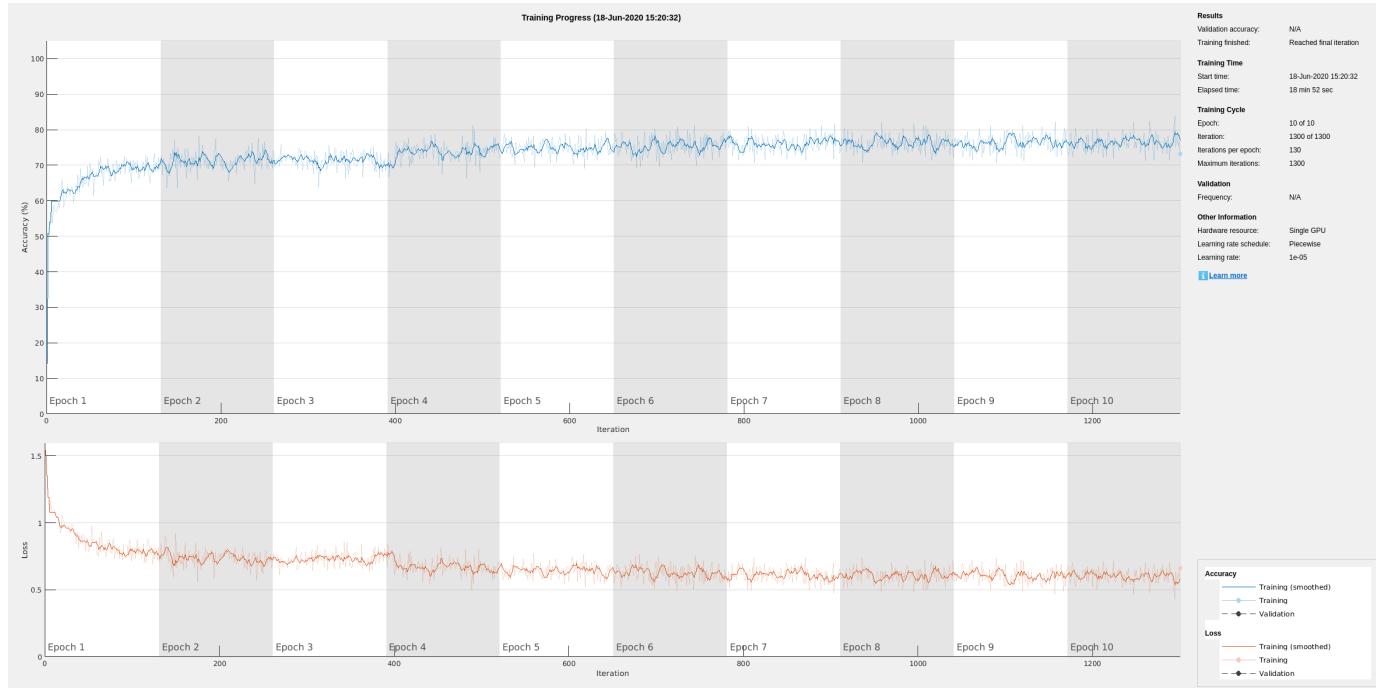
```
- Pass 1 of 1: Completed in 2.9 sec
```

```
Evaluation completed in 3.1 sec
```

### 训练网络

使用 **trainNetwork** 命令训练 LSTM 网络。

```
if actionFlag == "Train networks"
    rawNet = trainNetwork(trainData(:,1),trainData(:,2),layers,options);
end
```



图窗中的训练准确度和损失子图跟踪所有迭代的训练进度。使用原始信号数据，网络将大约 77% 的采样正确分类为属于 P 波、QRS 复波、T 波或不带标签的区域 "n/a"。

### 对测试数据分类

使用经过训练的 LSTM 网络和 `classify` 命令对测试数据进行分类。指定小批量大小为 50 以匹配训练选项。

```
predTest = classify(rawNet,testData(:,1),'MiniBatchSize',50);
```

混淆矩阵提供了一种直观的分类性能可视化方式。使用 `confusionchart` 命令计算用于测试数据预测的总体分类准确度。对于每个输入，将分类标签的元胞数组转换为行向量。指定列归一化显示，以每个类的采样百分比形式查看结果。

```
confusionchart([predTest{:}],[testData{:},2],'Normalization','column-normalized');
```

		P	QRS	T	n/a
		37.4%	2.3%	1.1%	2.1%
True Class	P	4.1%	61.4%	0.6%	4.3%
	QRS	2.5%	1.4%	58.7%	7.3%
n/a	P	56.0%	34.8%	39.6%	86.2%
	Predicted Class				

使用原始 ECG 信号作为网络输入，则只有大约 60% 的 T 波采样、40% 的 T 波采样和 60% 的 QRS 复波采样是正确的。为了提高性能，请在输入到深度学习网络之前应用一些 ECG 信号特性的知识，例如由患者呼吸运动引起的基线漂移。

### 应用滤波方法以消除基线漂移和高频噪声

这三种心跳形态占据不同频带。QRS 复波的典型频谱以大约 10-25 Hz 为中心频率，并且其分量低于 40 Hz。发生 P 波和 T 波的频率甚至更低：P 波分量低于 20 Hz，T 波分量低于 10 Hz [5 (第 12-0 页) ]。

基线漂移是由患者呼吸运动引起的低频 (< 0.5 Hz) 振荡。这种振荡与心跳形态无关，不会提供有意义的信息 [6 (第 12-0 页) ]。

设计一个通带频率范围为 [0.5, 40] Hz 的带通滤波器，以消除漂移和任何高频噪声。消除这些分量可改进 LSTM 训练，因为网络不会学习不相关特征。对 tall 数据元胞数组使用 `cellfun` 以并行方式对数据集进行滤波。

```
% Bandpass filter design
hFilt = designfilt('bandpassiir', 'StopbandFrequency1',0.4215,'PassbandFrequency1', 0.5, ...
    'PassbandFrequency2',40,'StopbandFrequency2',53.345, ...
    'StopbandAttenuation1',60,'PassbandRipple',0.1,'StopbandAttenuation2',60, ...
    'SampleRate',250,'DesignMethod','ellip');

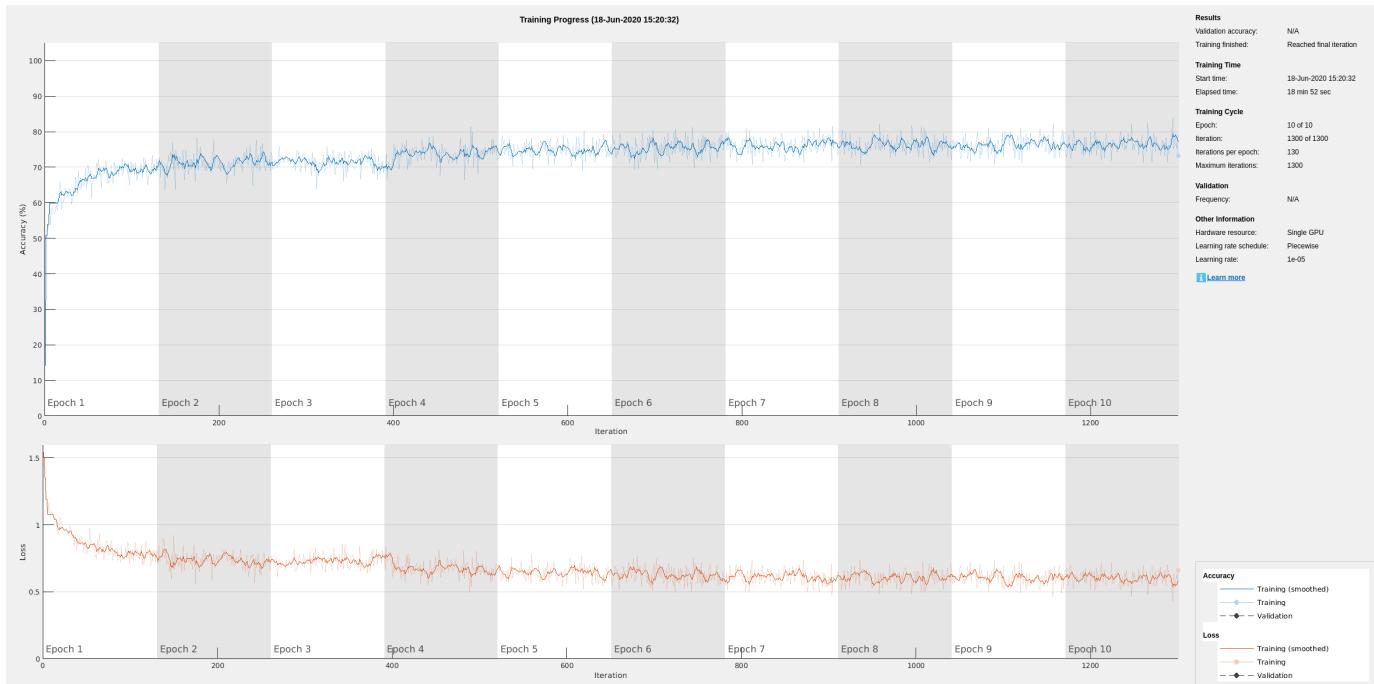
% Create tall arrays from the transformed datastores and filter the signals
tallTrainSet = tall(trainDs);
tallTestSet = tall(testDs);
```

```
filteredTrainSignals = gather(cellfun(@(x)filter(hFilt,x),tallTrainSet(:,1),'UniformOutput',false));
```

Evaluating tall expression using the Parallel Pool 'local':

- Pass 1 of 1: 0% complete

Evaluation 0% complete



- Pass 1 of 1: Completed in 13 sec

Evaluation completed in 14 sec

```
trainLabels = gather(tallTrainSet(:,2));
```

Evaluating tall expression using the Parallel Pool 'local':

- Pass 1 of 1: Completed in 3.6 sec

Evaluation completed in 4 sec

```
filteredTestSignals = gather(cellfun(@(x)filter(hFilt,x),tallTestSet(:,1),'UniformOutput',false));
```

Evaluating tall expression using the Parallel Pool 'local':

- Pass 1 of 1: Completed in 2.4 sec

Evaluation completed in 2.5 sec

```
testLabels = gather(tallTestSet(:,2));
```

Evaluating tall expression using the Parallel Pool 'local':

- Pass 1 of 1: Completed in 1.9 sec

Evaluation completed in 2 sec

对一种典型情况下的原始信号和滤波后的信号绘图。

```
trainData = gather(tallTrainSet);
```

Evaluating tall expression using the Parallel Pool 'local':

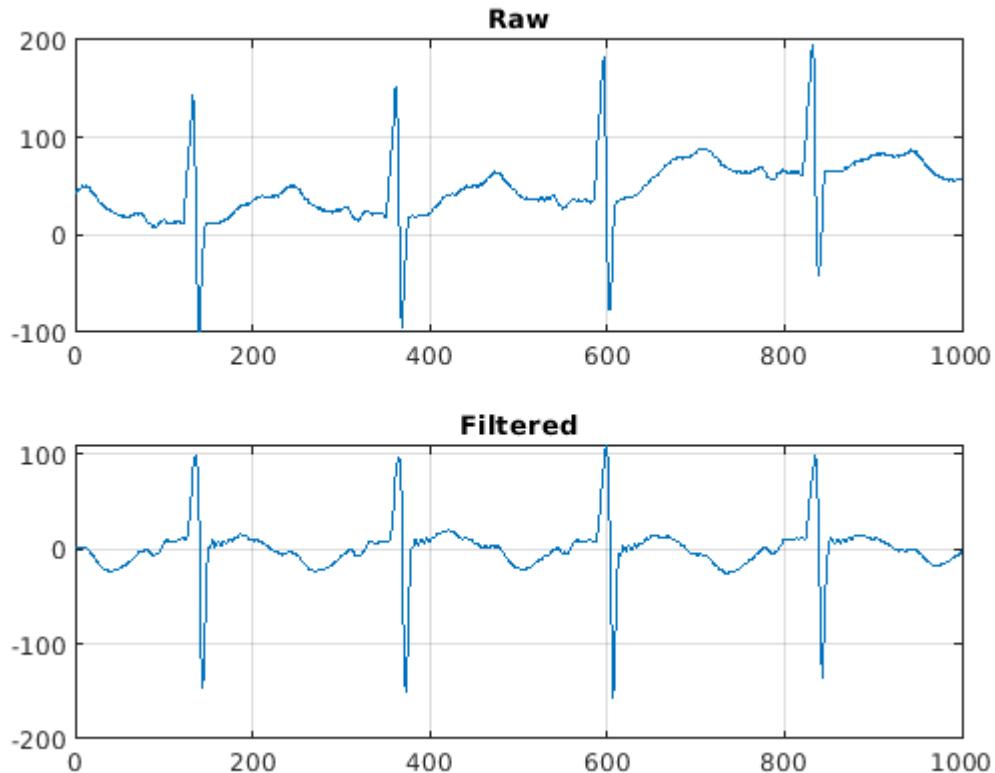
- Pass 1 of 1: Completed in 4 sec

Evaluation completed in 4.2 sec

```

figure
subplot(2,1,1)
plot(trainData{95,1}(2001:3000))
title('Raw')
grid
subplot(2,1,2)
plot(filteredTrainSignals{95}(2001:3000))
title('Filtered')
grid

```



尽管滤波信号的基线可能会使习惯于在医疗设备上进行传统 ECG 测量的医生感到困惑，但实际上网络将受益于漂移消除。

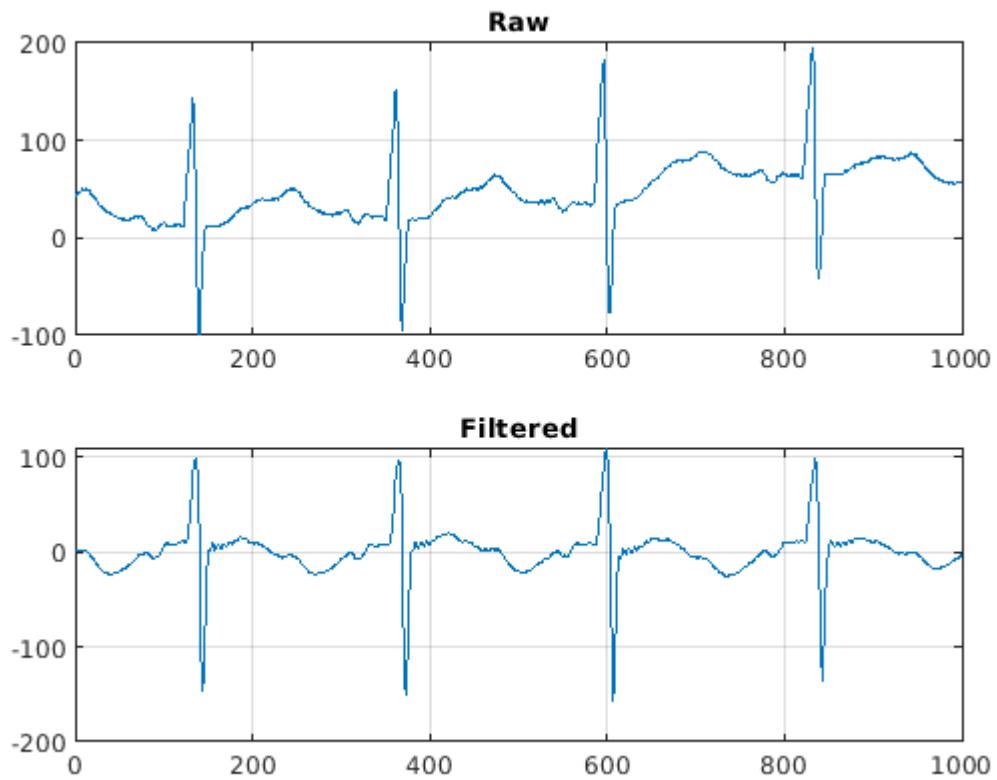
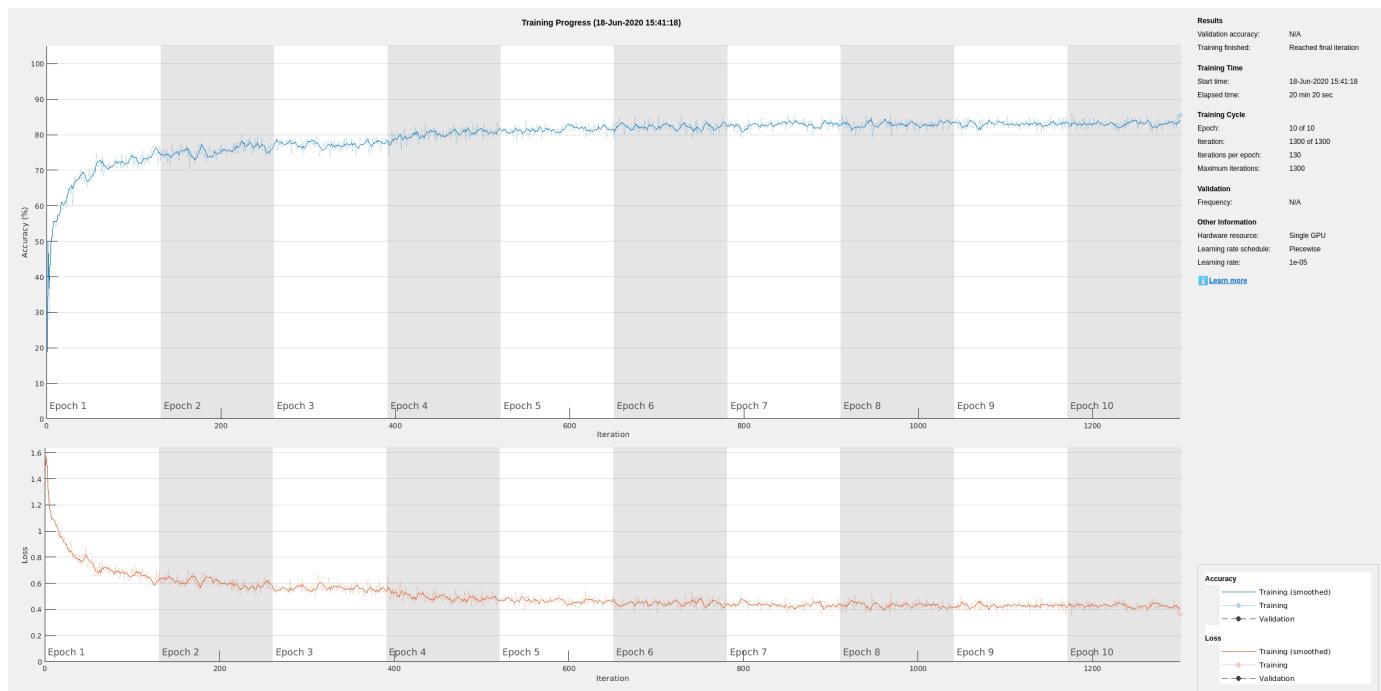
### 使用滤波后的 ECG 信号训练网络

使用与以前相同的网络架构基于滤波后的 ECG 信号训练 LSTM 网络。

```

if actionFlag == "Train networks"
    filteredNet = trainNetwork(filteredTrainSignals,trainLabels,layers,options);
end

```



信号预处理将训练准确度提高到 80% 以上。

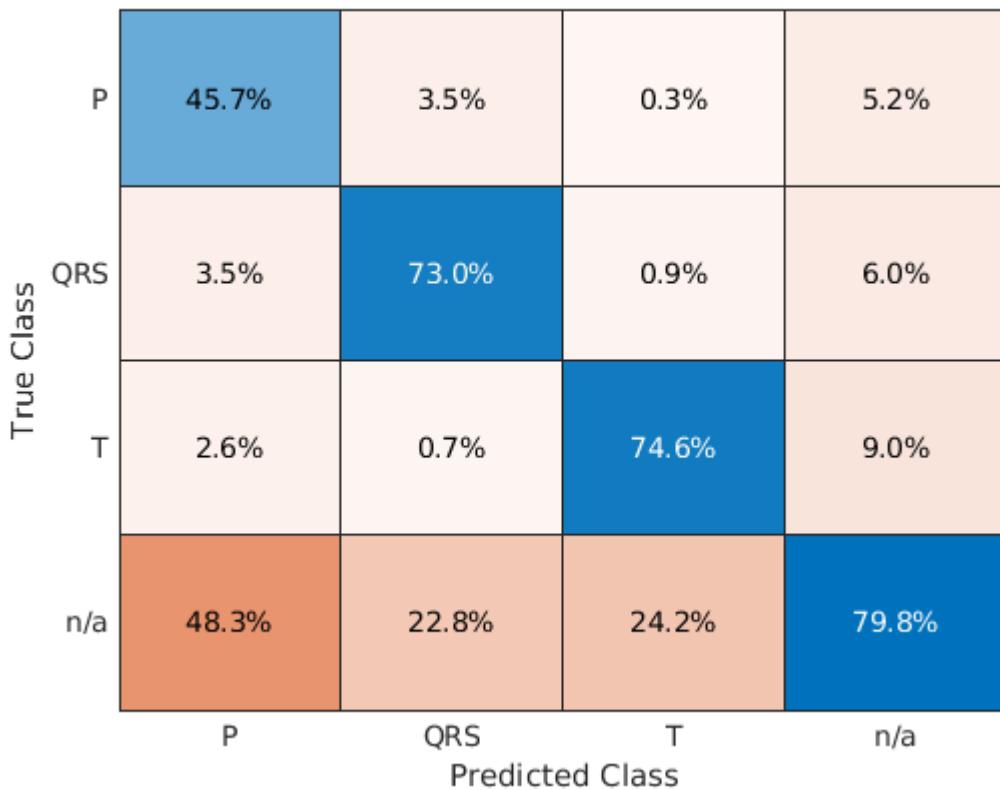
### 对滤波后的 ECG 信号进行分类

用更新后的 LSTM 网络对预处理后的测试数据进行分类。

```
predFilteredTest = classify(filteredNet,filteredTestSignals,'MiniBatchSize',50);
```

将分类性能可视化为混淆矩阵。

```
figure
confusionchart([predFilteredTest{:}],[testLabels{:}],'Normalization','column-normalized');
```



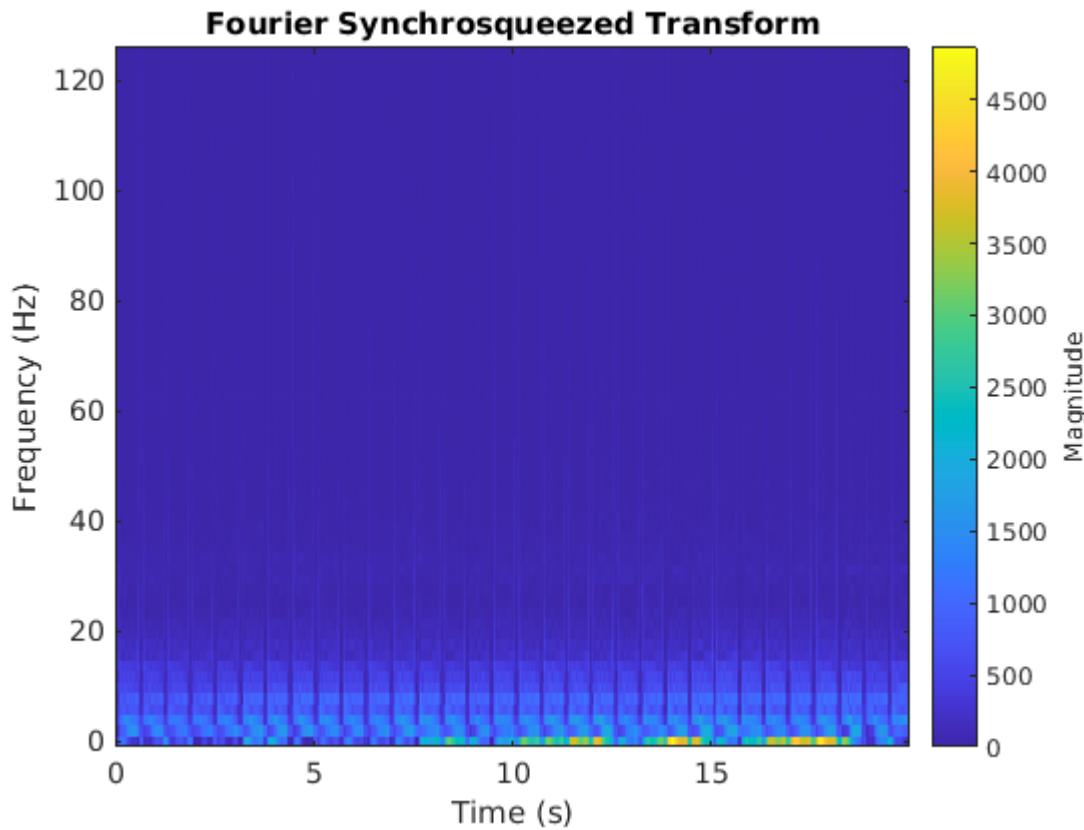
简单的预处理将 T 波分类提高了约 15%，将 QRS 复波和 P 波分类提高了约 10%。

### ECG 信号的时频表示

时序数据成功分类的常见方法是提取时频特征并将其馈送到网络而不是原始数据。然后，网络同时跨时间和频率学习模式 [7 (第 12-0 页)]。

傅里叶同步压缩变换 (FSST) 计算每个信号采样的频谱，因此对于需要保持与原始信号相同的时间分辨率的分割问题，它是可直接使用的理想选择。使用 `fsst` 函数检查一个训练信号的变换。指定长度为 128 的 Kaiser 窗以提供足够的频率分辨率。

```
data = preview(trainDs);
figure
fsst(data{1,1},250,kaiser(128),'yaxis')
```

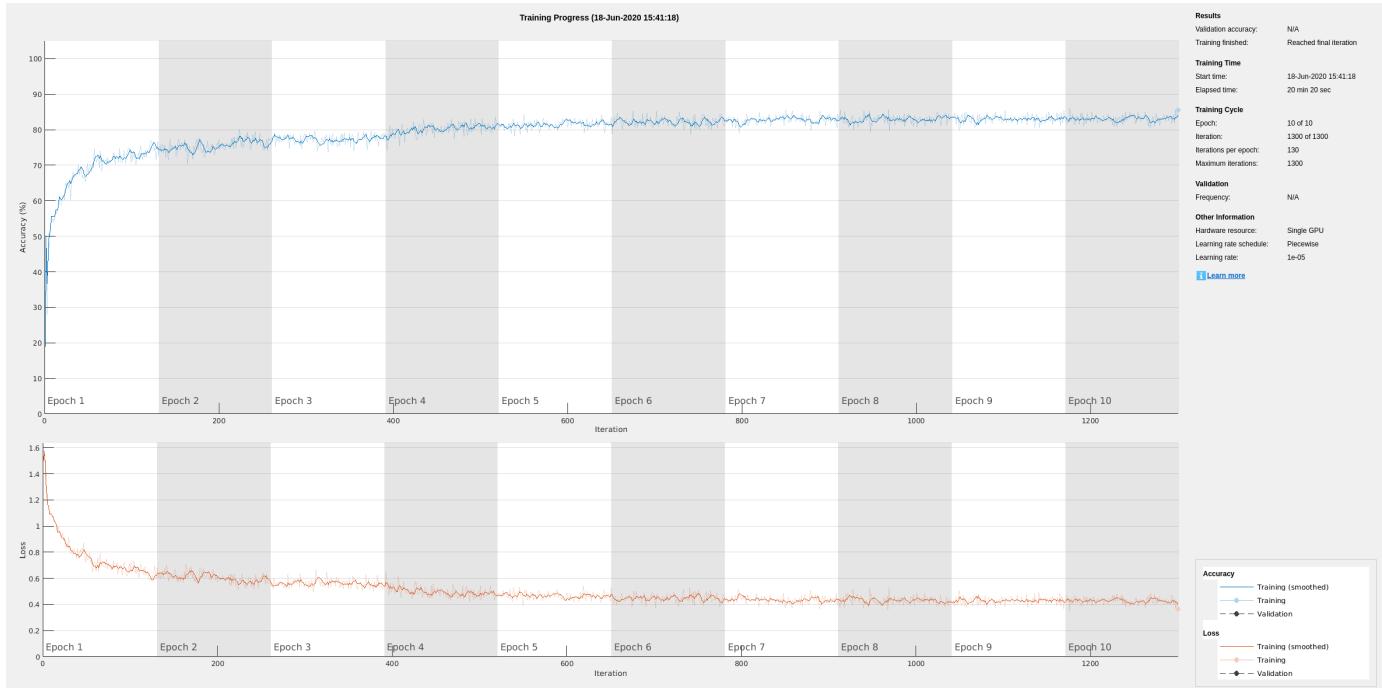


基于关注的频率范围 [0.5, 40] Hz 计算训练数据集中每个信号的 FSST。将 FSST 的实部和虚部视为单独的特征，并将两个分量都馈送到网络中。而且，通过减去均值并除以标准差来标准化训练特征。使用变换后的数据存储、**extractFSSTFeatures** 辅助函数和 **tall** 函数来并行处理数据。

```
fsstTrainDs = transform(trainDs,@(x)extractFSSTFeatures(x,250));
fsstTallTrainSet = tall(fsstTrainDs);
fsstTrainData = gather(fsstTallTrainSet);
```

Evaluating tall expression using the Parallel Pool 'local':

- Pass 1 of 1: 0% complete
- Evaluation 0% complete



- Pass 1 of 1: Completed in 2 min 35 sec  
 Evaluation completed in 2 min 35 sec

对测试数据重复此过程。

```
fsstTTestDs = transform(testDs,@(x)extractFSSTFeatures(x,250));
fsstTallTestSet = tall(fsstTTestDs);
fsstTestData = gather(fsstTallTestSet);
```

Evaluating tall expression using the Parallel Pool 'local':  
 - Pass 1 of 1: Completed in 1 min 4 sec  
 Evaluation completed in 1 min 4 sec

### 调整网络架构

修改 LSTM 架构，使网络接受每个采样的频谱，而不是单一值。检查 FSST 的大小以查看频率的数量。

```
size(fsstTrainData{1,1})
```

```
ans = 1×2
```

```
40      5000
```

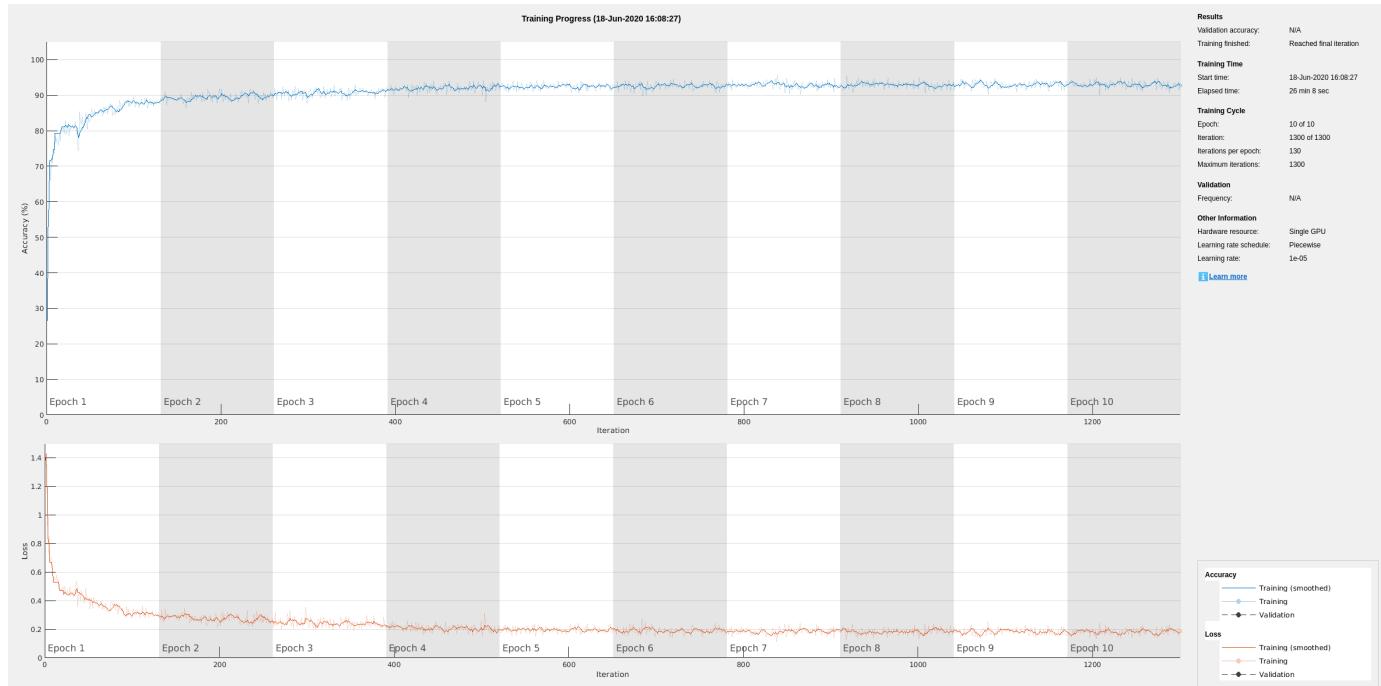
指定一个包含 40 个输入特征的 sequenceInputLayer。保持其余网络参数不变。

```
layers = [ ...
    sequenceInputLayer(40)
    lstmLayer(200,'OutputMode','sequence')
    fullyConnectedLayer(4)
    softmaxLayer
    classificationLayer];
```

## 使用 ECG 信号的 FSST 训练网络

使用变换后的数据集训练更新后的 LSTM 网络。

```
if actionFlag == "Train networks"
    fsstNet = trainNetwork(fsstTrainData(:,1),fsstTrainData(:,2),layers,options);
end
```



使用时频特征提高了训练准确度，现在已超过 90%。

## 用 FSST 对测试数据进行分类

使用更新后的 LSTM 网络和提取的 FSST 特征，对测试数据进行分类。

```
predFsstTest = classify(fsstNet,fsstTestData(:,1),'MiniBatchSize',50);
```

将分类性能可视化为混淆矩阵。

```
confusionchart([predFsstTest{:}],[fsstTestData{:,2}], 'Normalization','column-normalized');
```

	P	QRS	T	n/a
P	80.5%	0.4%	0.3%	3.2%
QRS	0.7%	90.7%	0.3%	2.1%
T	1.0%	0.3%	82.2%	7.7%
n/a	17.8%	8.7%	17.2%	87.1%
Predicted Class				

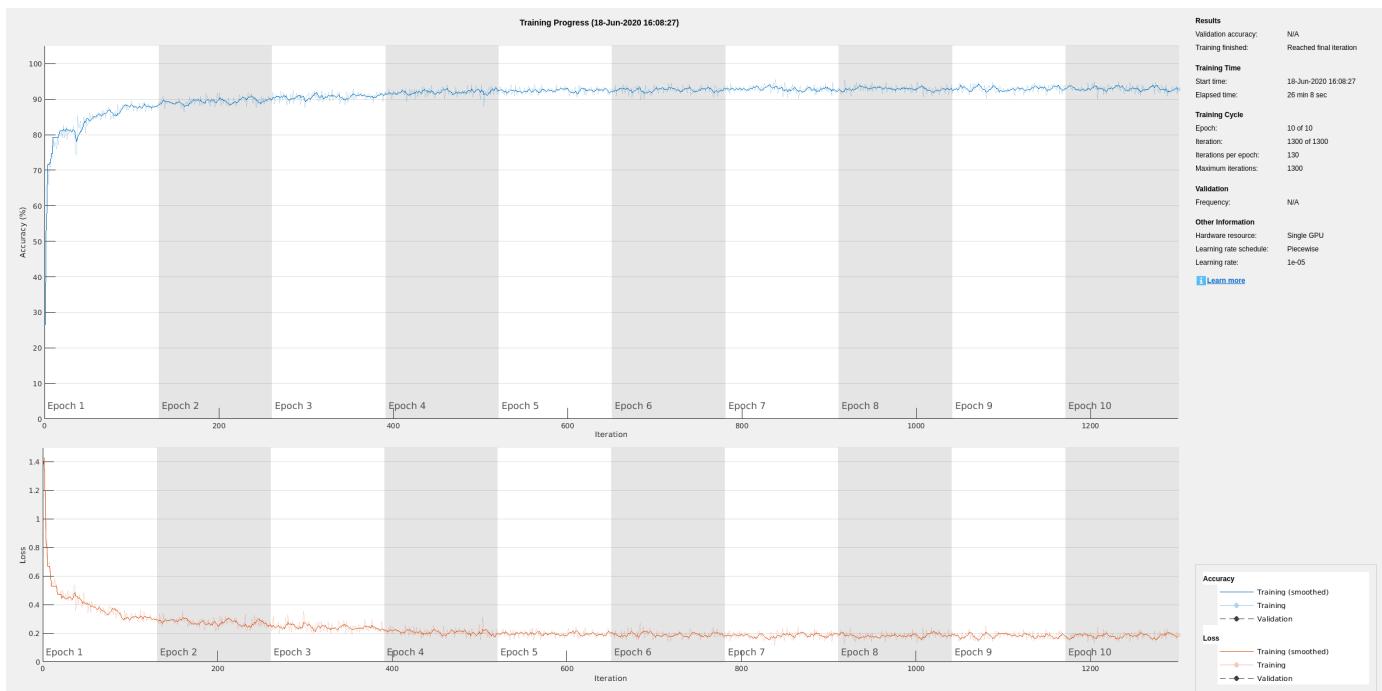
与原始数据结果相比，使用时间频率表示法将 T 波分类提高了约 25%，将 P 波分类提高了约 40%，将 QRS 复波分类提高了 30%。

使用 `signalMask` 对象将网络预测与单个 ECG 信号的真实值标签进行比较。绘制关注区域时忽略 "n/a" 标签。

```
testData = gather(tall(testDs));
```

Evaluating tall expression using the Parallel Pool 'local':

- Pass 1 of 1: 0% complete
- Evaluation 0% complete



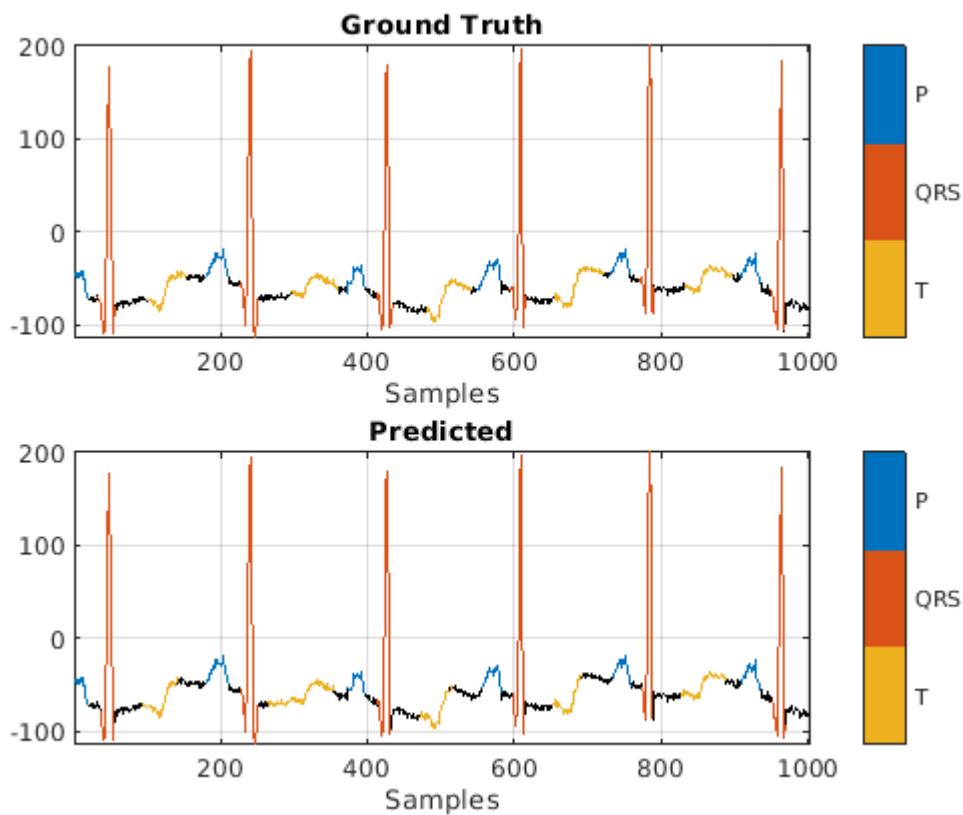
- Pass 1 of 1: Completed in 37 sec  
Evaluation completed in 37 sec

```
Mtest = signalMask(testData{1,2}(3000:4000));
Mtest.SpecifySelectedCategories = true;
Mtest.SelectedCategories = find(Mtest.Categories ~= "n/a");
```

```
figure
subplot(2,1,1)
plotsigroi(Mtest,testData{1,1}(3000:4000))
title('Ground Truth')

Mpred = signalMask(predFsstTest{1}(3000:4000));
Mpred.SpecifySelectedCategories = true;
Mpred.SelectedCategories = find(Mpred.Categories ~= "n/a");

subplot(2,1,2)
plotsigroi(Mpred,testData{1,1}(3000:4000))
title('Predicted')
```



## 结论

此示例说明信号预处理和时频分析是如何提高 LSTM 波形分割性能的。带通滤波和基于傅里叶的同步压缩使所有输出类的平均改进程度从 55% 提高到 85% 左右。

## 参考资料

- [1] McSharry, Patrick E., et al."A dynamical model for generating synthetic electrocardiogram signals."IEEE® Transactions on Biomedical Engineering.Vol. 50, No. 3, 2003, pp. 289–294.
- [2] Laguna, Pablo, Raimon Jané, and Pere Caminal."Automatic detection of wave boundaries in multilead ECG signals:Validation with the CSE database."Computers and Biomedical Research.Vol. 27, No. 1, 1994, pp. 45–60.
- [3] Goldberger, Ary L., Luis A. N. Amaral, Leon Glass, Jeffery M. Hausdorff, Plamen Ch.Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley."PhysioBank, PhysioToolkit, and PhysioNet:Components of a New Research Resource for Complex Physiologic Signals."Circulation.Vol. 101, No. 23, 2000, pp. e215–e220. [Circulation Electronic Pages; <http://circ.ahajournals.org/content/101/23/e215.full>].
- [4] Laguna, Pablo, Roger G. Mark, Ary L. Goldberger, and George B. Moody."A Database for Evaluation of Algorithms for Measurement of QT and Other Waveform Intervals in the ECG."Computers in Cardiology.Vol.24, 1997, pp. 673–676.
- [5] Sörnmo, Leif, and Pablo Laguna."Electrocardiogram (ECG) signal processing."Wiley Encyclopedia of Biomedical Engineering, 2006.

[6] Kohler, B-U., Carsten Hennig, and Reinhold Orglmeister."The principles of software QRS detection."IEEE Engineering in Medicine and Biology Magazine.Vol. 21, No. 1, 2002, pp. 42–57.

[7] Salamon, Justin, and Juan Pablo Bello."Deep convolutional neural networks and data augmentation for environmental sound classification."IEEE Signal Processing Letters.Vol. 24, No. 3, 2017, pp. 279–283.

## 另请参阅

`confusionchart` | `fsst` | `labeledSignalSet` | `lstmLayer` | `trainingOptions` | `trainNetwork`

## 详细信息

- “长短期记忆网络” (第 1-38 页)
- “使用深度学习进行“序列到序列” 回归” (第 4-38 页)
- “使用深度学习进行“序列到序列” 分类” (第 4-33 页)

## 使用长短期记忆网络对 ECG 信号进行分类

此示例说明如何使用深度学习和信号处理对来自 PhysioNet 2017 Challenge 的心电图 (ECG) 数据进行分类。具体而言，该示例使用长短期记忆网络和时频分析。

### 简介

ECG 记录一段时间内人体心脏的电活动。医生使用 ECG 检测患者的心跳是正常还是不规则。

心房纤维性颤动 (AFib) 是一种不规则的心跳，当心脏的上腔（即心房）与下腔（即心室）失去协调时就会发生心房纤维性颤动。

此示例使用的 ECG 数据来自 PhysioNet 2017 Challenge [1 (第 12-0 页)]、[2 (第 12-0 页)]、[3 (第 12-0 页)]，可从 <https://physionet.org/challenge/2017/> 获得。数据由一组以 300 Hz 采样的 ECG 信号组成，由一组专家分成四个不同类：正常 (N)、AFib (A)、其他心律 (O) 和含噪记录 (~)。此示例说明如何使用深度学习来自动化分类过程。该过程使用一个二类分类器，该二类分类器可以将正常 ECG 信号和显示 AFib 符号的信号区分开来。

本示例使用长短期记忆 (LSTM) 网络，这是一种循环神经网络 (RNN)，非常适合研究序列和时序数据。LSTM 网络可以学习序列的时间步之间的长期相关性。LSTM 层 (`lstmLayer`) 可以前向分析时间序列，而双向 LSTM 层 (`bilstmLayer`) 可以前向和后向分析时间序列。此示例使用双向 LSTM 层。

为了加快训练过程，请在具有 GPU 的机器上运行此示例。如果您的机器同时有 GPU 和 Parallel Computing Toolbox™，则 MATLAB® 会自动使用 GPU 进行训练；否则，它使用 CPU。

### 加载并检查数据

运行 `ReadPhysionetData` 脚本，从 PhysioNet 网站下载数据，并生成包含适当格式的 ECG 信号的 MAT 文件 (`PhysionetData.mat`)。下载数据可能需要几分钟。仅在当前文件夹中不存在 `PhysionetData.mat` 时，才使用运行脚本的条件语句。

```
if ~isfile('PhysionetData.mat')
    ReadPhysionetData
end
load PhysionetData
```

加载操作会向工作区添加两个变量：`Signals` 和 `Labels`。`Signals` 是保存 ECG 信号的元胞数组。`Labels` 是分类数组，它保存信号的对应真实值标签。

`Signals(1:5)`

```
ans=5×1 cell array
{1×9000 double}
{1×9000 double}
{1×18000 double}
{1×9000 double}
{1×18000 double}
```

`Labels(1:5)`

```
ans = 5×1 categorical
N
N
N
N
A
```

A

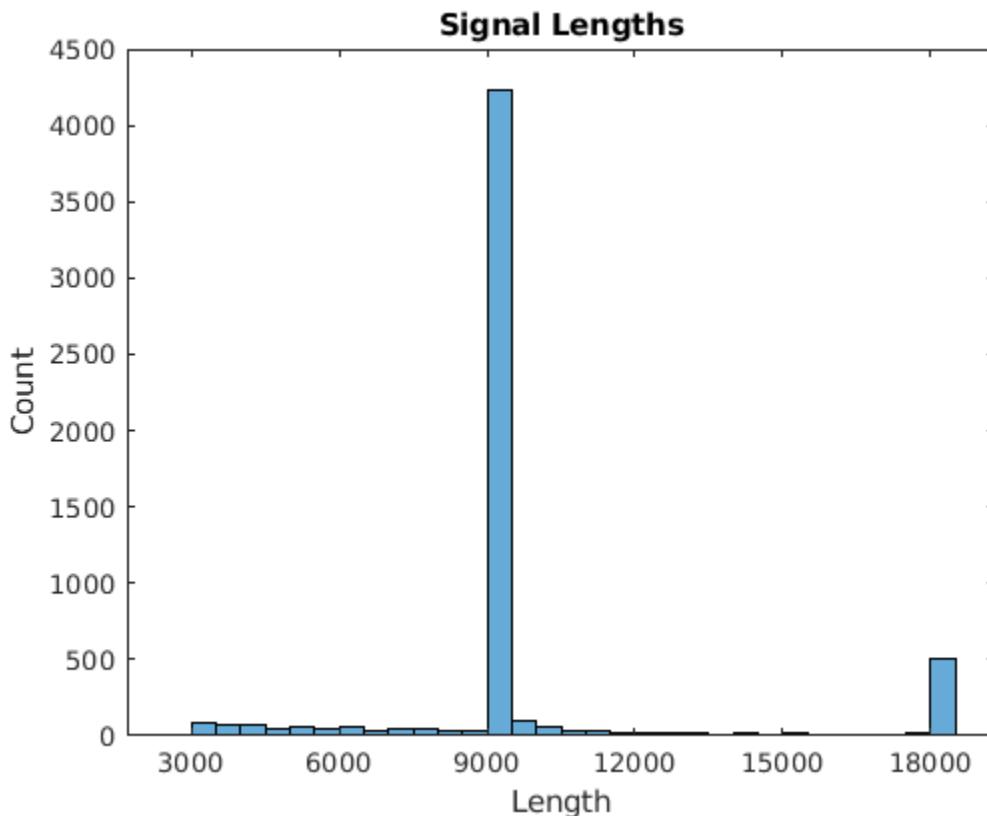
使用 `summary` 函数查看数据中包含多少 AFib 信号和正常信号。

```
summary(Labels)
```

A	738
N	5050

生成信号长度的直方图。大多数信号的长度是 9000 个采样。

```
L = cellfun(@length, Signals);
h = histogram(L);
xticks(0:3000:18000);
xticklabels(0:3000:18000);
title('Signal Lengths')
xlabel('Length')
ylabel('Count')
```



可视化每个类中一个信号的一段。AFib 心跳间隔不规则，而正常心跳会周期性发生。AFib 心跳信号还经常缺失 P 波，P 波在正常心跳信号的 QRS 复合波之前出现。正常信号的绘图会显示 P 波和 QRS 复合波。

```
normal = Signals{1};
aFib = Signals{4};

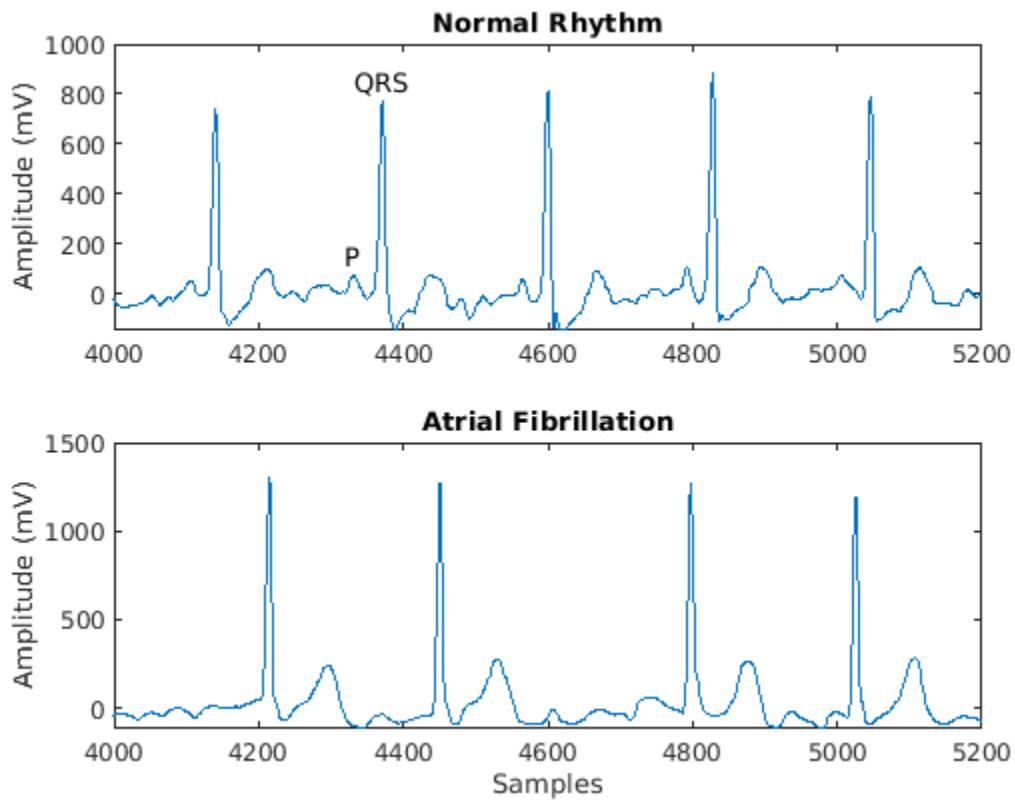
subplot(2,1,1)
plot(normal)
```

```

title('Normal Rhythm')
xlim([4000,5200])
ylabel('Amplitude (mV)')
text(4330,150,'P','HorizontalAlignment','center')
text(4370,850,'QRS','HorizontalAlignment','center')

subplot(2,1,2)
plot(aFib)
title('Atrial Fibrillation')
xlim([4000,5200])
xlabel('Samples')
ylabel('Amplitude (mV)')

```



### 准备用于训练的数据

在训练期间，`trainNetwork` 函数将数据分成小批量。然后，该函数在同一个小批量中填充或截断信号，使它们都具有相同的长度。过多的填充或截断会对网络性能产生负面影响，因为网络可能会根据添加或删除的信息错误地解释信号。

为避免过度填充或截断，请对 ECG 信号应用 `segmentSignals` 函数，使它们的长度都为 9000 个样本。该函数会忽略少于 9000 个样本的信号。如果信号的样本超过 9000 个，`segmentSignals` 会将其分成尽可能多的包含 9000 个样本的信号段，并忽略剩余样本。例如，具有 18500 个样本的信号将变为两个包含 9000 个样本的信号，剩余的 500 个样本被忽略。

```
[Signals,Labels] = segmentSignals(Signals,Labels);
```

查看 `Signals` 数组的前五个元素，以验证每个条目的长度现在为 9000 个样本。

**Signals(1:5)**

```
ans=5×1 cell array
{1×9000 double}
{1×9000 double}
{1×9000 double}
{1×9000 double}
{1×9000 double}
```

**使用原始信号数据训练分类器**

要设计分类器，请使用上一节中生成的原始信号。将信号分成一个训练集（用于训练分类器）和一个测试集（用于基于新数据测试分类器的准确度）。

使用 **summary** 函数显示 AFib 信号与正常信号的比率为 718:4937，约为 1:7。

**summary(Labels)**

A	718
N	4937

由于约 7/8 的信号是正常信号，因此分类器会发现通过简单地将所有信号分类为正常信号就可达到高准确度。为了避免这种偏置，需要通过复制数据集中的 AFib 信号来增加 AFib 数据，以便正常信号和 AFib 信号的数量相同。这种复制通常称为过采样，是深度学习中使用的一种数据增强形式。

根据信号所属的类划分信号。

```
afibX = Signals(Labels=='A');
afibY = Labels(Labels=='A');

normalX = Signals(Labels=='N');
normalY = Labels(Labels=='N');
```

接下来，使用 **dividerand** 将每个类的目标随机分为训练集和测试集。

```
[trainIndA,~,testIndA] = dividerand(718,0.9,0.0,0.1);
[trainIndN,~,testIndN] = dividerand(4937,0.9,0.0,0.1);

XTrainA = afibX(trainIndA);
YTrainA = afibY(trainIndA);

XTrainN = normalX(trainIndN);
YTrainN = normalY(trainIndN);

XTestA = afibX(testIndA);
YTestA = afibY(testIndA);

XTestN = normalX(testIndN);
YTestN = normalY(testIndN);
```

现在有 646 个 AFib 信号和 4443 个正常信号用于训练。要在每个类中获得相同数量的信号，请使用前 4438 个正常信号，然后使用 **repmat** 对前 634 个 AFib 信号重复七次。

对于测试集，现在有 72 个 AFib 信号和 494 个正常信号。使用前 490 个正常信号，然后使用 **repmat** 对前 70 个 AFib 信号重复七次。默认情况下，神经网络会在训练前随机对数据进行乱序处理，以确保相邻信号不都有相同的标签。

```
XTrain = [repmat(XTrainA(1:634),7,1); XTrainN(1:4438)];
YTrain = [repmat(YTrainA(1:634),7,1); YTrainN(1:4438)];
```

```
XTest = [repmat(XTestA(1:70),7,1); XTestN(1:490)];
YTest = [repmat(YTestA(1:70),7,1); YTestN(1:490)];
```

现在，正常信号和 AFib 信号在训练集和测试集中的分布均衡。

```
summary(YTrain)
```

```
A    4438
N    4438
```

```
summary(YTest)
```

```
A    490
N    490
```

### 定义 LSTM 网络架构

LSTM 网络可以学习序列数据的时间步之间的长期相关性。此示例使用双向 LSTM 层 `bilstmLayer`，因为它前向和后向检测序列。

由于输入信号各有一个维度，将输入大小指定是大小为 1 的序列。指定输出大小为 100 的一个双向 LSTM 层，并输出序列的最后一个元素。此命令指示双向 LSTM 层将输入时序映射到 100 个特征，然后为全连接层准备输出。最后，通过包含大小为 2 的全连接层，后跟 softmax 层和分类层，来指定两个类。

```
layers = [
    sequenceInputLayer(1)
    bilstmLayer(100,'OutputMode','last')
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer
]

layers =
5x1 Layer array with layers:

1 " Sequence Input      Sequence input with 1 dimensions
2 " BiLSTM              BiLSTM with 100 hidden units
3 " Fully Connected     2 fully connected layer
4 " Softmax              softmax
5 " Classification Output crossentropyex
```

接下来指定分类器的训练选项。将 `'MaxEpochs'` 设置为 10，以允许基于训练数据对网络进行 10 轮训练。`'MiniBatchSize'` 为 150 指示网络一次分析 150 个训练信号。`'InitialLearnRate'` 为 0.01 有助于加快训练过程。指定 `'SequenceLength'` 为 1000 以将信号分解成更小的片段，这样机器就不会因为一次处理太多数据而耗尽内存。将 `'GradientThreshold'` 设置为 1 以防止梯度过大，从而稳定训练过程。将 `'Plots'` 指定为 `'training-progress'`，以生成显示训练随迭代次数的增加而变化的进度图。将 `'Verbose'` 设置为 `false` 以隐藏对应于图中所示数据的表输出。如果您要查看此表，请将 `'Verbose'` 设置为 `true`。

此示例使用自适应矩估计 (ADAM) 求解器。与默认的具有动量的随机梯度下降 (SGDM) 求解器相比，ADAM 在使用 LSTM 之类的 RNN 时性能更好。

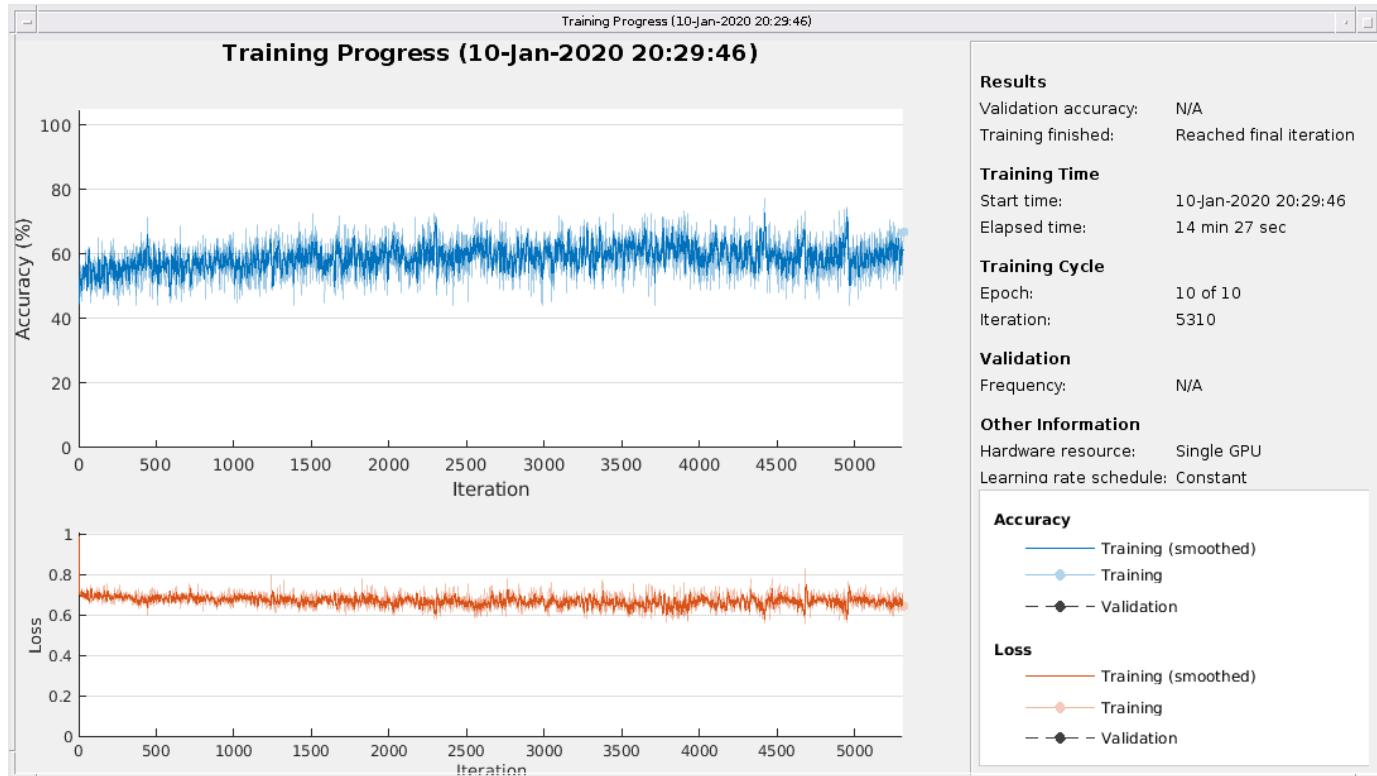
```
options = trainingOptions('adam',...
    'MaxEpochs',10, ...
    'MiniBatchSize', 150, ...
    'InitialLearnRate', 0.01, ...
```

```
'SequenceLength', 1000, ...
'GradientThreshold', 1, ...
'ExecutionEnvironment','auto',...
'plots','training-progress',...
'Verbose',false);
```

## 训练 LSTM 网络

通过使用 `trainNetwork` 用指定的训练选项和层架构训练 LSTM 网络。由于训练集很大，训练过程可能需要几分钟。

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



训练进度图的顶部子图表示训练准确度，即基于每个小批量的分类准确度。当训练在成功进行时，此值通常会逐渐增大，直到 100%。底部子图显示训练损失，即基于每个小批量的交叉熵损失。当训练在成功进行时，该值通常会逐渐降低，直到为零。

如果训练未收敛，绘图可能会在各值之间振荡，而不会呈现向上或向下趋势。这种振荡意味着训练准确度没有提高，训练损失没有减少。这种情况可能发生在训练开始时，或者在训练准确度有初步提高后，绘图可能趋于平稳。在许多情况下，更改训练选项可以帮助网络实现收敛。减少 `MiniBatchSize` 或减少 `InitialLearnRate` 可能会导致更长的训练时间，但这可能有助于网络更好地学习。

分类器的训练准确度在约 50% 和约 60% 之间震荡，在 10 轮结束时，训练已进行了几分钟。

## 可视化训练和测试准确度

计算训练准确度，该准确度表示分类器对于所训练信号的准确度。首先，对训练数据进行分类。

```
trainPred = classify(net,XTrain,'SequenceLength',1000);
```

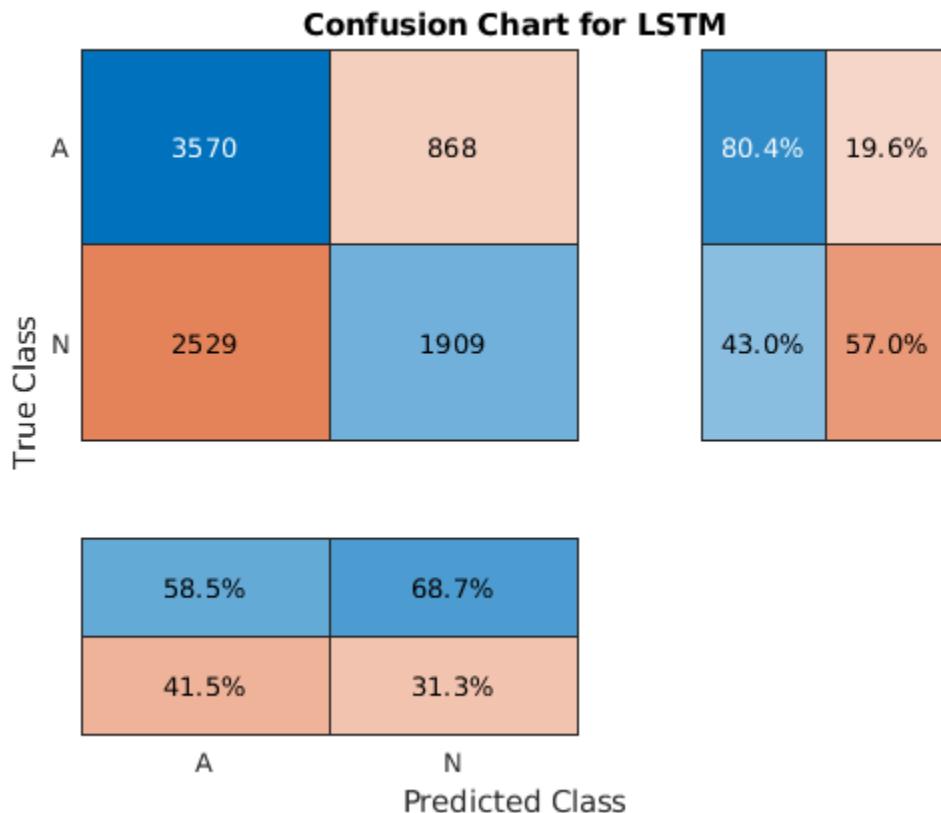
在分类问题中，混淆矩阵用于可视化分类器对于一组已知真实数值的数据上的性能。目标类是信号的真实值标签，输出类是网络分配给信号的标签。坐标区标签表示类标签 AFib (A) 和 Normal (N)。

使用 `confusionchart` 命令计算用于测试数据预测的总体分类准确度。将 'RowSummary' 指定为 'row-normalized' 以在行汇总中显示真正率和假正率。此外，将 'ColumnSummary' 指定为 'column-normalized' 以在列汇总中显示正预测值和假发现率。

```
LSTMAccuracy = sum(trainPred == YTrain)/numel(YTrain)*100
```

```
LSTMAccuracy = 61.7283
```

```
figure  
confusionchart(YTrain,trainPred,'ColumnSummary','column-normalized',...  
    'RowSummary','row-normalized','Title','Confusion Chart for LSTM');
```



现在用相同的网络对测试数据进行分类。

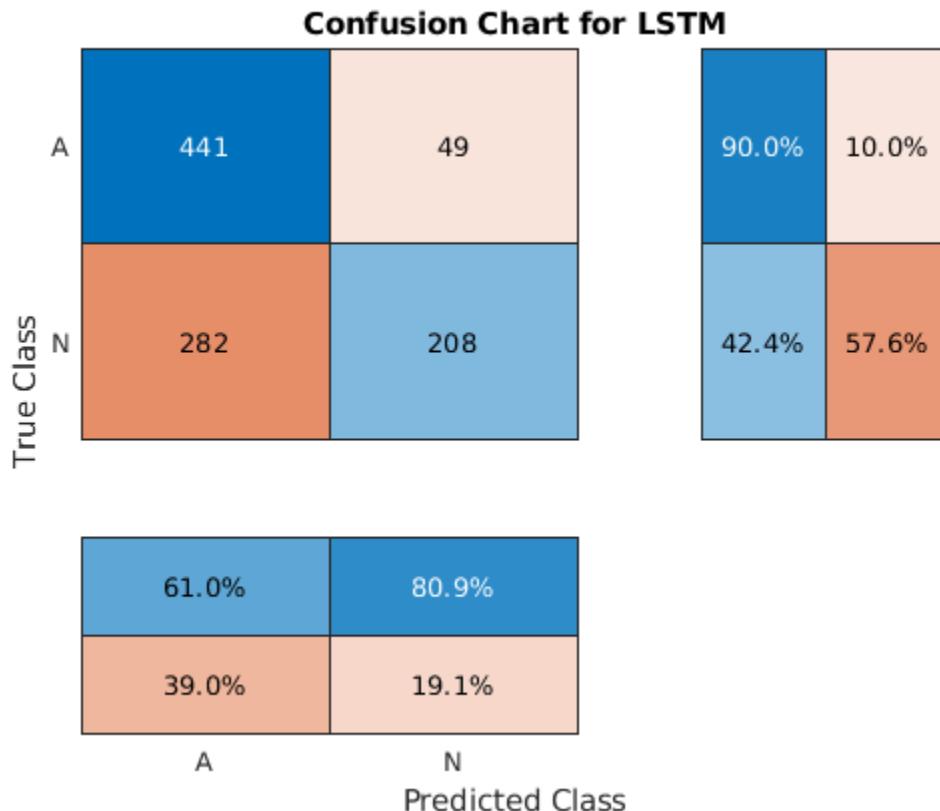
```
testPred = classify(net,XTest,'SequenceLength',1000);
```

计算测试准确度，并使用混淆矩阵将分类性能可视化。

```
LSTMAccuracy = sum(testPred == YTest)/numel(YTest)*100
```

```
LSTMAccuracy = 66.2245
```

```
figure  
confusionchart(YTest,testPred,'ColumnSummary','column-normalized',...  
    'RowSummary','row-normalized','Title','Confusion Chart for LSTM');
```



### 通过特征提取提高性能

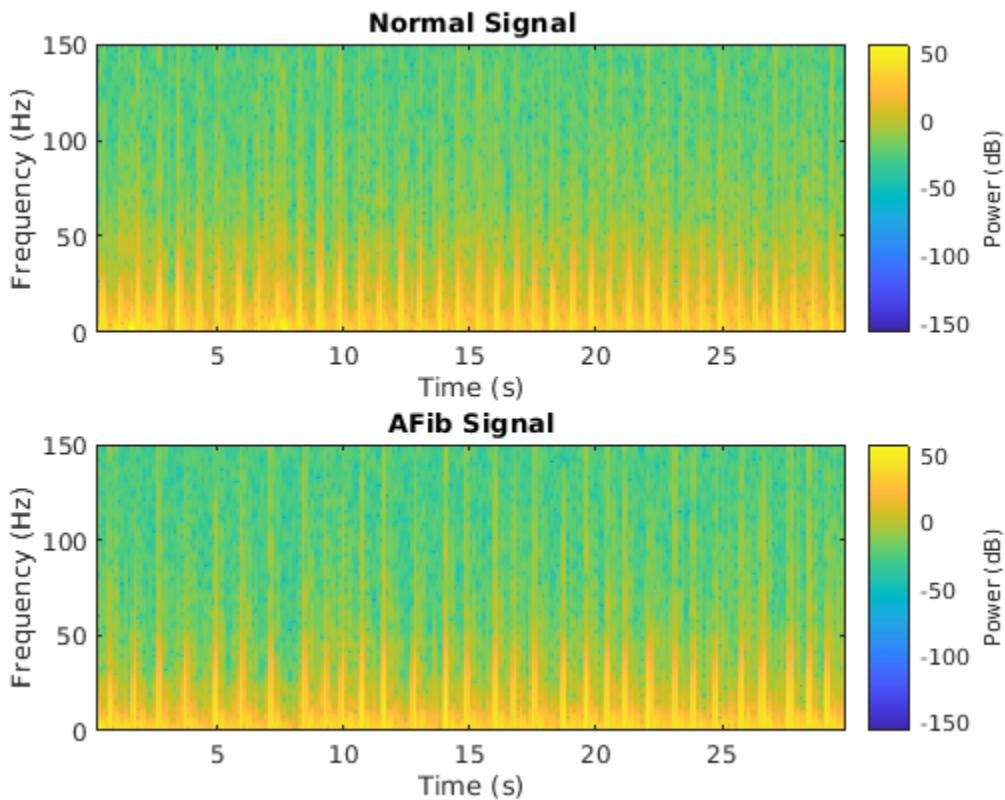
从数据中提取特征有助于提高分类器的训练和测试准确度。为了决定提取哪些特征，本示例采用的方法是先计算时频图像（如频谱图），然后使用它们来训练卷积神经网络 (CNN) [4 (第 12-0 页) ]、[5 (第 12-0 页) ]。

可视化每个信号类型的频谱图。

```
fs = 300;
```

```
figure
subplot(2,1,1);
pspectrum(normal,fs,'spectrogram','TimeResolution',0.5)
title('Normal Signal')

subplot(2,1,2);
pspectrum(aFib,fs,'spectrogram','TimeResolution',0.5)
title('AFib Signal')
```



因为本示例使用 LSTM 而不是 CNN，必须转换该方法以应用于一维信号。时频 (TF) 矩从频谱图中提取信息。每个矩都可以用作一维特征以输入到 LSTM。

探查时域中的两个 TF 矩：

- 瞬时频率 (instfreq)
- 谱熵 (pentropy)

`instfreq` 函数估计信号的时变频率，作为功率谱图的第一个矩。该函数使用时间窗上的短时傅里叶变换计算频谱图。在本示例中，该函数使用 255 个时间窗。该函数的时间输出对应于时间窗的中心。

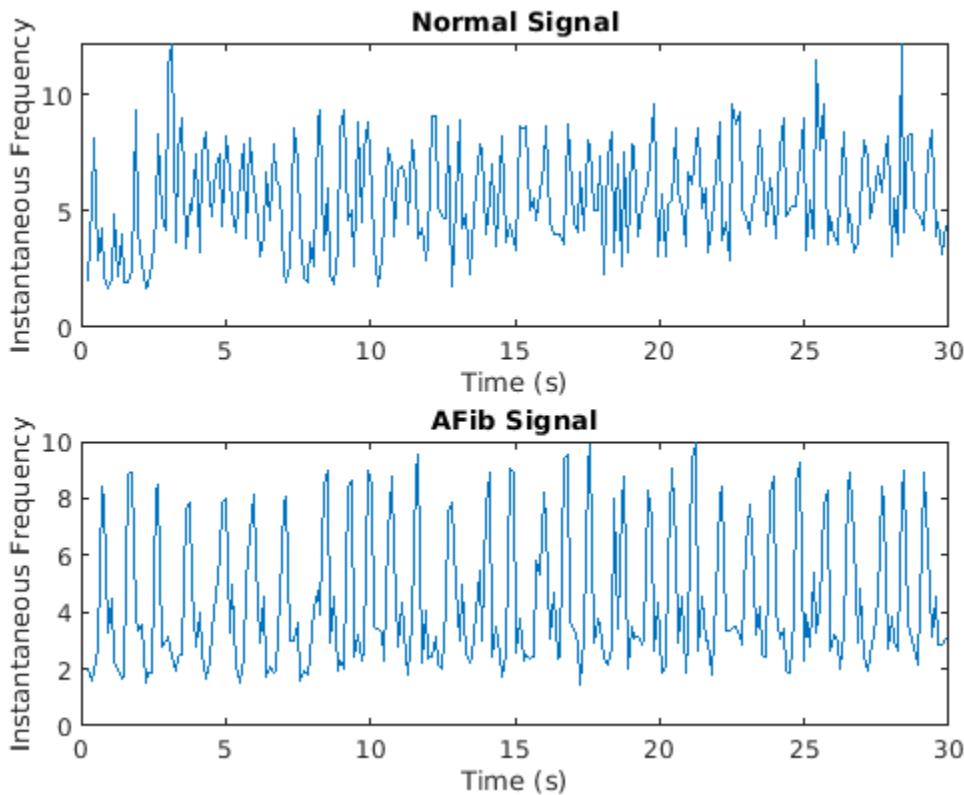
可视化每个信号类型的瞬时频率。

```
[instFreqA,tA] = instfreq(aFib,fs);
[instFreqN,tN] = instfreq(normal,fs);
```

```
figure
subplot(2,1,1);
plot(tN,instFreqN)
title('Normal Signal')
xlabel('Time (s)')
ylabel('Instantaneous Frequency')

subplot(2,1,2);
plot(tA,instFreqA)
title('AFib Signal')
```

```
xlabel('Time (s)')
ylabel('Instantaneous Frequency')
```



使用 `cellfun` 将 `instfreq` 函数应用于训练集和测试集中的每个单元。

```
instfreqTrain = cellfun(@(x)instfreq(x,fs)',XTrain,'UniformOutput',false);
instfreqTest = cellfun(@(x)instfreq(x,fs)',XTest,'UniformOutput',false);
```

谱熵测量信号的频谱的尖度或平坦度。具有尖峰频谱的信号（如正弦波之和）具有低谱熵。具有平坦频谱的信号（如白噪声）具有高谱熵。`pentropy` 函数基于功率谱估计谱熵。与瞬时频率估计情况一样，`pentropy` 使用 255 个时间窗来计算频谱图。函数的时间输出对应于时间窗的中心。

可视化每个信号类型的谱熵。

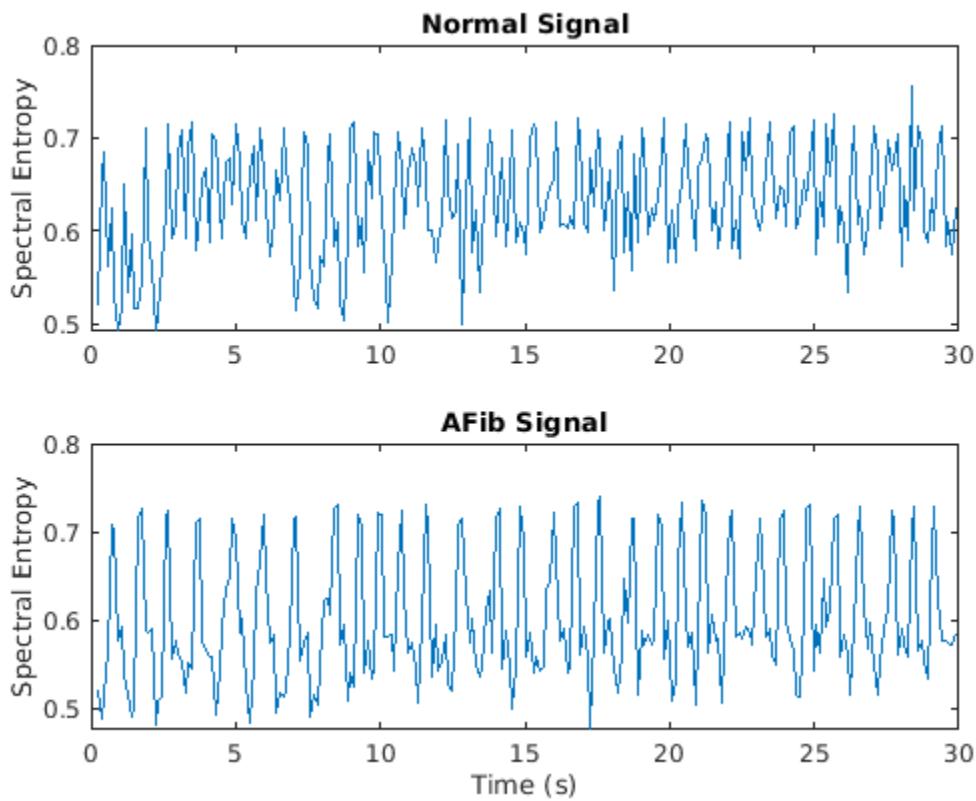
```
[pentropyA,tA2] = pentropy(aFib,fs);
[pentropyN,tN2] = pentropy(normal,fs);
```

```
figure
```

```
subplot(2,1,1)
plot(tN2,pentropyN)
title('Normal Signal')
ylabel('Spectral Entropy')

subplot(2,1,2)
plot(tA2,pentropyA)
title('AFib Signal')
```

```
xlabel('Time (s)')
ylabel('Spectral Entropy')
```



使用 **cellfun** 将 **pentropy** 函数应用于训练中和测试集中的每个单元。

```
pentropyTrain = cellfun(@(x)pentropy(x,fs)',XTrain,'UniformOutput',false);
pentropyTest = cellfun(@(x)pentropy(x,fs)',XTest,'UniformOutput',false);
```

串联这些特征，使新的训练集和测试集中的每个单元都有两个维度（即两个特征）。

```
XTrain2 = cellfun(@(x,y)[x;y],instfreqTrain,pentropyTrain,'UniformOutput',false);
XTest2 = cellfun(@(x,y)[x;y],instfreqTest,pentropyTest,'UniformOutput',false);
```

可视化新输入的格式。每个单元不再包含一个长度为 9000 个样本的信号；现在它包含两个长度为 255 个样本的特征。

```
XTrain2(1:5)
```

```
ans=5×1 cell array
{2×255 double}
{2×255 double}
{2×255 double}
{2×255 double}
{2×255 double}
```

## 标准化数据

瞬时频率和谱熵的均值相差几乎一个数量级。而且，瞬时频率均值可能太高，以致 LSTM 无法高效学习。当网络适合于均值和极差较大的数据时，大的输入可能会减慢网络的学习和收敛速度 [6 (第 12-0 页) ]。

```
mean(instFreqN)
```

```
ans = 5.5615
```

```
mean(pentropyN)
```

```
ans = 0.6326
```

使用训练集均值和标准差来标准化训练集和测试集。标准化，或 z 分数，是一种在训练过程中提高网络性能的常用方法。

```
XV = [XTrain2{:}];  
mu = mean(XV,2);  
sg = std(XV,[],2);  
  
XTrainSD = XTrain2;  
XTrainSD = cellfun(@(x)(x-mu)./sg,XTrainSD,'UniformOutput',false);  
  
XTestSD = XTest2;  
XTestSD = cellfun(@(x)(x-mu)./sg,XTestSD,'UniformOutput',false);
```

显示标准化瞬时频率和谱熵的均值。

```
instFreqNSD = XTrainSD{1}(1,:);  
pentropyNSD = XTrainSD{1}(2,:);
```

```
mean(instFreqNSD)
```

```
ans = -0.3211
```

```
mean(pentropyNSD)
```

```
ans = -0.2416
```

## 修改 LSTM 网络架构

现在每个信号都有两个维度，就有必要通过将输入序列大小指定为 2 来修改网络架构。指定输出大小为 100 的一个双向 LSTM 层，并输出序列的最后一个元素。通过使用一个大小为 2 的全连接层，后跟 softmax 层和分类层，来指定两个类。

```
layers = [...  
sequenceInputLayer(2)  
biLSTMLayer(100,'OutputMode','last')  
fullyConnectedLayer(2)  
softmaxLayer  
classificationLayer  
]  
  
layers =  
5x1 Layer array with layers:  
1 " Sequence Input      Sequence input with 2 dimensions  
2 " BiLSTM              BiLSTM with 100 hidden units
```

```

3 " Fully Connected    2 fully connected layer
4 " Softmax           softmax
5 " Classification Output crossentropyex

```

指定训练选项。将最大轮数设置为 30，以允许基于训练数据对网络进行 30 轮训练。

```

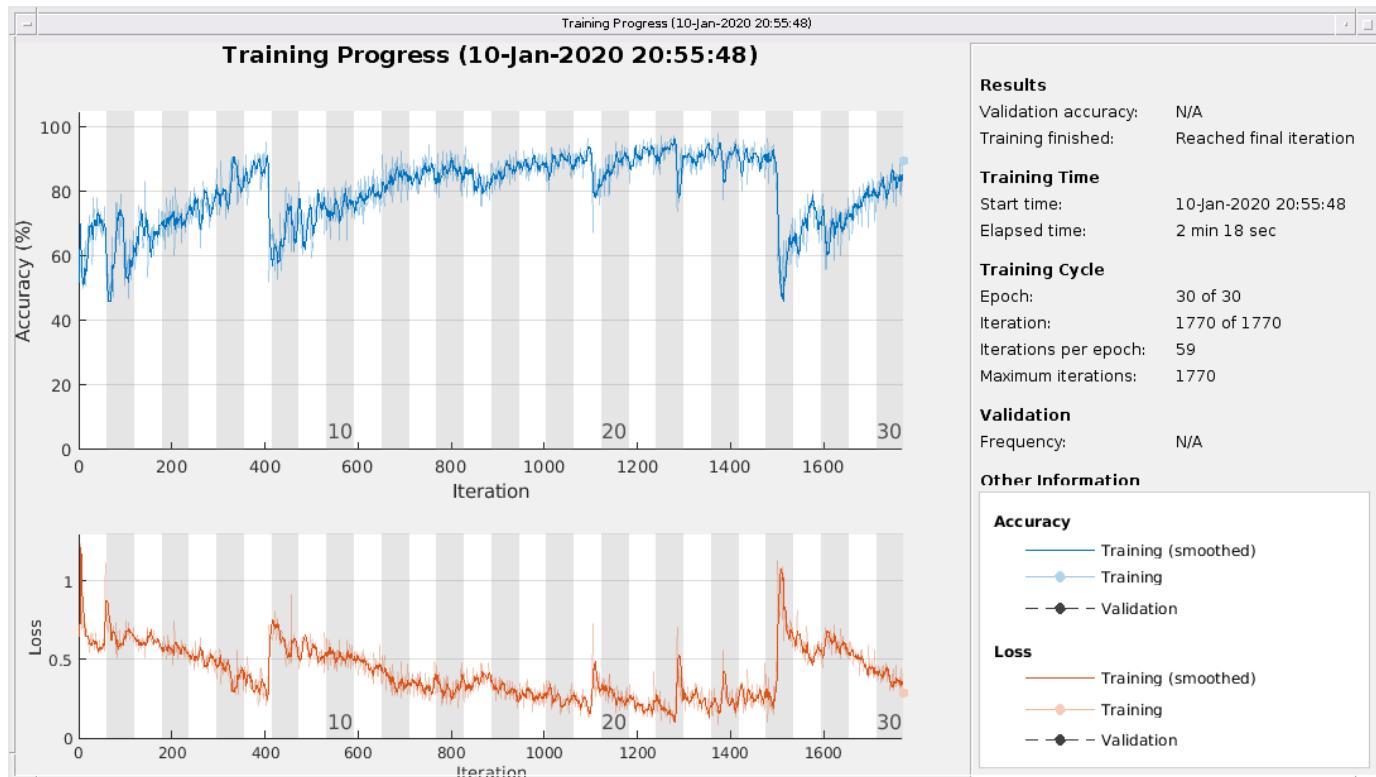
options = trainingOptions('adam', ...
    'MaxEpochs',30, ...
    'MiniBatchSize', 150, ...
    'InitialLearnRate', 0.01, ...
    'GradientThreshold', 1, ...
    'ExecutionEnvironment',"auto",...
    'plots','training-progress', ...
    'Verbose',false);

```

### 用时频特征训练 LSTM 网络

通过使用 `trainNetwork` 用指定的训练选项和层架构训练 LSTM 网络。

```
net2 = trainNetwork(XTrainSD,YTrain,layers,options);
```



训练准确度有很大提高。交叉熵损失趋向于 0。而且，训练所需的时间减少，因为 TF 矩比原始序列短。

### 可视化训练和测试准确度

使用更新后的 LSTM 网络对训练数据进行分类。将分类性能可视化为混淆矩阵。

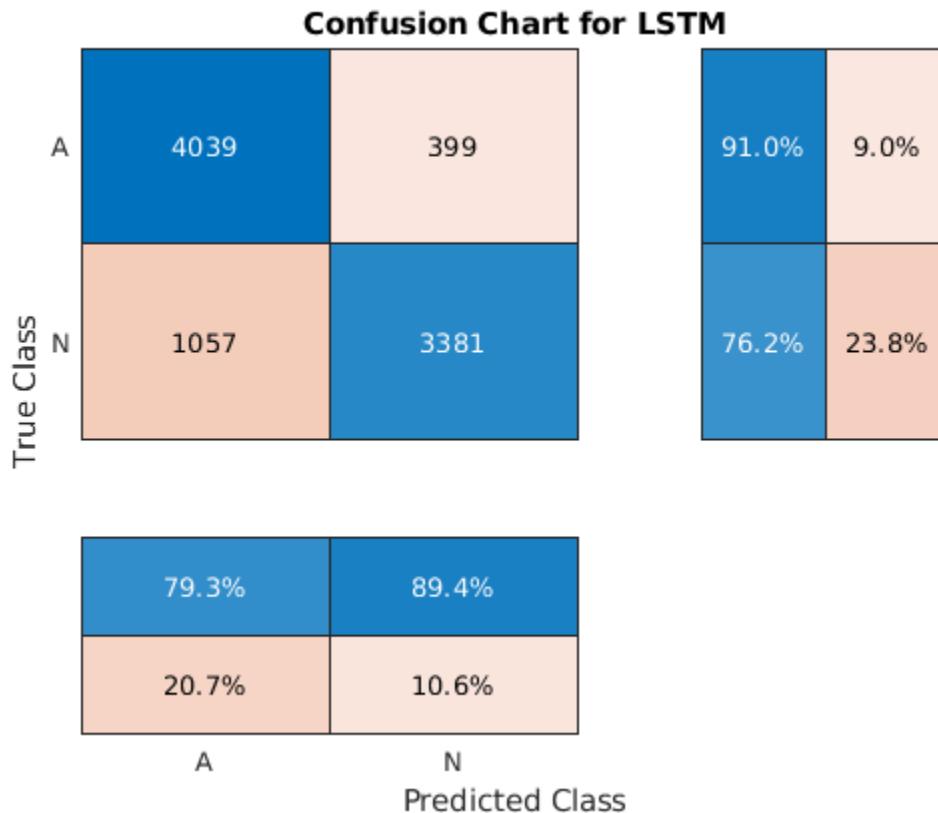
```

trainPred2 = classify(net2,XTrainSD);
LSTMAccuracy = sum(trainPred2 == YTrain)/numel(YTrain)*100

```

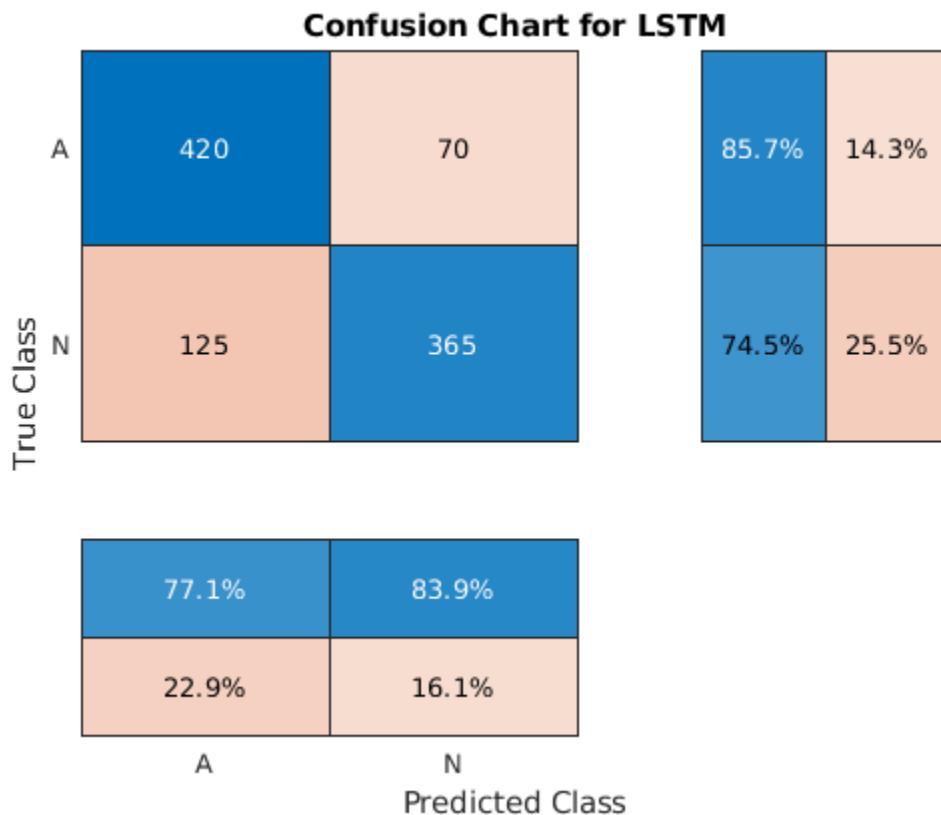
```
LSTMAccuracy = 83.5962
```

```
figure
confusionchart(YTrain,trainPred2,'ColumnSummary','column-normalized',...
    'RowSummary','row-normalized','Title','Confusion Chart for LSTM');
```



使用更新后的网络对测试数据进行分类。绘制混淆矩阵以检查测试准确度。

```
testPred2 = classify(net2,XTestSD);
LSTMAccuracy = sum(testPred2 == YTest)/numel(YTest)*100
LSTMAccuracy = 80.1020
figure
confusionchart(YTest,testPred2,'ColumnSummary','column-normalized',...
    'RowSummary','row-normalized','Title','Confusion Chart for LSTM');
```



## 结论

此示例说明如何使用 LSTM 网络构建分类器来检测 ECG 信号中的心房颤动。该过程使用过采样来避免在主要由健康被测者组成的人群中检测异常情况时出现的分类偏置。使用原始信号数据训练 LSTM 网络会导致分类准确度差。对每个信号使用两个时频矩特征来训练网络可显著提高分类性能，同时减少训练时间。

## 参考资料

- [1] AF Classification from a Short Single Lead ECG Recording: the PhysioNet/Computing in Cardiology Challenge, 2017. <https://physionet.org/challenge/2017/>
- [2] Clifford, Gari, Chengyu Liu, Benjamin Moody, Li-wei H. Lehman, Ilkaro Silva, Qiao Li, Alistair Johnson, and Roger G. Mark."AF Classification from a Short Single Lead ECG Recording:The PhysioNet Computing in Cardiology Challenge 2017."Computing in Cardiology (Rennes:IEEE).Vol. 44, 2017, pp. 1–4.
- [3] Goldberger, A. L., L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch.Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley."PhysioBank, PhysioToolkit, and PhysioNet:Components of a New Research Resource for Complex Physiologic Signals".Circulation.Vol. 101, No. 23, 13 June 2000, pp. e215–e220. <http://circ.ahajournals.org/content/101/23/e215.full>
- [4] Pons, Jordi, Thomas Lidy, and Xavier Serra."Experimenting with Musically Motivated Convolutional Neural Networks".14th International Workshop on Content-Based Multimedia Indexing (CBMI).June 2016.
- [5] Wang, D."Deep learning reinvents the hearing aid," IEEE Spectrum, Vol. 54, No. 3, March 2017, pp. 32–37. doi:10.1109/MSPEC.2017.7864754.

[6] Brownlee, Jason. How to Scale Data for Long Short-Term Memory Networks in Python. 7 July 2017. <https://machinelearningmastery.com/how-to-scale-data-for-long-short-term-memory-networks-in-python/>.

## 另请参阅

### 函数

`instfreq` | `pentropy` | `trainingOptions` | `trainNetwork` | `bilstmLayer` | `lstmLayer`

## 详细信息

- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 使用小波分析和深度学习对时序分类

此示例说明如何使用连续小波变换 (CWT) 和深度卷积神经网络 (CNN) 对人体心电图 (ECG) 信号进行分类。

从头开始训练深度 CNN 的计算成本很高，并且需要大量的训练数据。在很多应用中，并没有足够数量的训练数据可用，并且人工新建符合实际情况的训练数据也不可行。在这些情况下，利用已基于大型数据集训练的现有神经网络来完成概念相似的任务是可取的。这种对现有神经网络的利用称为迁移学习。在本示例中，我们采用两个深度 CNN（即 GoogLeNet 和 SqueezeNet，它们针对图像识别进行过预训练）基于时频表示对 ECG 波形进行分类。

GoogLeNet 和 SqueezeNet 是深度 CNN，最初是用于将图像分类至 1000 个类别。我们可重用 CNN 的网络架构，以基于时序数据的 CWT 图像对 ECG 信号进行分类。本示例中使用的数据可从 PhysioNet 公开获取。

### 数据说明

在本示例中，您使用从三组人获得的 ECG 数据：心律失常者 (ARR)、充血性心力衰竭者 (CHF) 和正常窦性心律者 (NSR)。您总共使用来自三个 PhysioNet 数据库的 162 份 ECG 录音：MIT-BIH Arrhythmia 数据库 [3][7]、MIT-BIH Normal Sinus Rhythm 数据库 [3] 和 BIDMC Congestive Heart Failure 数据库 [1][3]。更具体地说，使用了心律失常者的 96 份录音、充血性心力衰竭者的 30 份录音和正常窦性心率者的 36 份录音。目标是训练一个分类器来区分 ARR、CHF 和 NSR。

### 下载数据

第一步是从 GitHub 存储库下载数据。要从该网站下载数据，请点击 **Code**，然后选择 **Download ZIP**。将文件 **physionet\_ECG\_data-master.zip** 保存在您拥有写入权限的文件夹中。此示例的说明假设您已在 MATLAB 中将文件下载到临时目录 **tempdir** 中。如果您选择将数据下载到不同于 **tempdir** 的文件夹中，请修改后续的解压缩和加载数据的说明。

从 GitHub 下载数据后，将文件解压缩到临时目录中。

```
unzip(fullfile(tempdir,'physionet_ECG_data-master.zip'),tempdir)
```

解压缩会在您的临时目录中创建文件夹 **physionet-ECG\_data-master**。此文件夹包含文本文件 **README.md** 和 **ECGData.zip**。**ECGData.zip** 文件包含

- **ECGData.mat**
- **Modified\_physionet\_data.txt**
- **License.txt**

**ECGData.mat** 保存本示例中使用的数据。文本文件 **Modified\_physionet\_data.txt** 是 PhysioNet 的复制政策要求的文件，该文件提供数据的来源说明以及对应用于每份 ECG 录音的预处理步骤的说明。

解压缩 **physionet-ECG\_data-master** 中的 **ECGData.zip**。将数据文件加载到您的 MATLAB 工作区中。

```
unzip(fullfile(tempdir,'physionet_ECG_data-master','ECGData.zip'),...
    fullfile(tempdir,'physionet_ECG_data-master'))
load(fullfile(tempdir,'physionet_ECG_data-master','ECGData.mat'))
```

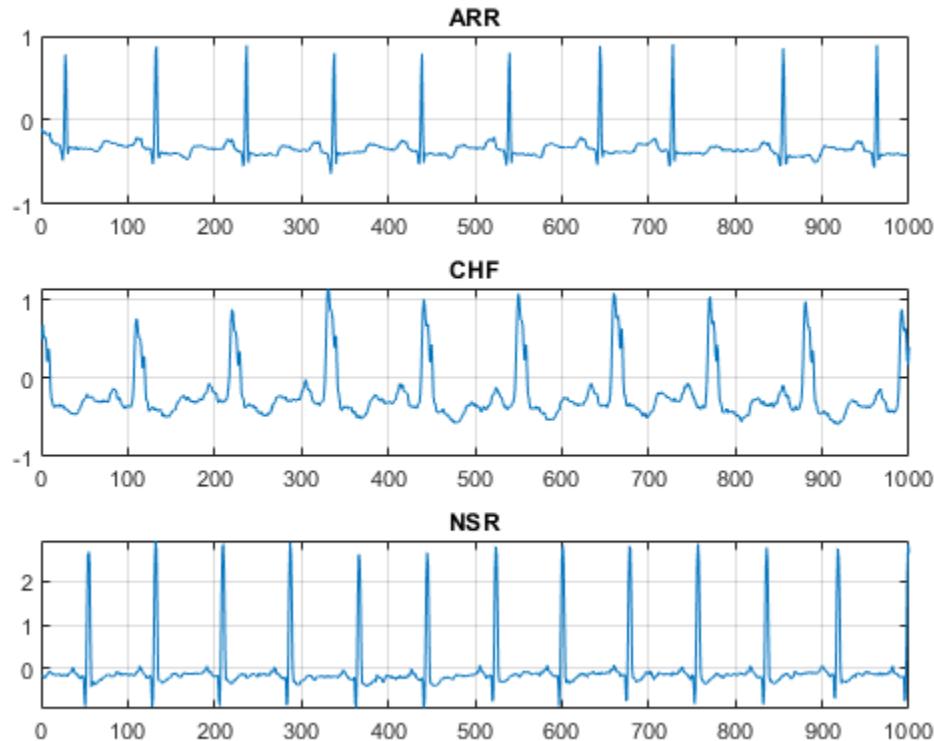
**ECGData** 是包含两个字段的结构体数组：**Data** 和 **Labels**。**Data** 字段是一个  $162 \times 65536$  矩阵，其中每行均为以 128 赫兹采样的一份 ECG 录音。**Labels** 是一个  $162 \times 1$  诊断标签元胞数组，**Data** 的每行对应一个标签。三个诊断类别是：'ARR'、'CHF' 和 'NSR'。

要存储每个类别的预处理数据，首先在 `tempdir` 内创建一个 ECG 数据目录 `dataDir`。然后在 '`data`' 中创建三个子目录，以每个 ECG 类别命名。辅助函数 `helperCreateECGDirectories` 可用于完成这一工作。`helperCreateECGDirectories` 接受 `ECGData`、ECG 数据目录的名称和父目录的名称作为输入参数。您可以用您具有写入权限的另一个目录替换 `tempdir`。此示例末尾的“支持函数”部分提供了该辅助函数的源代码。

```
parentDir = tempdir;
dataDir = 'data';
helperCreateECGDirectories(ECGData, parentDir, dataDir)
```

绘制每个 ECG 类别的表示图。辅助函数 `helperPlotReps` 用于实现此目的。`helperPlotReps` 接受 `ECGData` 作为输入。此示例末尾的“支持函数”部分提供了该辅助函数的源代码。

```
helperPlotReps(ECGData)
```



## 创建时频表示

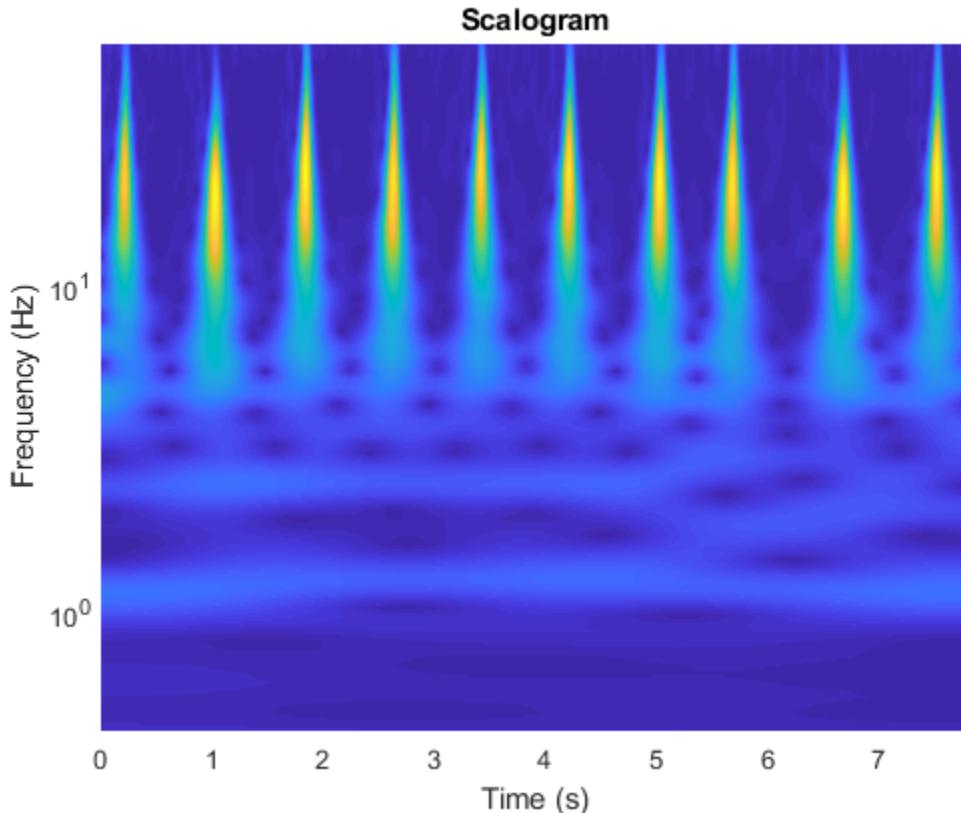
在创建文件夹后，创建 ECG 信号的时频表示。这些表示称为尺度图。尺度图是信号的 CWT 系数的绝对值。

要创建尺度图，请预先计算一个 CWT 滤波器组。当要使用相同的参数获取众多信号的 CWT 时，建议预先计算 CWT 滤波器组。

我们先尝试生成一个尺度图。使用 `cwtfilterbank` (Wavelet Toolbox) 为具有 1000 个样本的信号创建一个 CWT 滤波器组。使用滤波器组获取信号的前 1000 个样本的 CWT，并基于系数获得尺度图。

```
Fs = 128;
fb = cwtfilterbank('SignalLength',1000,...
```

```
'SamplingFrequency',Fs,...  
'VoicesPerOctave',12);  
sig = ECGData.Data(1,1:1000);  
[cfs,frq] = wt(fb,sig);  
t = (0:999)/Fs;figure;pcolor(t,frq,abs(cfs))  
set(gca,'yscale','log');shading interp;axis tight;  
title('Scalogram');xlabel('Time (s)');ylabel('Frequency (Hz)')
```



使用辅助函数 `helperCreateRGBfromTF` 将尺度图创建为 RGB 图像，并将其写入 `dataDir` 中的适当子目录。此辅助函数的源代码在此示例末尾的“支持函数”部分提供。为了与 GoogLeNet 架构兼容，每个 RGB 图像是大小为  $224 \times 224 \times 3$  的数组。

```
helperCreateRGBfromTF(ECGData,parentDir,dataDir)
```

#### 分为训练数据和验证数据

将尺度图图像加载为图像数据存储。`imageDatastore` 函数自动根据文件夹名称对图像加标签，并将数据存储为 `Image datastore` 对象。通过图像数据存储可以存储大图像数据，包括无法放入内存的数据，并在 CNN 的训练过程中高效分批读取图像。

```
allImages = imageDatastore(fullfile(parentDir,dataDir),...  
'IncludeSubfolders',true,...  
'LabelSource','foldernames');
```

将图像随机分成两组，一组用于训练，另一组用于验证。使用 80% 的图像进行训练，其余的用于验证。为了实现可再现性，我们将随机种子设置为默认值。

```

rng default
[imgsTrain, imgsValidation] = splitEachLabel(allImages, 0.8, 'randomized');
disp(['Number of training images: ', num2str(numel(imgsTrain.Files))]);

```

Number of training images: 130

```
disp(['Number of validation images: ', num2str(numel(imgsValidation.Files))]);
```

Number of validation images: 32

## GoogLeNet

### 加载

加载预训练的 GoogLeNet 神经网络。如果未安装 Deep Learning Toolbox™ Model for GoogLeNet Network 支持包，软件将在附加功能资源管理器中提供所需支持包的链接。要安装支持包，请点击链接，然后点击 **Install**。

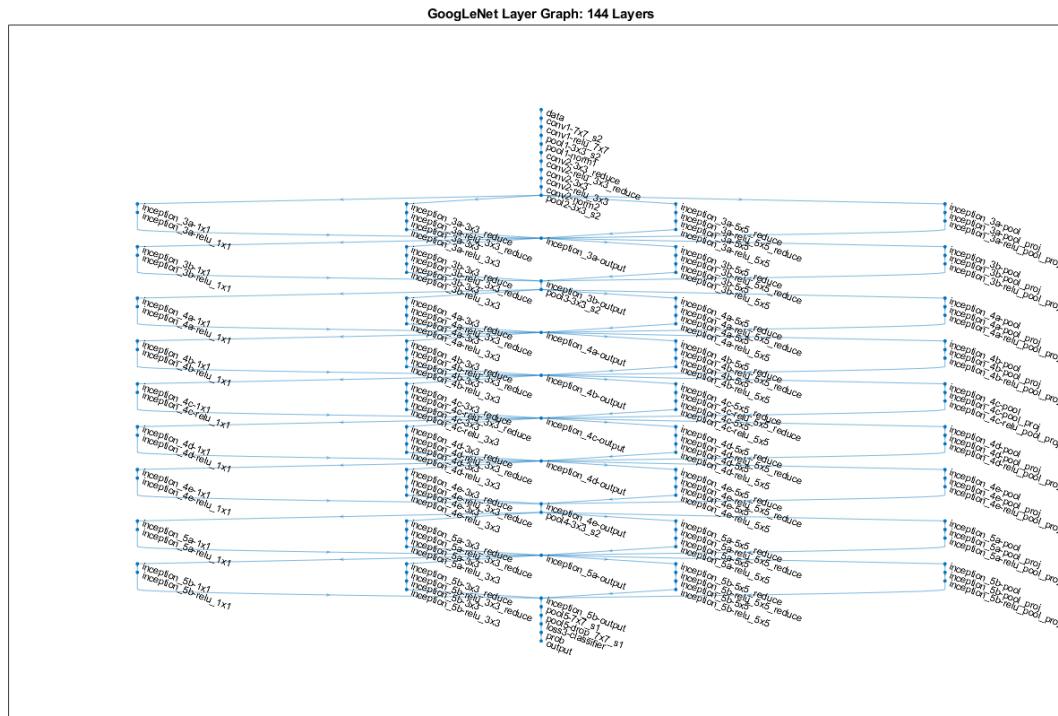
```
net = googlenet;
```

从网络中提取并显示层次图。

```

lgraph = layerGraph(net);
numberOfLayers = numel(lgraph.Layers);
figure('Units','normalized','Position',[0.1 0.1 0.8 0.8]);
plot(lgraph)
title(['GoogLeNet Layer Graph: ', num2str(numberOfLayers), ' Layers']);

```



检查网络层属性的第一个元素。确认 GoogLeNet 需要大小为  $224 \times 224 \times 3$  的 RGB 图像。

```
net.Layers(1)

ans =
  ImageInputLayer with properties:

    Name: 'data'
    InputSize: [224 224 3]

  Hyperparameters
  DataAugmentation: 'none'
  Normalization: 'zerocenter'
  Mean: [224×224×3 single]
```

### 修改 GoogLeNet 网络参数

网络架构中的每层都可以视为一个滤波器。较浅的层识别图像的更常见特征，如斑点、边缘和颜色。后续层侧重于更具体的特征，以便区分类别。GoogLeNet 经训练可将图像分类至 1000 个目标类别。对于我们的 ECG 分类问题，必须重新训练 GoogLeNet。

为防止过拟合，使用了丢弃层。丢弃层以给定的概率将输入元素随机设置为零。有关详细信息，请参阅 `dropoutLayer`。默认概率为 0.5。将网络中的最终丢弃层 '`pool5-drop_7x7_s1`' 替换为概率为 0.6 的丢弃层。

```
newDropoutLayer = dropoutLayer(0.6,'Name','new_Dropout');
lgraph = replaceLayer(lgraph,'pool5-drop_7x7_s1',newDropoutLayer);
```

网络的卷积层会提取最后一个可学习层和最终分类层用来对输入图像进行分类的图像特征。GoogLeNet 中的 '`loss3-classifier`' 和 '`output`' 这两个层包含有关如何将网络提取的特征合并为类概率、损失值和预测标签的信息。要重新训练 GoogLeNet 以对 RGB 图像进行分类，请将这两个层替换为适合数据的新层。

将全连接层 '`loss3-classifier`' 替换为新的全连接层，其中滤波器的数量等于类的数量。要使新层中的学习速度快于迁移的层，请增大全连接层的学习率因子。

```
numClasses = numel(categories(imgsTrain.Labels));
newConnectedLayer = fullyConnectedLayer(numClasses,'Name','new_fc',...
  'WeightLearnRateFactor',5,'BiasLearnRateFactor',5);
lgraph = replaceLayer(lgraph,'loss3-classifier',newConnectedLayer);
```

分类层指定网络的输出类。将分类层替换为没有类标签的新分类层。`trainNetwork` 会在训练时自动设置层的输出类。

```
newClassLayer = classificationLayer('Name','new_classoutput');
lgraph = replaceLayer(lgraph,'output',newClassLayer);
```

### 设置训练选项并训练 GoogLeNet

训练神经网络是一个使损失函数最小的迭代过程。要使损失函数最小，使用梯度下降算法。在每次迭代中，会评估损失函数的梯度并更新下降算法权重。

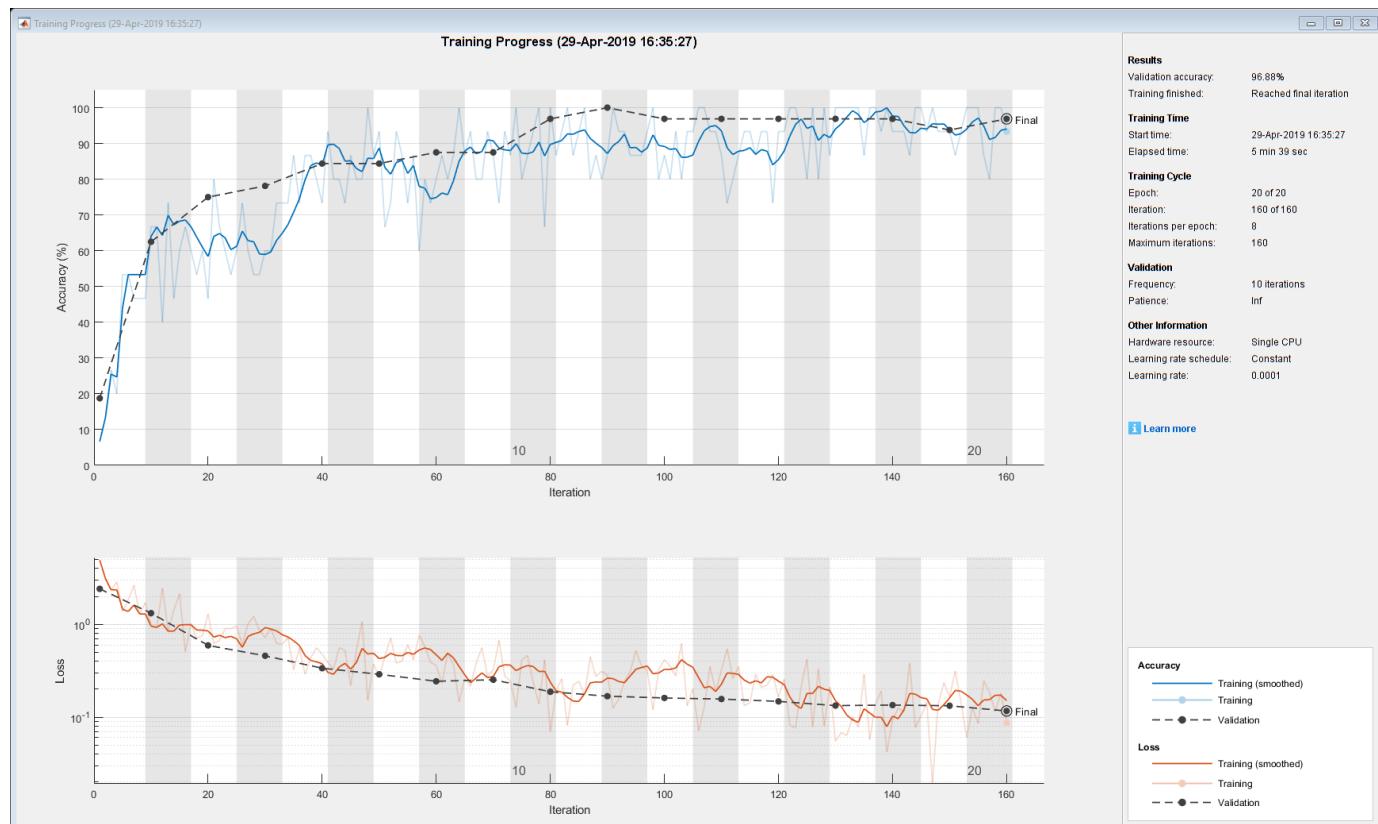
可以通过设置各种选项来调整训练。`InitialLearnRate` 指定损失函数负梯度方向的初始步长大小。`MiniBatchSize` 指定在每次迭代中使用的训练集子集的大小。一轮指对整个训练集完整运行一遍训练算法。`MaxEpochs` 指定用于训练的最大轮数。选择正确的轮数至关重要。减少轮数会导致模型欠拟合，而增加轮数会导致过拟合。

使用 `trainingOptions` 函数指定训练选项。将 `MiniBatchSize` 设置为 10, `MaxEpochs` 置为 10, `InitialLearnRate` 置为 0.0001。通过将 `Plots` 设置为 `training-progress` 来可视化训练进度。使用带动量的随机梯度下降优化器。默认情况下, 训练在 GPU (如果有) 上进行。使用 GPU 需要 Parallel Computing Toolbox™。要查看支持哪些 GPU, 请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。为了实现可再现性, 将 `ExecutionEnvironment` 设置为 `cpu`, 以便 `trainNetwork` 使用 CPU。将随机种子设置为默认值。如果您能使用 GPU, 运行速度会更快。

```
options = trainingOptions('sgdm',...
    'MiniBatchSize',15,...
    'MaxEpochs',20,...
    'InitialLearnRate',1e-4,...
    'ValidationData',imgsValidation,...
    'ValidationFrequency',10,...
    'Verbose',1,...
    'ExecutionEnvironment','cpu',...
    'Plots','training-progress');
rng default
```

训练网络。在桌面计算机 CPU 上, 训练过程通常需要 1-5 分钟。命令行窗口显示运行期间的训练信息。结果包括验证数据的轮数、迭代次数、经过的时间、小批量准确度、验证准确度和损失函数值。

```
trainedGN = trainNetwork(imgsTrain,lgraph,options);
```



Initializing input data normalization.

Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning
		(hh:mm:ss)	Accuracy	Accuracy	Loss	Loss	Rate

1	1	00:00:03	6.67%	18.75%	4.9207	2.4141	1.0000e-04
2	10	00:00:23	66.67%	62.50%	0.9589	1.3191	1.0000e-04
3	20	00:00:43	46.67%	75.00%	1.2973	0.5928	1.0000e-04
4	30	00:01:04	60.00%	78.13%	0.7219	0.4576	1.0000e-04
5	40	00:01:25	73.33%	84.38%	0.4750	0.3367	1.0000e-04
7	50	00:01:46	93.33%	84.38%	0.2714	0.2892	1.0000e-04
8	60	00:02:07	80.00%	87.50%	0.3617	0.2433	1.0000e-04
9	70	00:02:29	86.67%	87.50%	0.3246	0.2526	1.0000e-04
10	80	00:02:50	100.00%	96.88%	0.0701	0.1876	1.0000e-04
12	90	00:03:11	86.67%	100.00%	0.2836	0.1681	1.0000e-04
13	100	00:03:32	86.67%	96.88%	0.4160	0.1607	1.0000e-04
14	110	00:03:53	86.67%	96.88%	0.3237	0.1565	1.0000e-04
15	120	00:04:14	93.33%	96.88%	0.1646	0.1476	1.0000e-04
17	130	00:04:35	100.00%	96.88%	0.0551	0.1330	1.0000e-04
18	140	00:04:57	93.33%	96.88%	0.0927	0.1347	1.0000e-04
19	150	00:05:18	93.33%	93.75%	0.1666	0.1325	1.0000e-04
20	160	00:05:39	93.33%	96.88%	0.0873	0.1164	1.0000e-04

查看经过训练的网络的最后一层。确认分类输出层包括三个类。

```
trainedGN.Layers(end)
```

```
ans =
ClassificationOutputLayer with properties:

    Name: 'new_classoutput'
    Classes: [ARR CHF NSR]
    OutputSize: 3

    Hyperparameters
    LossFunction: 'crossentropyex'
```

## 评估 GoogLeNet 准确度

使用验证数据评估网络。

```
[YPred,probs] = classify(trainedGN,imgsValidation);
accuracy = mean(YPred==imgsValidation.Labels);
disp(['GoogLeNet Accuracy:',num2str(100*accuracy),'%'])
```

GoogLeNet Accuracy: 96.875%

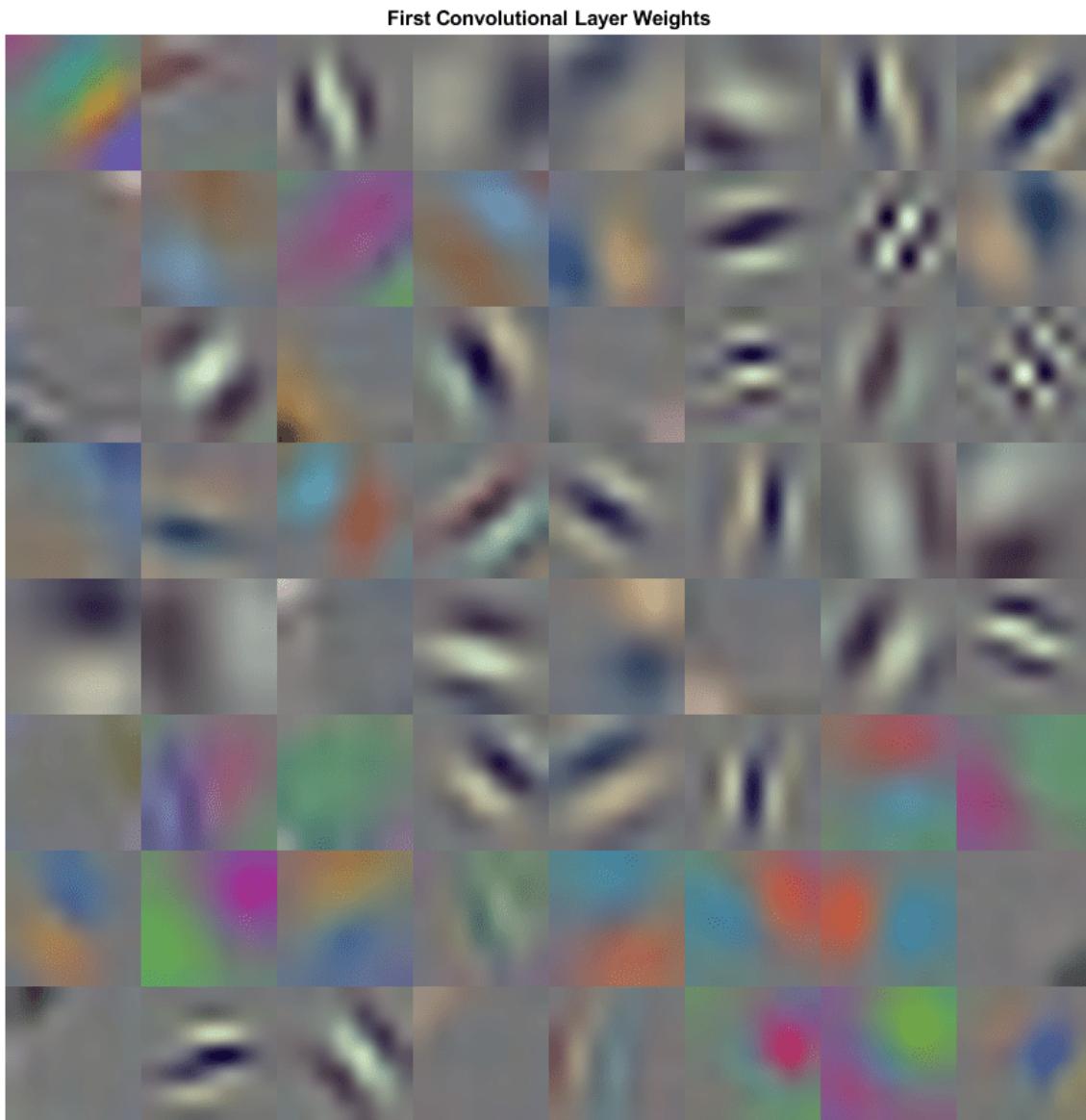
精确度与训练可视化图上报告的验证精确度相同。尺度图分成训练集合和验证集合。这两个集合都用于训练 GoogLeNet。评估训练结果的理想方法是让网络对它没有见过的数据进行分类。由于数据量不足，无法分为训练、验证和测试，我们将计算的验证准确度视为网络准确度。

## 了解 GoogLeNet 激活

CNN 的每层都对输入图像产生响应或激活。然而，一个 CNN 内只有少数几个层适合图像特征提取。网络开始的几个层捕获基本的图像特征，如边缘和斑点。要了解这一点，请可视化第一个卷积层的网络滤波器权重。第一个层有 64 组权重。

```
wghts = trainedGN.Layers(2).Weights;
wghts = rescale(wghts);
wghts = imresize(wghts,5);
figure
```

```
montage(wghts)
title('First Convolutional Layer Weights')
```



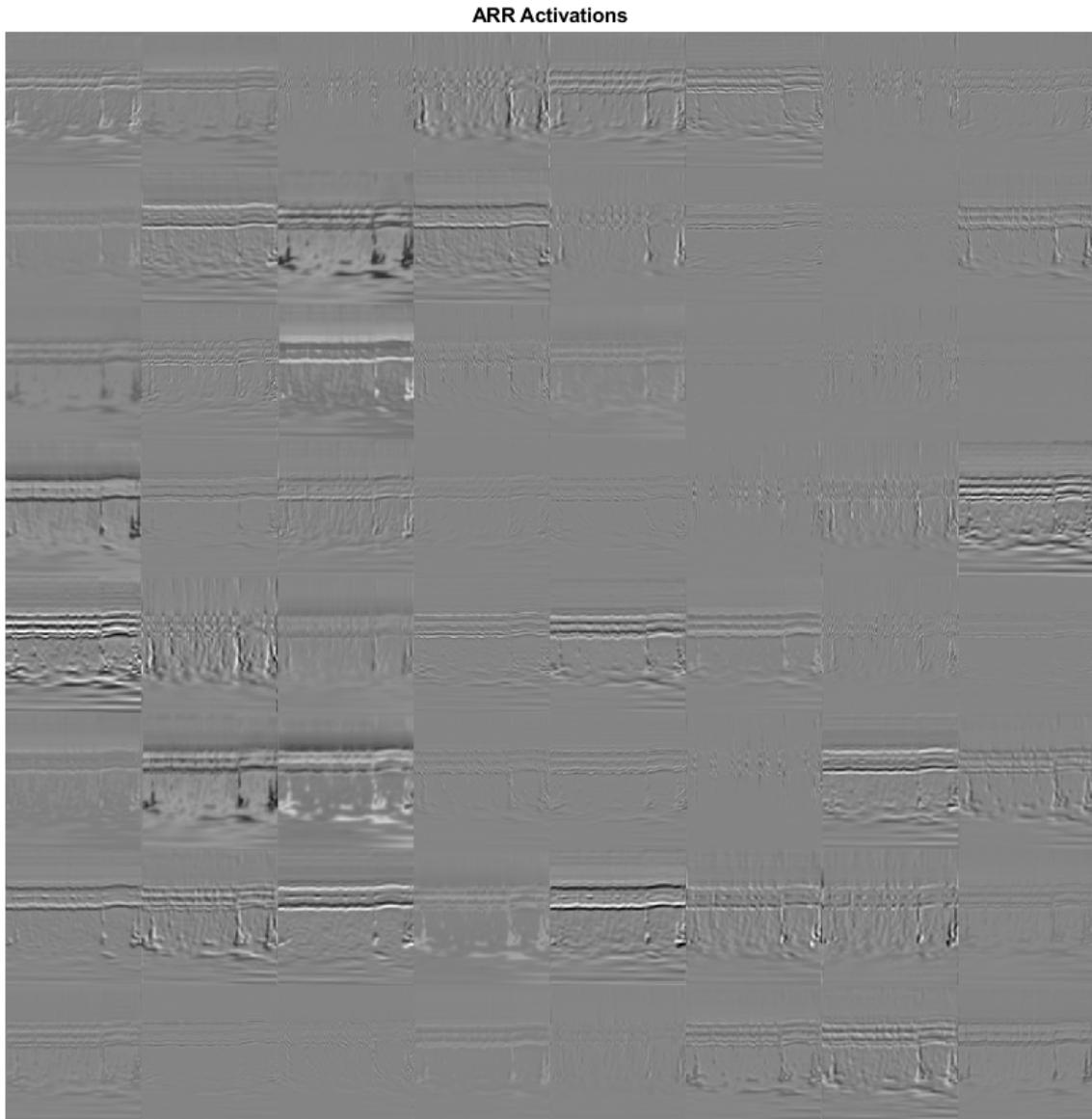
通过将激活区域与原始图像进行比较，您可以检查激活区域并发现 GoogLeNet 学习的特征。有关详细信息，请参阅“可视化卷积神经网络的激活区域”（第 5-35 页）和“可视化卷积神经网络的特征”（第 5-46 页）。

检查卷积层中的哪些区域在来自 ARR 类的图像上激活。与原始图像中的对应区域进行比较。卷积神经网络的每层由许多称为通道的二维数组组成。将网络应用于图像，并检查第一个卷积层 'conv1-7x7\_s2' 的输出激活区域。

```
convLayer = 'conv1-7x7_s2';
```

```
imgClass = 'ARR';
imgName = 'ARR_10.jpg';
imarr = imread(fullfile(parentDir,dataDir,imgClass,imgName));

trainingFeaturesARR = activations(trainedGN,imarr,convLayer);
sz = size(trainingFeaturesARR);
trainingFeaturesARR = reshape(trainingFeaturesARR,[sz(1) sz(2) 1 sz(3)]);
figure
montage(rescale(trainingFeaturesARR),'Size',[8 8])
title([imgClass,' Activations'])
```

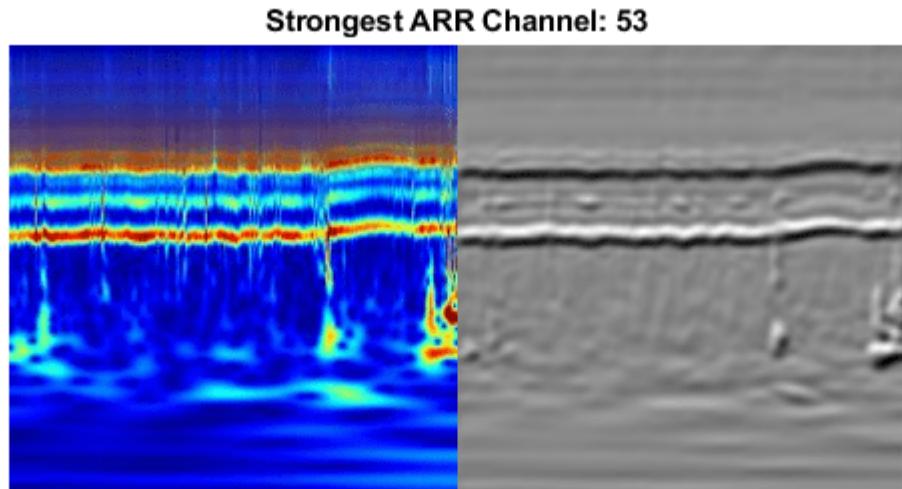


找到此图像的最强通道。将该最强通道与原始图像进行比较。

```

imgSize = size(imarr);
imgSize = imgSize(1:2);
[~,maxValueIndex] = max(max(max(trainingFeaturesARR)));
arrMax = trainingFeaturesARR(:,:,:,:maxValueIndex);
arrMax = rescale(arrMax);
arrMax = imresize(arrMax,imgSize);
figure;
imshowpair(imarr,arrMax,'montage')
title(['Strongest ',imgClass,' Channel:',num2str(maxValueIndex)])

```



## SqueezeNet

SqueezeNet 是一个深度 CNN，其架构支持大小为  $227 \times 227 \times 3$  的图像。即使 GoogLeNet 的图像大小不同，您也不必以 SqueezeNet 大小生成新 RGB 图像。您可以使用原始的 RGB 图像。

### 加载

加载预训练的 SqueezeNet 神经网络。如果未安装 Deep Learning Toolbox™ Model for SqueezeNet Network 支持包，软件将在附加功能资源管理器中提供所需支持包的链接。要安装支持包，请点击链接，然后点击 **Install**。

```
sqz = squeezenet;
```

从网络中提取层次图。确认 SqueezeNet 的层数少于 GoogLeNet。还要确认 SqueezeNet 是针对大小为  $227 \times 227 \times 3$  的图像配置的

```

lgraphSqz = layerGraph(sqz);
disp(['Number of Layers: ',num2str(numel(lgraphSqz.Layers))])

```

```
Number of Layers: 68
```

```
disp(lgraphSqz.Layers(1).InputSize)
```

```
227 227 3
```

### 修改 SqueezeNet 网络参数

要重新训练 SqueezeNet 对新图像进行分类，请进行与对 GoogLeNet 类似的更改。

检查最后六个网络层。

```
lgraphSqz.Layers(end-5:end)
```

```
ans =
6x1 Layer array with layers:
```

```
1 'drop9'           Dropout      50% dropout
2 'conv10'          Convolution  1000 1x1x512 convolutions with stride [1 1] and padding [0 0 0]
3 'relu_conv10'     ReLU        ReLU
4 'pool10'          Average Pooling 14x14 average pooling with stride [1 1] and padding [0 0 0]
5 'prob'            Softmax     softmax
6 'ClassificationLayer_predictions' Classification Output crossentropyex with 'tencn' and 999 other classes
```

将网络中的最后一个丢弃层 'drop9' 替换为概率为 0.6 的丢弃层。

```
tmpLayer = lgraphSqz.Layers(end-5);
newDropoutLayer = dropoutLayer(0.6,'Name','new_dropout');
lgraphSqz = replaceLayer(lgraphSqz,tmpLayer.Name,newDropoutLayer);
```

与 GoogLeNet 不同，SqueezeNet 中最后一个可学习层是  $1 \times 1$  卷积层 'conv10'，而不是全连接层。将 'conv10' 层替换为新的卷积层，其中滤波器的数量等于类的数量。与对 GoogLeNet 执行的操作一样，增大新层的学习率因子。

```
numClasses = numel(categories(imgsTrain.Labels));
tmpLayer = lgraphSqz.Layers(end-4);
newLearnableLayer = convolution2dLayer(1,numClasses, ...
    'Name','new_conv',...
    'WeightLearnRateFactor',10, ...
    'BiasLearnRateFactor',10);
lgraphSqz = replaceLayer(lgraphSqz,tmpLayer.Name,newLearnableLayer);
```

将分类层替换为没有类标签的新分类层。

```
tmpLayer = lgraphSqz.Layers(end);
newClassLayer = classificationLayer('Name','new_classoutput');
lgraphSqz = replaceLayer(lgraphSqz,tmpLayer.Name,newClassLayer);
```

检查网络的最后六层。确认丢弃层、卷积层和输出层已更改。

```
lgraphSqz.Layers(63:68)
```

```
ans =
6x1 Layer array with layers:
```

```
1 'new_dropout'   Dropout      60% dropout
2 'new_conv'      Convolution  3 1x1 convolutions with stride [1 1] and padding [0 0 0 0]
3 'relu_conv10'   ReLU        ReLU
4 'pool10'        Average Pooling 14x14 average pooling with stride [1 1] and padding [0 0 0 0]
5 'prob'          Softmax     softmax
6 'new_classoutput' Classification Output crossentropyex
```

### 为 SqueezeNet 准备 RGB 数据

RGB 图像具有适合 GoogLeNet 架构的大小。创建增强的图像数据存储，这些数据存储会自动为 SqueezeNet 架构调整现有 RGB 图像的大小。有关详细信息，请参阅 [augmentedImageDatastore](#)。

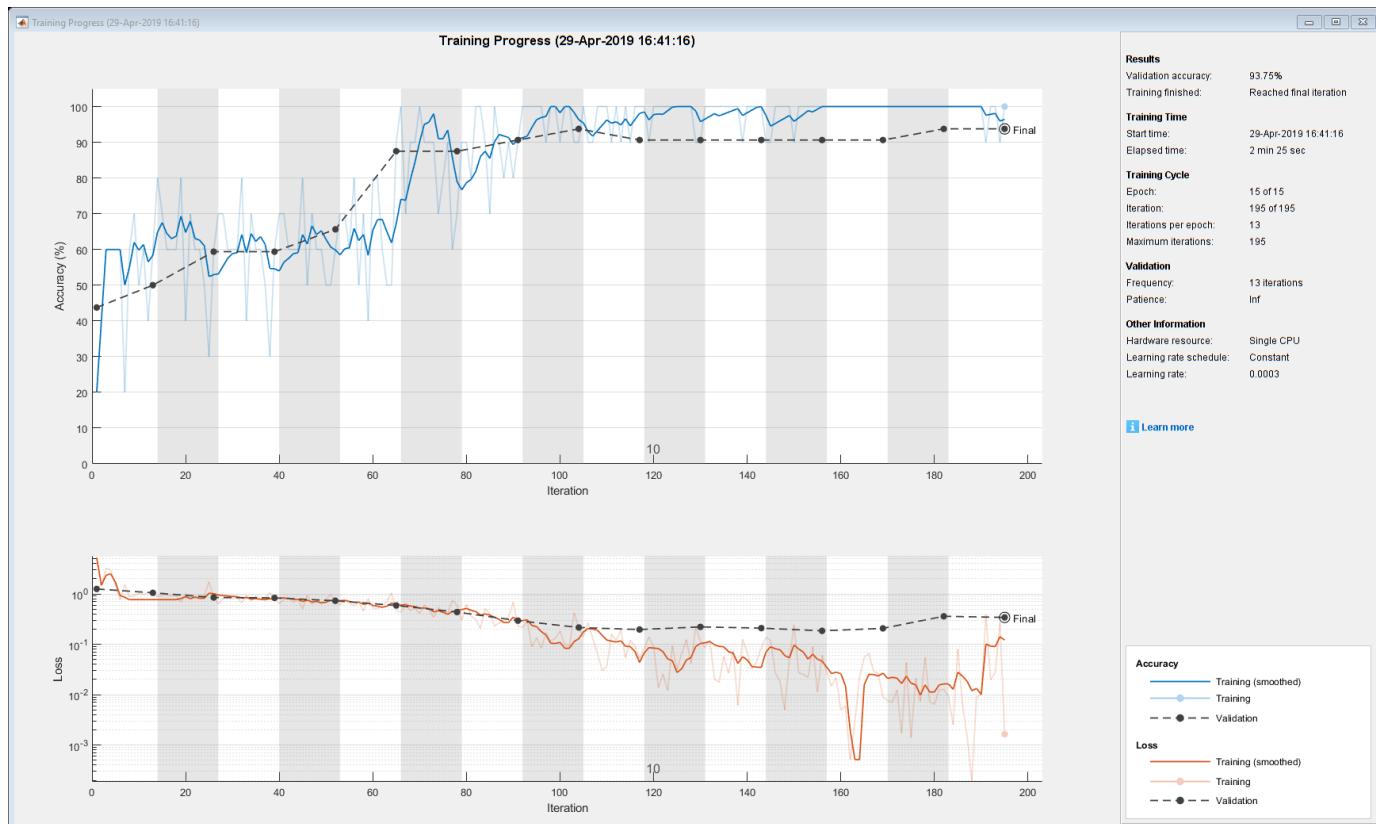
```
augimgsTrain = augmentedImageDatastore([227 227], imgsTrain);
augimgsValidation = augmentedImageDatastore([227 227], imgsValidation);
```

### 设置训练选项并训练 SqueezeNet

创建一组新的用于 SqueezeNet 的训练选项。将随机种子设置为默认值并训练网络。在桌面计算机 CPU 上，训练过程通常需要 1-5 分钟。

```
ilr = 3e-4;
miniBatchSize = 10;
maxEpochs = 15;
valFreq = floor(numel(augimgsTrain.Files)/miniBatchSize);
opts = trainingOptions('sgdm',...
    'MiniBatchSize',miniBatchSize,...
    'MaxEpochs',maxEpochs,...
    'InitialLearnRate',ilr,...
    'ValidationData',augimgsValidation,...
    'ValidationFrequency',valFreq,...
    'Verbose',1,...
    'ExecutionEnvironment','cpu',...
    'Plots','training-progress');

rng default
trainedSN = trainNetwork(augimgsTrain,lgraphSqz,opts);
```



Initializing input data normalization.

Epoch	Iteration	Time Elapsed	Mini-batch	Validation	Mini-batch	Validation	Base Learning
-------	-----------	--------------	------------	------------	------------	------------	---------------

		(hh:mm:ss)	Accuracy	Accuracy	Loss	Loss	Rate	
1	1	00:00:01	20.00%	43.75%	5.2508	1.2540	0.0003	
1	13	00:00:11	60.00%	50.00%	0.9912	1.0519	0.0003	
2	26	00:00:20	60.00%	59.38%	0.8554	0.8497	0.0003	
3	39	00:00:30	60.00%	59.38%	0.8120	0.8328	0.0003	
4	50	00:00:38	50.00%		0.7885		0.0003	
4	52	00:00:40	60.00%	65.63%	0.7091	0.7314	0.0003	
5	65	00:00:49	90.00%	87.50%	0.4639	0.5893	0.0003	
6	78	00:00:59	70.00%	87.50%	0.6021	0.4355	0.0003	
7	91	00:01:08	90.00%	90.63%	0.2307	0.2945	0.0003	
8	100	00:01:15	90.00%		0.1827		0.0003	
8	104	00:01:18	90.00%	93.75%	0.2139	0.2153	0.0003	
9	117	00:01:28	100.00%	90.63%	0.0521	0.1964	0.0003	
10	130	00:01:38	90.00%	90.63%	0.1134	0.2214	0.0003	
11	143	00:01:47	100.00%	90.63%	0.0855	0.2095	0.0003	
12	150	00:01:52	90.00%		0.2394		0.0003	
12	156	00:01:57	100.00%	90.63%	0.0606	0.1849	0.0003	
13	169	00:02:06	100.00%	90.63%	0.0090	0.2071	0.0003	
14	182	00:02:16	100.00%	93.75%	0.0127	0.3597	0.0003	
15	195	00:02:25	100.00%	93.75%	0.0016	0.3414	0.0003	

检查网络的最后一层。确认分类输出层包括三个类。

`trainedSN.Layers(end)`

```
ans =
ClassificationOutputLayer with properties:

    Name: 'new_classoutput'
    Classes: [ARR  CHF  NSR]
    OutputSize: 3

    Hyperparameters
    LossFunction: 'crossentropyex'
```

## 评估 SqueezeNet 准确度

使用验证数据评估网络。

```
[YPred,probs] = classify(trainedSN,augimgsValidation);
accuracy = mean(YPred==imgsValidation.Labels);
disp(['SqueezeNet Accuracy:',num2str(100*accuracy),'%'])
```

SqueezeNet Accuracy: 93.75%

## 结论

此示例说明如何利用预训练的 CNN (GoogLeNet 和 SqueezeNet) 使用迁移学习和连续小波分析对三类 ECG 信号进行分类。ECG 信号的基于小波的时频表示用于创建尺度图。示例生成了尺度图的 RGB 图像。这些图像用于微调这两个深度 CNN。示例还探讨了不同网络层的激活。

利用预训练的 CNN 模型对信号进行分类的方法众多，本示例中介绍的只是其中的一种。也可以使用其他工作流。“Deploy Signal Classifier on NVIDIA Jetson Using Wavelet Analysis and Deep Learning” (Wavelet Toolbox) 和 “Deploy Signal Classifier Using Wavelets and Deep Learning on Raspberry Pi” (Wavelet Toolbox) 介绍了如何将代码部署到硬件上进行信号分类。GoogLeNet 和 SqueezeNet 是

在 ImageNet 数据库 [10] 子集上预训练的模型，用于 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [8] 中。ImageNet 集合包含真实世界物品的图像，例如鱼、鸟、设备和真菌。尺度图不属于现实世界物品的类。为了适应 GoogLeNet 和 SqueezeNet 架构，还对尺度图进行了数据缩减。除了通过微调预训练的 CNN 来对尺度图分类外，也可以选择基于原始尺度图大小从头开始训练 CNN。

## 参考资料

- 1 Baim, D. S., W. S. Colucci, E. S. Monrad, H. S. Smith, R. F. Wright, A. Lanoue, D. F. Gauthier, B. J. Ransil, W. Grossman, and E. Braunwald."Survival of patients with severe congestive heart failure treated with oral milrinone."Journal of the American College of Cardiology.Vol. 7, Number 3, 1986, pp. 661–670.
- 2 Engin, M."ECG beat classification using neuro-fuzzy network."Pattern Recognition Letters.Vol. 25, Number 15, 2004, pp.1715–1722.
- 3 Goldberger A. L., L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch.Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley."PhysioBank, PhysioToolkit, and PhysioNet:Components of a New Research Resource for Complex Physiologic Signals."Circulation.Vol. 101, Number 23: e215–e220. [Circulation Electronic Pages; <http://circ.ahajournals.org/content/101/23/e215.full>]; 2000 (June 13). doi:10.1161/01.CIR.101.23.e215.
- 4 Leonarduzzi, R. F., G. Schlotthauer, and M. E. Torres."Wavelet leader based multifractal analysis of heart rate variability during myocardial ischaemia."In Engineering in Medicine and Biology Society (EMBC), Annual International Conference of the IEEE, 110–113.Buenos Aires, Argentina:IEEE, 2010.
- 5 Li, T., and M. Zhou."ECG classification using wavelet packet entropy and random forests."Entropy.Vol. 18, Number 8, 2016, p.285.
- 6 Maharaj, E. A., and A. M. Alonso."Discriminant analysis of multivariate time series:Application to diagnosis based on ECG signals."Computational Statistics and Data Analysis.Vol. 70, 2014, pp. 67–87.
- 7 Moody, G. B., and R. G. Mark."The impact of the MIT-BIH Arrhythmia Database."IEEE Engineering in Medicine and Biology Magazine.Vol. 20.Number 3, May-June 2001, pp. 45–50. (PMID: 11446209)
- 8 Russakovsky, O., J. Deng, and H. Su et al."ImageNet Large Scale Visual Recognition Challenge."International Journal of Computer Vision.Vol. 115, Number 3, 2015, pp. 211–252.
- 9 Zhao, Q., and L. Zhang."ECG feature extraction and classification using wavelet transform and support vector machines."In IEEE International Conference on Neural Networks and Brain, 1089–1092.Beijing, China:IEEE, 2005.
- 10 ImageNet. <http://www.image-net.org>

## 支持函数

**helperCreateECGDataDirectories** 在父目录中创建一个数据目录，然后在该数据目录中创建三个子目录。子目录以 ECGData 中发现的 ECG 信号的每个类命名。

```
function helperCreateECGDirectories(ECGData,parentFolder,dataFolder)
% This function is only intended to support the ECGAndDeepLearningExample.
% It may change or be removed in a future release.
```

```
rootFolder = parentFolder;
localFolder = dataFolder;
mkdir(fullfile(rootFolder,localFolder))
```

```

folderLabels = unique(ECGData.Labels);
for i = 1:numel(folderLabels)
    mkdir(fullfile(rootFolder,localFolder,char(folderLabels(i))));
end
end

```

**helperPlotReps** 绘制在 ECGData 中发现的 ECG 信号的每个类表示的前一千个样本。

```

function helperPlotReps(ECGData)
% This function is only intended to support the ECGAndDeepLearningExample.
% It may change or be removed in a future release.

folderLabels = unique(ECGData.Labels);

for k=1:3
    ecgType = folderLabels{k};
    ind = find(ismember(ECGData.Labels,ecgType));
    subplot(3,1,k)
    plot(ECGData.Data(ind(1),1:1000));
    grid on
    title(ecgType)
end
end

```

**helperCreateRGBfromTF** 使用 **cwtfilterbank** (Wavelet Toolbox) 获得 ECG 信号的连续小波变换，并根据小波系数生成尺度图。辅助函数调整尺度图的大小，并将它们作为 jpeg 图像写入磁盘。

```

function helperCreateRGBfromTF(ECGData,parentFolder,childFolder)
% This function is only intended to support the ECGAndDeepLearningExample.
% It may change or be removed in a future release.

imageRoot = fullfile(parentFolder,childFolder);

data = ECGData.Data;
labels = ECGData.Labels;

 $[\sim, \text{signalLength}] = \text{size}(\text{data});$ 

fb = cwtfilterbank('SignalLength', signalLength, 'VoicesPerOctave', 12);
r = size(data,1);

for ii = 1:r
    cfs = abs(fb.wt(data(ii,:)));
    im = ind2rgb(im2uint8(rescale(cfs)),jet(128));

    imgLoc = fullfile(imageRoot,char(labels(ii)));
    imFileName = strcat(char(labels(ii)), '_', num2str(ii), '.jpg');
    imwrite(imresize(im,[224 224]),fullfile(imgLoc,imFileName));
end
end

```

## 另请参阅

[cwtfilterbank](#) | [googlenet](#) | [squeezeNet](#) | [trainNetwork](#) | [trainingOptions](#) | [imageDatastore](#) | [augmentedImageDatastore](#)

## 相关示例

- “训练深度学习网络以对新图像进行分类” (第 3-6 页)
- “预训练的深度神经网络” (第 1-8 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)



# 无线通信示例

---

## 使用深度学习进行调制分类

此示例说明如何使用卷积神经网络 (CNN) 进行调制分类。您将生成合成的、通道减损波形。使用生成的波形作为训练数据，训练 CNN 进行调制分类。然后用软件定义的无线电 (SDR) 硬件和无线信号测试 CNN。

### 使用 CNN 预测调制类型

本示例中经过训练的 CNN 可识别以下八种数字调制类型和三种模拟调制类型：

- 二相相移键控 (BPSK)
- 四相相移键控 (QPSK)
- 八相相移键控 (8-PSK)
- 十六相正交幅值调制 (16-QAM)
- 六十四相正交幅值调制 (64-QAM)
- 四相脉冲幅值调制 (PAM4)
- 高斯频移键控 (GFSK)
- 连续相位频移键控 (CPFSK)
- 广播 FM (B-FM)
- 双边带幅值调制 (DSB-AM)
- 单边带幅值调制 (SSB-AM)

```
modulationTypes = categorical(["BPSK", "QPSK", "8PSK", ...
    "16QAM", "64QAM", "PAM4", "GFSK", "CPFSK", ...
    "B-FM", "DSB-AM", "SSB-AM"]);
```

首先，加载经过训练的网络。有关网络训练的详细信息，请参阅“训练 CNN (第 13-0 页)”一节。

```
load trainedModulationClassificationNetwork
trainedNet

trainedNet =
SeriesNetwork with properties:

    Layers: [28×1 nnet.cnn.layer.Layer]
    InputNames: {'Input Layer'}
    OutputNames: {'Output'}
```

经过训练的 CNN 接受 1024 个通道减损采样，并预测每个帧的调制类型。生成几个因莱斯多径衰落、中心频率和采样时间漂移以及 AWGN 而有所减损的 PAM4 帧。使用以下函数生成合成信号来测试 CNN。然后使用 CNN 预测帧的调制类型。

- **randi**: 生成随机位
- **pammod** (Communications Toolbox): PAM4 调制位
- **rcosdesign** (Signal Processing Toolbox): 设计平方根升余弦脉冲整形滤波器
- **filter**: 脉冲确定符号的形状
- **comm.RicianChannel** (Communications Toolbox): 应用莱斯多径通道
- **comm.PhaseFrequencyOffset** (Communications Toolbox): 应用时钟偏移引起的相位和/或频率偏移

- **interp1**: 应用时钟偏移引起的计时漂移
- **awgn** (Communications Toolbox): 添加 AWGN

```
% Set the random number generator to a known state to be able to regenerate
% the same frames every time the simulation is run
rng(123456)
% Random bits
d = randi([0 3], 1024, 1);
% PAM4 modulation
syms = pammod(d,4);
% Square-root raised cosine filter
filterCoeffs = rcosdesign(0.35,4,8);
tx = filter(filterCoeffs,1,upsample(syms,8));

% Channel
SNR = 30;
maxOffset = 5;
fc = 902e6;
fs = 200e3;
multipathChannel = comm.RicianChannel(...%
    'SampleRate', fs, ...
    'PathDelays', [0 1.8 3.4] / 200e3, ...
    'AveragePathGains', [0 -2 -10], ...
    'KFactor', 4, ...
    'MaximumDopplerShift', 4);

frequencyShifter = comm.PhaseFrequencyOffset(...%
    'SampleRate', fs);

% Apply an independent multipath channel
reset(multipathChannel)
outMultipathChan = multipathChannel(tx);

% Determine clock offset factor
clockOffset = (rand() * 2*maxOffset) - maxOffset;
C = 1 + clockOffset / 1e6;

% Add frequency offset
frequencyShifter.FrequencyOffset = -(C-1)*fc;
outFreqShifter = frequencyShifter(outMultipathChan);

% Add sampling time drift
t = (0:length(tx)-1)' / fs;
newFs = fs * C;
tp = (0:length(tx)-1)' / newFs;
outTimeDrift = interp1(t, outFreqShifter, tp);

% Add noise
rx = awgn(outTimeDrift,SNR,0);

% Frame generation for classification
unknownFrames = helperModClassGetNNFrames(rx);

% Classification
[prediction1,score1] = classify(trainedNet,unknownFrames);
```

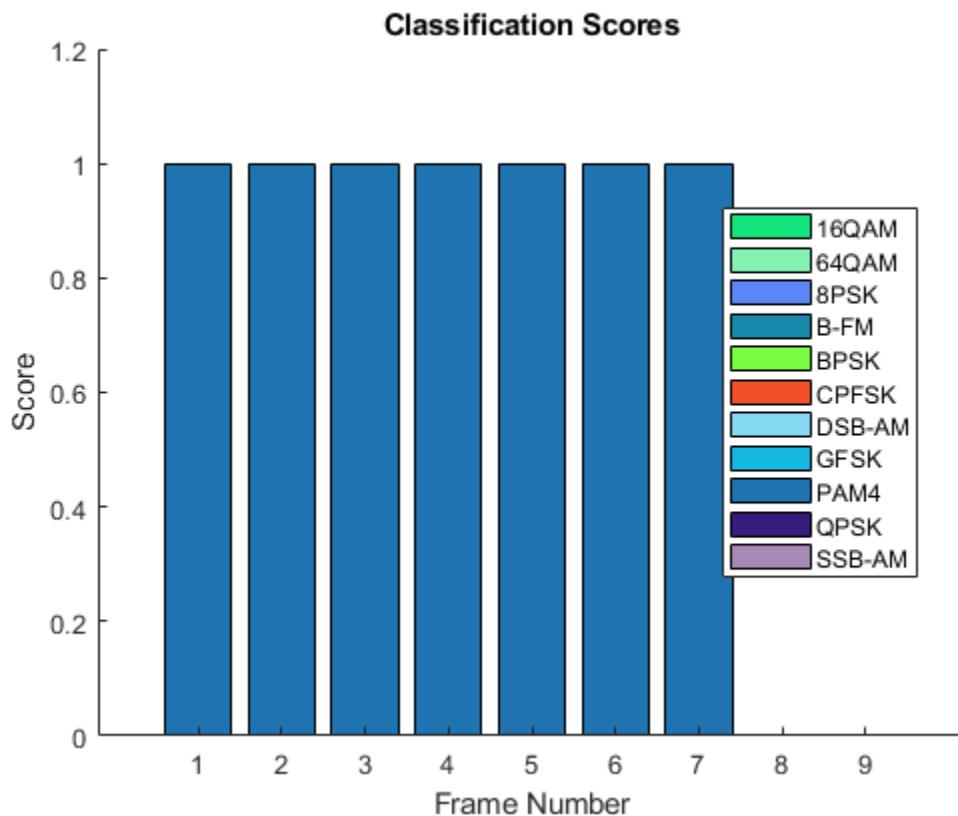
返回分类器预测，这类似于硬判决。网络将帧正确识别为 PAM4 帧。有关生成调制信号的详细信息，请参阅 `helperModClassGetModulator` 函数。

`prediction1`

```
prediction1 = 7×1 categorical
PAM4
PAM4
PAM4
PAM4
PAM4
PAM4
PAM4
```

分类器还返回一个包含每一帧分数的向量。分数对应于每个帧具有预测的调制类型的概率。绘制分数图。

`helperModClassPlotScores(score1,modulationTypes)`



我们首先需要用已知（即已加标签的）数据训练 CNN，然后才能使用 CNN 进行调制分类或执行任何其他任务。此示例的第一部分说明如何使用 Communications Toolbox 功能（如调制器、滤波器和通道减损）来生成合成的训练数据。第二部分着重于针对调制分类任务来定义、训练和测试 CNN。第三部分通过软件定义无线电 (SDR) 平台使用无线信号来测试网络性能。

### 生成用于训练的波形

为每种调制类型生成 10000 个帧，其中 80% 用于训练，10% 用于验证，10% 用于测试。我们在网络训练阶段使用训练和验证帧。使用测试帧获得最终分类准确度。每帧的长度为 1024 个样本，采样率为 200

kHz。对于数字调制类型，八个采样表示一个符号。网络根据单个帧而不是多个连续帧（如视频）作出每个决定。假设数字和模拟调制类型的中心频率分别为 902 MHz 和 100 MHz。

要快速运行此示例，请使用经过训练的网络并生成少量训练帧。要在您的计算机上训练网络，请选择“Train network now”选项（即，将 trainNow 设置为 true）。

```
trainNow = Use trained network; % Set training mode
if trainNow == true
    numFramesPerModType = 10000;
else
    numFramesPerModType = 500;
end
percentTrainingSamples = 80;
percentValidationSamples = 10;
percentTestSamples = 10;

sps = 8; % Samples per symbol
spf = 1024; % Samples per frame
symbolsPerFrame = spf / sps;
fs = 200e3; % Sample rate
fc = [902e6 100e6]; % Center frequencies
```

## 创建通道减损

让每帧通过通道并具有

- AWGN
- 莱斯多径衰落
- 时钟偏移，导致中心频率偏移和采样时间漂移

由于本示例中的网络基于单个帧作出决定，因此每个帧必须通过独立的通道。

### AWGN

通道增加 SNR 为 30 dB 的 AWGN。使用 **awgn** (Communications Toolbox) 函数实现通道。

### 莱斯多径

通道使用 **comm.RicianChannel** (Communications Toolbox) System object 通过莱斯多径衰落通道传递信号。假设延迟分布为 [0 1.8 3.4] 个样本，对应的平均路径增益为 [0 -2 -10] dB。K 因子为 4，最大多普勒频移为 4 Hz，等效于 902 MHz 的步行速度。使用以下设置实现通道。

### 时钟偏移

时钟偏移是发送器和接收器的内部时钟源不准确造成的。时钟偏移导致中心频率（用于将信号下变频至基带）和数模转换器采样率不同于理想值。通道仿真器使用时钟偏移因子  $C$ ，表示为  $C = 1 + \frac{\Delta_{clock}}{10^6}$ ，其中  $\Delta_{clock}$  是时钟偏移。对于每个帧，通道基于  $[-\max\Delta_{clock} \max\Delta_{clock}]$  范围内一组均匀分布的值生成一个随机  $\Delta_{clock}$  值，其中  $\max\Delta_{clock}$  是最大时钟偏移。时钟偏移以百万分率 (ppm) 为单位测量。对于本示例，假设最大时钟偏移为 5 ppm。

```
maxDeltaOff = 5;
deltaOff = (rand()*2*maxDeltaOff) - maxDeltaOff;
C = 1 + (deltaOff/1e6);
```

## 频率偏移

基于时钟偏移因子  $C$  和中心频率，对每帧进行频率偏移。使用 **comm.PhaseFrequencyOffset** (Communications Toolbox) 实现通道。

## 采样率偏移

基于时钟偏移因子  $C$ ，对每帧进行采样率偏移。使用 **interp1** 函数实现通道，以  $C \times f_s$  的新速率对帧进行重新采样。

## 合并后的通道

使用 **helperModClassTestChannel** 对象对帧应用所有三种通道减损。

```
channel = helperModClassTestChannel(...  
    'SampleRate', fs, ...  
    'SNR', SNR, ...  
    'PathDelays', [0 1.8 3.4] / fs, ...  
    'AveragePathGains', [0 -2 -10], ...  
    'KFactor', 4, ...  
    'MaximumDopplerShift', 4, ...  
    'MaximumClockOffset', 5, ...  
    'CenterFrequency', 902e6)  
  
channel =  
    helperModClassTestChannel with properties:  
  
        SNR: 30  
        CenterFrequency: 902000000  
        SampleRate: 200000  
        PathDelays: [0 9.0000e-06 1.7000e-05]  
        AveragePathGains: [0 -2 -10]  
        KFactor: 4  
        MaximumDopplerShift: 4  
        MaximumClockOffset: 5
```

您可以使用 **info** 对象函数查看有关通道的基本信息。

```
chInfo = info(channel)  
  
chInfo = struct with fields:  
    ChannelDelay: 6  
    MaximumFrequencyOffset: 4510  
    MaximumSampleRateOffset: 1
```

## 波形生成

创建一个循环，它为每种调制类型生成通道减损的帧并将这些帧及其对应标签存储在 MAT 文件中。通过将数据保存到文件中，您无需每次运行此示例时都生成数据。您还可以更高效地共享数据。

从每帧的开头删除随机数量的样本，以去除瞬变并确保帧相对于符号边界具有随机起点。

```
% Set the random number generator to a known state to be able to regenerate  
% the same frames every time the simulation is run  
rng(1235)  
tic
```

```

numModulationTypes = length(modulationTypes);

channelInfo = info(channel);
transDelay = 50;
dataDirectory = fullfile(tempdir,"ModClassDataFiles");
disp("Data file directory is " + dataDirectory)

Data file directory is C:\Users\esoyer\AppData\Local\Temp\ModClassDataFiles

fileNameRoot = "frame";

% Check if data files exist
dataFilesExist = false;
if exist(dataDirectory,'dir')
    files = dir(fullfile(dataDirectory,sprintf("%s*",fileNameRoot)));
    if length(files) == numModulationTypes*numFramesPerModType
        dataFilesExist = true;
    end
end

if ~dataFilesExist
    disp("Generating data and saving in data files...")
    [success,msg,msgID] = mkdir(dataDirectory);
    if ~success
        error(msgID,msg)
    end
    for modType = 1:numModulationTypes
        fprintf("%s - Generating %s frames\n", ...
            datestr(toc/86400,'HH:MM:SS'), modulationTypes(modType))

        label = modulationTypes(modType);
        numSymbols = (numFramesPerModType / sps);
        dataSrc = helperModClassGetSource(modulationTypes(modType), sps, 2*spf, fs);
        modulator = helperModClassGetModulator(modulationTypes(modType), sps, fs);
        if contains(char(modulationTypes(modType)), {'B-FM','DSB-AM','SSB-AM'})
            % Analog modulation types use a center frequency of 100 MHz
            channel.CenterFrequency = 100e6;
        else
            % Digital modulation types use a center frequency of 902 MHz
            channel.CenterFrequency = 902e6;
        end

        for p=1:numFramesPerModType
            % Generate random data
            x = dataSrc();

            % Modulate
            y = modulator(x);

            % Pass through independent channels
            rxSamples = channel(y);

            % Remove transients from the beginning, trim to size, and normalize
            frame = helperModClassFrameGenerator(rxSamples, spf, spf, transDelay, sps);

            % Save data file
            fileName = fullfile(dataDirectory,...
```

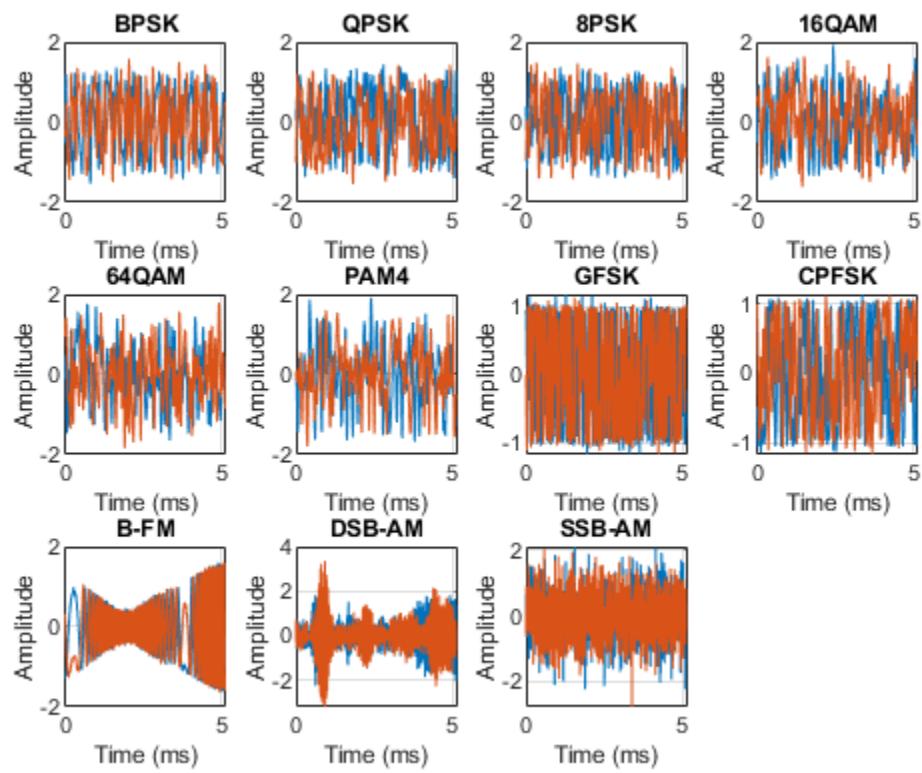
```

sprintf("%s%s%03d",fileNameRoot,modulationTypes(modType),p));
save(fileName,"frame","label")
end
end
else
    disp("Data files exist. Skip data generation.")
end

Data files exist. Skip data generation.

% Plot the amplitude of the real and imaginary parts of the example frames
% against the sample number
helperModClassPlotTimeDomain(dataDirectory,modulationTypes,fs)

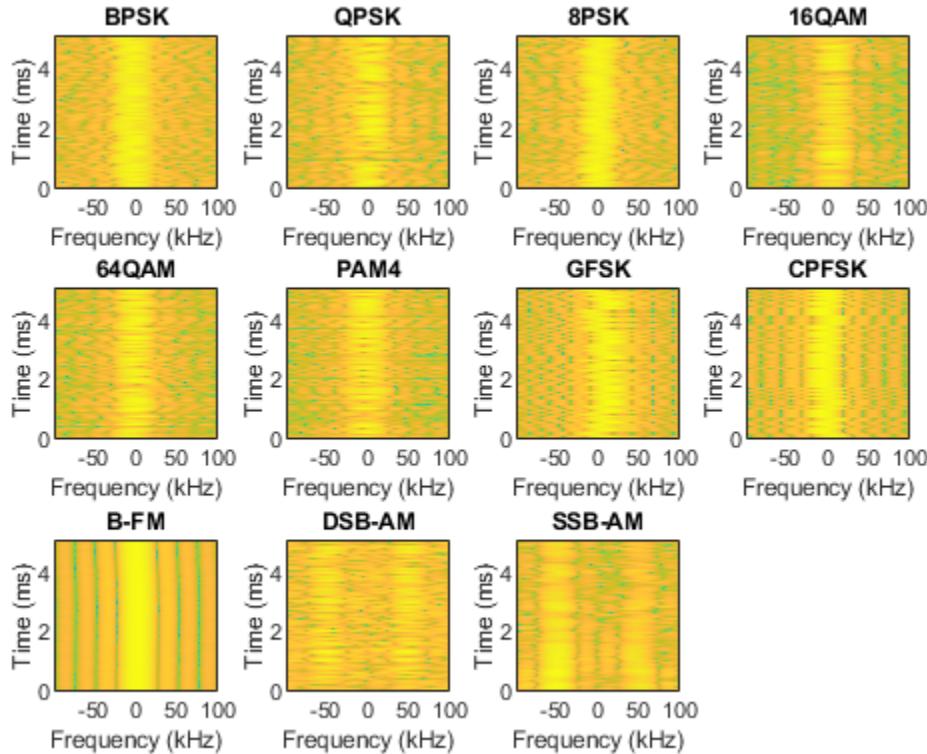
```



```

% Plot the spectrogram of the example frames
helperModClassPlotSpectrogram(dataDirectory,modulationTypes,fs,sps)

```



## 创建数据存储

使用 `signalDatastore` 对象来管理包含生成的复杂波形的文件。如果每个文件可以单独放入内存中，但整个集合不一定能放入内存，则数据存储特别有用。

```
frameDS = signalDatastore(dataDirectory,'SignalVariableNames',[ "frame", "label"]);
```

## 将复信号变换为实数数组

此示例中的深度学习网络需要实数输入，而接收的信号包含复数基带采样。将复信号变换为实数值四维数组。输出帧的大小为  $1 \times \text{spf} \times 2 \times N$ ，其中第一页（第三个维度）是同相采样，第二页是正交采样。当卷积滤波器的大小为  $1 \times \text{spf}$  时，这种方法可确保混合 I 和 Q（甚至是卷积层）中的信息，使我们能够更好地利用相位信息。有关详细信息，请参阅 `helperModClassIQAsPages`。

```
frameDSTrans = transform(frameDS,@helperModClassIQAsPages);
```

## 拆分为训练、验证和测试

接下来，将帧分为训练数据、验证数据和测试数据。有关详细信息，请参阅 `helperModClassSplitData`。

```
splitPercentages = [percentTrainingSamples, percentValidationSamples, percentTestSamples];
[trainDSTrans, validDSTrans, testDSTrans] = helperModClassSplitData(frameDSTrans, splitPercentages);
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

```
Evaluating tall expression using the Parallel Pool 'local':
- Pass 1 of 2: Completed in 11 sec
```

- Pass 2 of 2: Completed in 11 sec  
Evaluation completed in 25 sec

### 将数据导入内存

神经网络训练是迭代进行的。在每次迭代中，数据存储从文件中读取数据，变换数据，然后更新网络系数。如果数据可放入计算机的内存中，则将数据从文件导入内存可以消除重复的文件读取和变换过程，从而加快训练速度。这样，只需执行一次从文件读取并变换数据的操作。使用磁盘上的数据文件训练此网络需要大约 110 分钟，而使用内存中的数据只需要大约 50 分钟。

将文件中的所有数据导入内存。这些文件有两个变量：**frame** 和 **label**，对数据存储的每个 **read** 调用都返回一个元胞数组，其中第一个元素是 **frame**，第二个元素是 **label**。使用 **transform** 函数 **helperModClassReadFrame** 和 **helperModClassReadLabel** 读取帧和标签。如果您有 Parallel Computing Toolbox 许可证，请使用 **tall** 数组来实现变换函数的并行处理。由于 **gather** 函数默认情况下在第一个维度上串联 **read** 函数的输出，因此以元胞数组的形式返回帧并在第四个维度上手动进行串联。

```
% Gather the training and validation frames into the memory
trainFramesTall = tall(transform(trainDSTrans, @helperModClassReadFrame));
rxTrainFrames = gather(trainFramesTall);
```

Evaluating tall expression using the Parallel Pool 'local':  
- Pass 1 of 1: Completed in 4.3 sec  
Evaluation completed in 4.3 sec

```
rxTrainFrames = cat(4, rxTrainFrames{:});
validFramesTall = tall(transform(validDSTrans, @helperModClassReadFrame));
rxValidFrames = gather(validFramesTall);
```

Evaluating tall expression using the Parallel Pool 'local':  
- Pass 1 of 1: Completed in 0.76 sec  
Evaluation completed in 0.78 sec

```
rxValidFrames = cat(4, rxValidFrames{:});
```

```
% Gather the training and validation labels into the memory
trainLabelsTall = tall(transform(trainDSTrans, @helperModClassReadLabel));
rxTrainLabels = gather(trainLabelsTall);
```

Evaluating tall expression using the Parallel Pool 'local':  
- Pass 1 of 2: Completed in 4.6 sec  
- Pass 2 of 2: Completed in 7 sec  
Evaluation completed in 12 sec

```
validLabelsTall = tall(transform(validDSTrans, @helperModClassReadLabel));
rxValidLabels = gather(validLabelsTall);
```

Evaluating tall expression using the Parallel Pool 'local':  
- Pass 1 of 2: Completed in 0.7 sec  
- Pass 2 of 2: Completed in 0.85 sec  
Evaluation completed in 1.8 sec

### 训练 CNN

本示例使用的 CNN 由六个卷积层和一个全连接层组成。除最后一个卷积层外，每个卷积层后面都有一个批量归一化层、修正线性单元 (ReLU) 激活层和最大池化层。在最后一个卷积层中，最大池化层被一个平均池化层取代。输出层具有 softmax 激活。有关网络设计指导原则，请参阅 “Deep Learning Tips and Tricks”。

```
modClassNet = helperModClassCNN(modulationTypes,sps,spf);
```

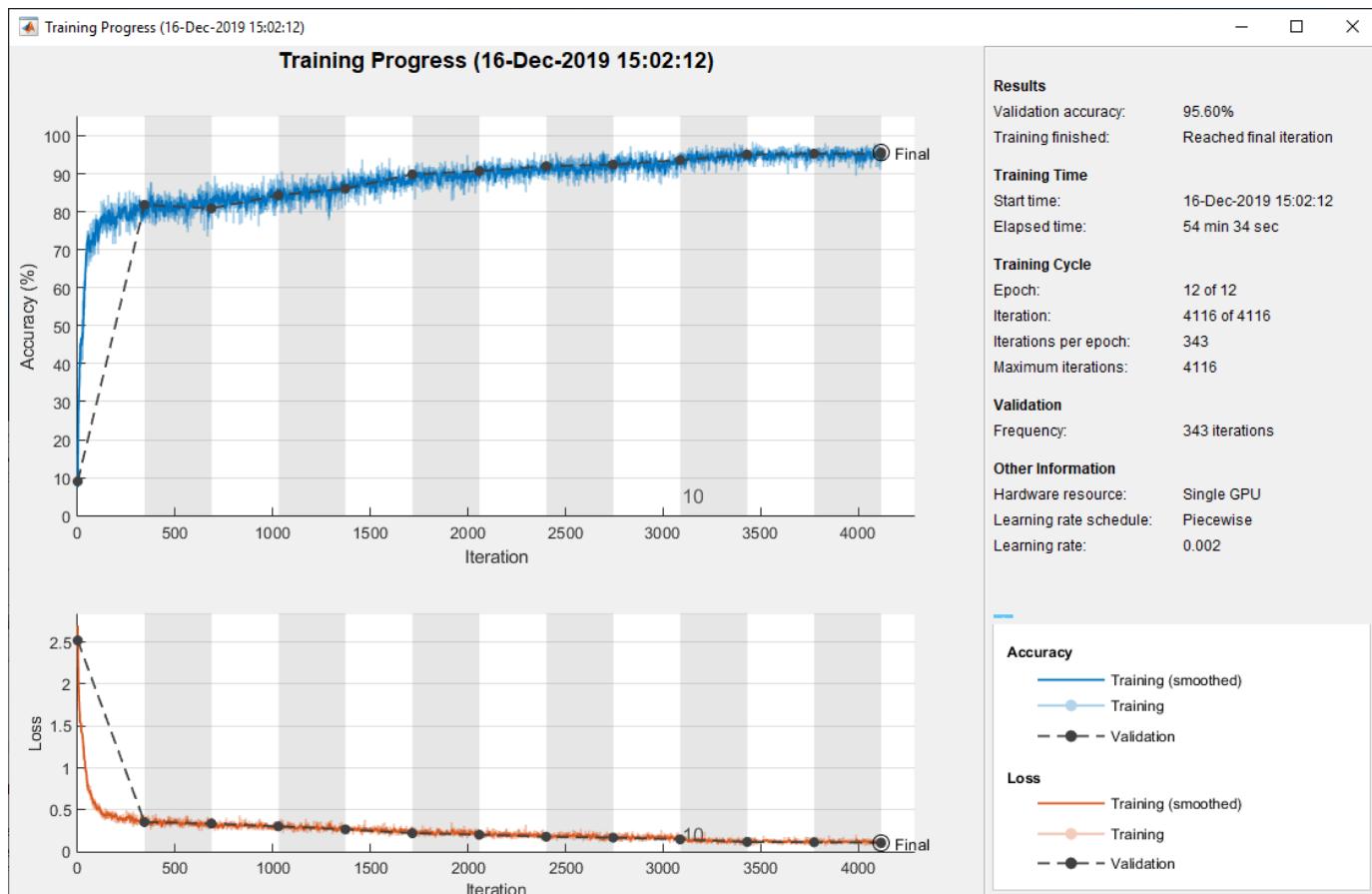
接下来配置 `TrainingOptionsSGDM` 以使用小批量大小为 256 的 SGDM 求解器。将最大轮数设置为 12，因为更多轮数不会提供进一步的训练优势。默认情况下，'ExecutionEnvironment' 属性设置为 'auto'，其中 `trainNetwork` 函数使用 GPU（如果可用）或使用 CPU（如果 GPU 不可用）。要使用 GPU，您必须拥有 Parallel Computing Toolbox 许可证。将初始学习率设置为  $2 \times 10^{-2}$ 。每 9 轮后将学习率降低十分之一。将 'Plots' 设置为 'training-progress' 以对训练进度绘图。在 NVIDIA Titan Xp GPU 上，网络需要大约 25 分钟来完成训练。

```
maxEpochs = 12;
miniBatchSize = 256;
options = helperModClassTrainingOptions(maxEpochs,miniBatchSize,...  
    numel(rxTrainLabels),rxValidFrames,rxValidLabels);
```

或者训练网络，或者使用已经过训练的网络。默认情况下，此示例使用经过训练的网络。

```
if trainNow == true
    fprintf("%s - Training the network\n", datestr(toc/86400,'HH:MM:SS'))
    trainedNet = trainNetwork(rxTrainFrames,rxTrainLabels,modClassNet,options);
else
    load trainedModulationClassificationNetwork
end
```

如训练进度图所示，网络在大约 12 轮后收敛于超过 95% 的准确度。



通过获得测试帧的分类准确度来评估经过训练的网络。结果表明，该网络对这组波形实现的准确度达到 94% 左右。

```
fprintf("%s - Classifying test frames\n", datestr(toc/86400,'HH:MM:SS'))  
00:02:18 - Classifying test frames  
  
% Gather the test frames into the memory  
testFramesTall = tall(transform(testDSTrans, @helperModClassReadFrame));  
rxTestFrames = gather(testFramesTall);  
  
Evaluating tall expression using the Parallel Pool 'local':  
- Pass 1 of 1: Completed in 0.68 sec  
Evaluation completed in 0.69 sec  
  
rxTestFrames = cat(4, rxTestFrames{:});  
  
% Gather the test labels into the memory  
testLabelsTall = tall(transform(testDSTrans, @helperModClassReadLabel));  
rxTestLabels = gather(testLabelsTall);  
  
Evaluating tall expression using the Parallel Pool 'local':  
- Pass 1 of 2: Completed in 0.7 sec  
- Pass 2 of 2: Completed in 0.86 sec  
Evaluation completed in 1.8 sec  
  
rxTestPred = classify(trainedNet,rxTestFrames);  
testAccuracy = mean(rxTestPred == rxTestLabels);  
disp("Test accuracy: " + testAccuracy*100 + "%")  
Test accuracy: 95.4545%
```

绘制测试帧的混淆矩阵。如矩阵所示，网络混淆了 16-QAM 和 64-QAM 帧。此问题是预料之中的，因为每个帧只携带 128 个符号，而 16-QAM 是 64-QAM 的子集。该网络还混淆了 QPSK 和 8-PSK 帧，因为在通道衰落和频率偏移引发相位旋转后，这些调制类型的星座图看起来相似。

```
figure  
cm = confusionchart(rxTestLabels, rxTestPred);  
cm.Title = 'Confusion Matrix for Test Data';  
cm.RowSummary = 'row-normalized';  
cm.Parent.Position = [cm.Parent.Position(1:2) 740 424];
```

Confusion Matrix for Test Data										
True Class	16QAM	42	8							
	64QAM	10	40							
	8PSK	2		46					2	
	B-FM				50					
	BPSK					50				
	CPFSK						50			
	DSB-AM							49		1
	GFSK								50	
	PAM4									50
	QPSK	1		1						48
	SSB-AM									50

Predicted Class

### 使用 SDR 进行测试

使用 `helperModClassSDRTTest` 函数，通过无线信号测试经过训练的网络的性能。要执行此测试，您必须有专用的 SDR 用于发送和接收。您可以使用两个 ADALM-PLUTO 无线电，或使用一个 ADALM-PLUTO 无线电和一个 USRP® 无线电分别进行发送和接收。您必须安装 Communications Toolbox Support Package for ADALM-PLUTO Radio。如果使用 USRP® 无线电，还必须安装 Communications Toolbox Support Package for USRP® Radio。`helperModClassSDRTTest` 函数使用生成训练信号所用的同一调制函数，然后使用 ADALM-PLUTO 无线电进行传输。使用针对信号接收配置的 SDR (ADALM-PLUTO 或 USRP® 无线电) 捕获通道减损的信号，而不对通道进行仿真。使用经过训练的网络和先前使用的同一 `classify` 函数来预测调制类型。运行下一个代码段会生成一个混淆矩阵，并输出测试准确度。

```

radioPlatform = ADALM-PLUTO;

switch radioPlatform
case "ADALM-PLUTO"
    if helperIsPlutoSDRInstalled() == true
        radios = findPlutoRadio();
        if length(radios) >= 2
            helperModClassSDRTTest(radios);
        else
            disp('Selected radios not found. Skipping over-the-air test.')
        end
    end
case {"USRP B2xx","USRP X3xx","USRP N2xx"}
    if (helperIsUSRPIInstalled() == true) && (helperIsPlutoSDRInstalled() == true)
        txRadio = findPlutoRadio();
        rxRadio = findsdr();
        switch radioPlatform

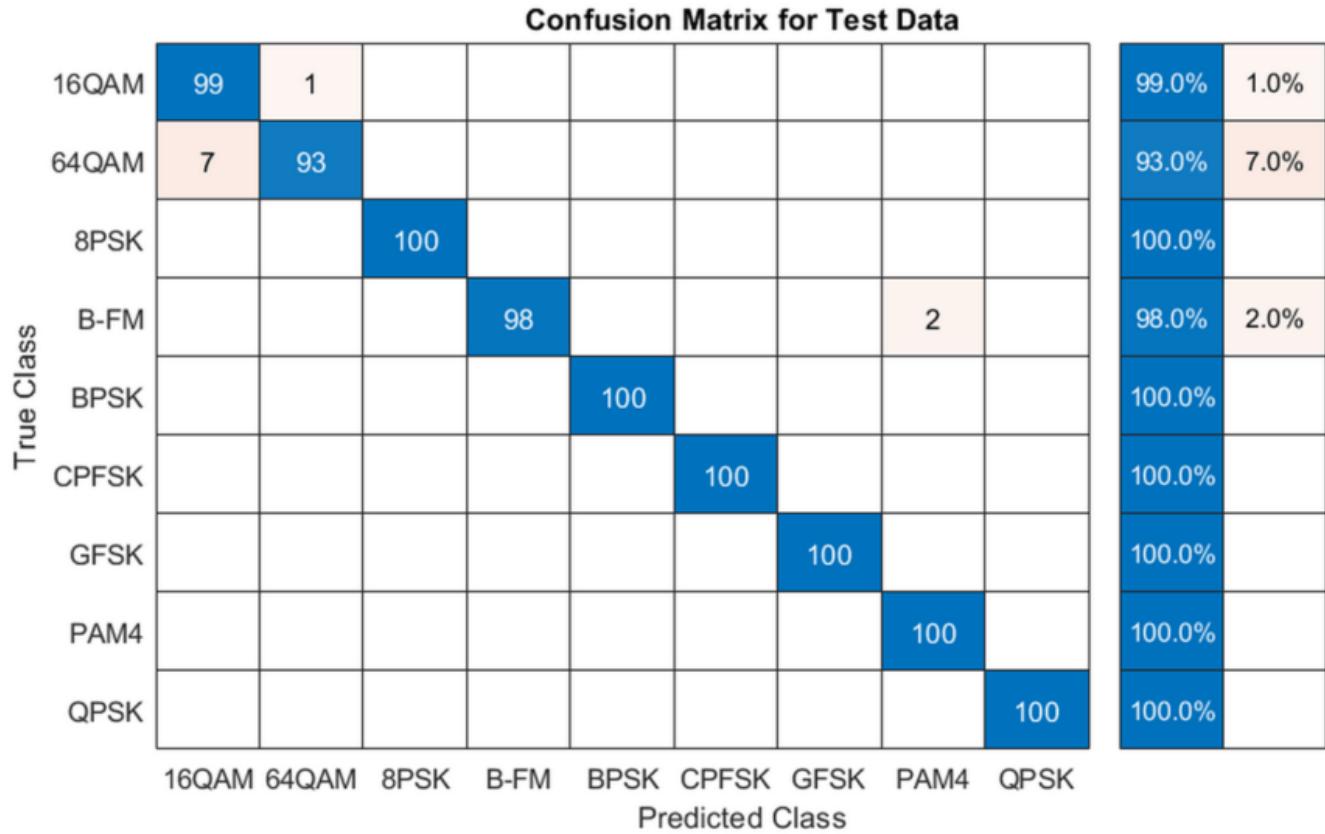
```

```

case "USRP B2xx"
idx = contains({rxRadio.Platform}, {'B200','B210'});
case "USRP X3xx"
idx = contains({rxRadio.Platform}, {'X300','X310'});
case "USRP N2xx"
idx = contains({rxRadio.Platform}, 'N200/N210/USRP2');
end
rxRadio = rxRadio(idx);
if (length(txRadio) >= 1) && (length(rxRadio) >= 1)
    helperModClassSDRTTest(rxRadio);
else
    disp('Selected radios not found. Skipping over-the-air test.')
end
end
end

```

当使用两个相隔约 2 英尺的固定 ADALM-PLUTO 无线电时，网络可实现 99% 的总体准确度，混淆矩阵如下图所示。结果可因试验设置而异。



### 进一步探查

要提高准确度，可以优化超参数，例如滤波器数量、滤波器大小；或者优化网络结构，例如添加更多层，使用不同激活层等。

Communication Toolbox 提供了更多调制类型和通道减损。有关详细信息，请参阅“Modulation”(Communications Toolbox)和“Propagation and Channel Models”(Communications Toolbox)部

分。您还可以使用 LTE Toolbox、WLAN Toolbox 和 5G Toolbox 添加标准特定信号。您还可以使用 Phased Array System Toolbox 添加雷达信号。

helperModClassGetModulator 函数提供用于生成调制信号的 MATLAB 函数。您还可以探查以下函数和 System object 以获得详细信息：

- helperModClassGetModulator.m
- helperModClassTestChannel.m
- helperModClassGetSource.m
- helperModClassFrameGenerator.m
- helperModClassCNN.m
- helperModClassTrainingOptions.m

## 参考资料

- 1 O'Shea, T. J., J. Corgan, and T. C. Clancy."Convolutional Radio Modulation Recognition Networks."Preprint, submitted June 10, 2016. <https://arxiv.org/abs/1602.04105>
- 2 O'Shea, T. J., T. Roy, and T. C. Clancy."Over-the-Air Deep Learning Based Radio Signal Classification."IEEE Journal of Selected Topics in Signal Processing.Vol. 12, Number 1, 2018, pp. 168–179.
- 3 Liu, X., D. Yang, and A. E. Gamal."Deep Neural Network Architectures for Modulation Classification."Preprint, submitted January 5, 2018. <https://arxiv.org/abs/1712.00443v3>

## 另请参阅

[trainingOptions](#) | [trainNetwork](#)

## 详细信息

- "在 MATLAB 中进行深度学习" (第 1-2 页)



## 音频示例

---

## 基于小波散射和深度学习的口述数字识别

此示例说明如何使用机器学习和深度学习方法对语音数字进行分类。在此示例中，您将使用小波时间散射配合支持向量机 (SVM) 和长短期记忆 (LSTM) 网络执行分类。您还可以应用贝叶斯优化来确定合适的超参数，以提高 LSTM 网络的准确度。此外，该示例说明了一种使用深度卷积神经网络 (CNN) 和 Mel 频谱图的方法。

### 数据

克隆或下载 Free Spoken Digit Dataset (FSDD)，可从 <https://github.com/Jakobovski/free-spoken-digit-dataset> 获取。FSDD 是一个开放数据集，这意味着它可以随着时间的推移而增长。此示例使用 2019 年 1 月 29 日提交的版本，其中包含四个说话者分别朗读英文数字 0 至 9 的 2000 个录音。在此版本中，两个说话者以美式英语为母语，另外两个说话者不以英语为母语且分别带有比利时法语和德语口音。数据以 8000 Hz 的频率采样。

使用 `audioDatastore` 管理数据访问，并确保将录音随机分为训练集和测试集。将 `location` 属性设置为计算机上 FSDD 录音文件夹的位置，例如：

```
pathToRecordingsFolder = fullfile(tempdir,'free-spoken-digit-dataset-master','recordings');
location = pathToRecordingsFolder;
```

将 `audioDatastore` 指向该位置。

```
ads = audioDatastore(location);
```

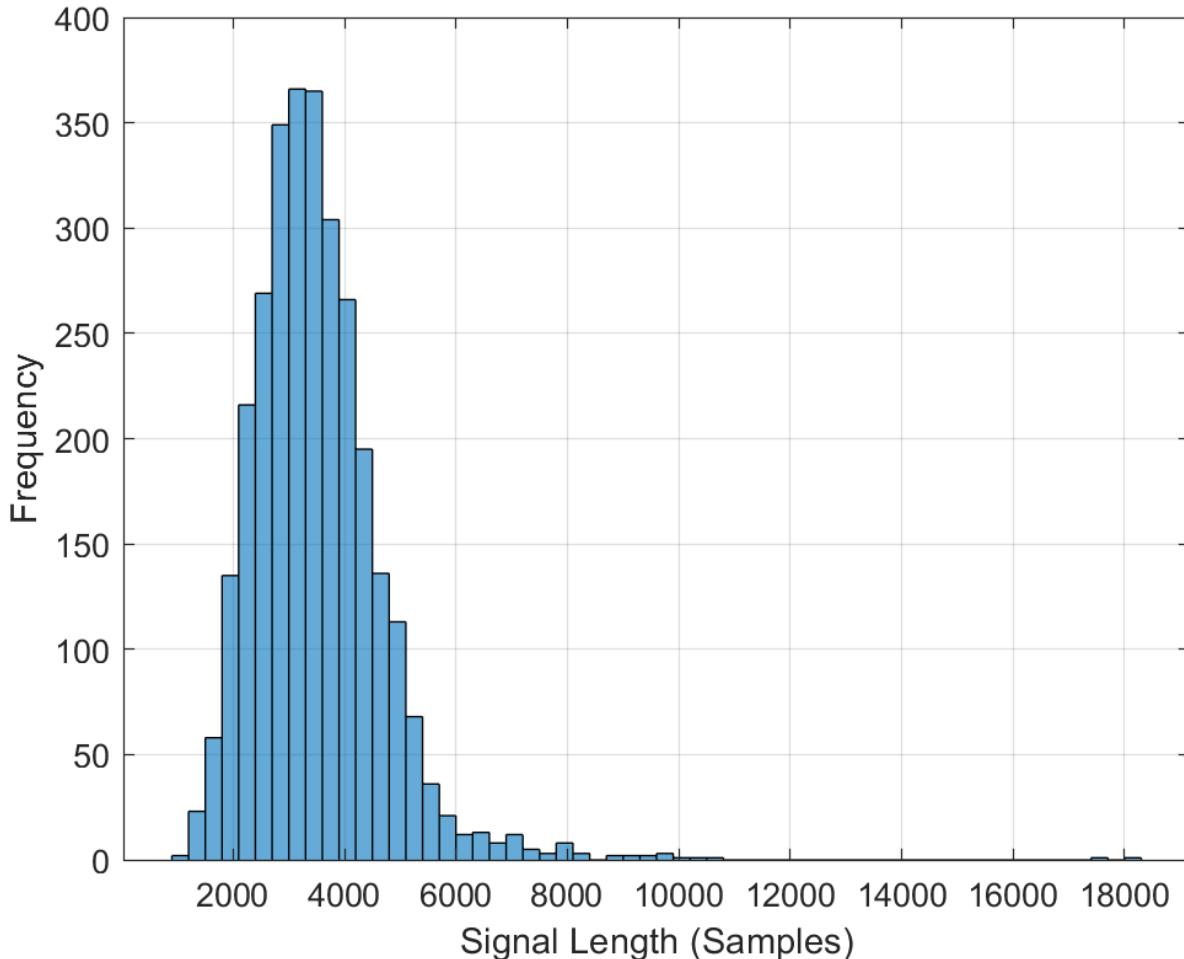
辅助函数 `helpergenLabels` 基于 FSDD 文件创建标签分类数组。附录中列出了 `helpergenLabels` 的源代码。列出各类和每个类中的示例数量。

```
ads.Labels = helpergenLabels(ads);
summary(ads.Labels)
```

```
0    300
1    300
2    300
3    300
4    300
5    300
6    300
7    300
8    300
9    300
```

FSDD 数据集由 10 个平衡类组成，每个平衡类有 200 个录音。FSDD 的录音时间长度不等。FSDD 不是特别大，因此通读 FSDD 文件并构造信号长度直方图。

```
LenSig = zeros(numel(ads.Files),1);
nr = 1;
while hasdata(ads)
    digit = read(ads);
    LenSig(nr) = numel(digit);
    nr = nr+1;
end
reset(ads)
histogram(LenSig)
grid on
xlabel('Signal Length (Samples)')
ylabel('Frequency')
```



直方图显示录音长度呈正偏态分布。对于分类，此示例使用 8192 个采样点这一常见信号长度，这是保守值，可确保截断较长的录音不会切断语音内容。如果信号长度大于 8192 个采样点（1.024 秒），录音将被截断为 8192 个采样点。如果信号长度小于 8192 个采样点，将以对称方式用零对信号进行前置和后置填充，使其达到 8192 个采样点的长度。

### 小波时间散射

使用 `waveletScattering` (Wavelet Toolbox) 以 0.22 秒的不变性尺度创建小波时间散射框架。在此示例中，通过对所有时间采样点的散射变换求平均值来创建特征向量。要使每个时间窗的散射系数数量足以求平均值，请将 `OversamplingFactor` 设置为 2，以使每条路径的散射系数数量相对于临界下采样值增加为原来的四倍。

```
sf = waveletScattering('SignalLength',8192,'InvarianceScale',0.22,...  
    'SamplingFrequency',8000,'OversamplingFactor',2);
```

将 FSDD 分成训练集和测试集。将 80% 的数据分配给训练集，将 20% 保留给测试集。训练数据用于基于散射变换训练分类器。测试数据用于验证模型。

```
rng default;  
ads = shuffle(ads);
```

```
[adsTrain,adsTest] = splitEachLabel(ads,0.8);
countEachLabel(adsTrain)
```

```
ans=10×2 table
  Label    Count
```

Label	Count
0	240
1	240
2	240
3	240
4	240
5	240
6	240
7	240
8	240
9	240

```
countEachLabel(adsTest)
```

```
ans=10×2 table
  Label    Count
```

Label	Count
0	60
1	60
2	60
3	60
4	60
5	60
6	60
7	60
8	60
9	60

辅助函数 `helperReadSPData` 将数据截断或填充为 8192 的长度，并将每个录音按其最大值进行归一化。附录中列出了 `helperReadSPData` 的源代码。创建一个  $8192 \times 1600$  矩阵，其中每列是一个语音数字录音。

```
Xtrain = [];
scatds_Train = transform(adsTrain,@(x)helperReadSPData(x));
while hasdata(scatds_Train)
    smat = read(scatds_Train);
    Xtrain = cat(2,Xtrain,smat);
end
```

对测试集重复该过程。生成的矩阵为  $8192 \times 400$ 。

```
Xtest = [];
scatds_Test = transform(adsTest,@(x)helperReadSPData(x));
while hasdata(scatds_Test)
    smat = read(scatds_Test);
    Xtest = cat(2,Xtest,smat);
end
```

对训练集和测试集应用小波散射变换。

```
Strain = sf.featureMatrix(Xtrain);
Stest = sf.featureMatrix(Xtest);
```

获取训练集和测试集的散射特征均值。排除零阶散射系数。

```
TrainFeatures = Strain(2:end,:,:);
TrainFeatures = squeeze(mean(TrainFeatures,2))';
TestFeatures = Stest(2:end,:,:);
TestFeatures = squeeze(mean(TestFeatures,2))';
```

## SVM 分类器

现在数据已简化为针对每个记录有一个特征向量，下一步是使用这些特征对录音进行分类。用二次多项式核创建一个 SVM 学习器模板。对训练数据进行 SVM 拟合。

```
template = templateSVM(...  
    'KernelFunction', 'polynomial', ...  
    'PolynomialOrder', 2, ...  
    'KernelScale', 'auto', ...  
    'BoxConstraint', 1, ...  
    'Standardize', true);  
classificationSVM = fitcecoc(...  
    TrainFeatures, ...  
    adsTrain.Labels, ...  
    'Learners', template, ...  
    'Coding', 'onevsone', ...  
    'ClassNames', categorical({'0'; '1'; '2'; '3'; '4'; '5'; '6'; '7'; '8'; '9'}));
```

使用 k 折交叉验证来基于训练数据预测模型的泛化准确度。将训练集分成五个组。

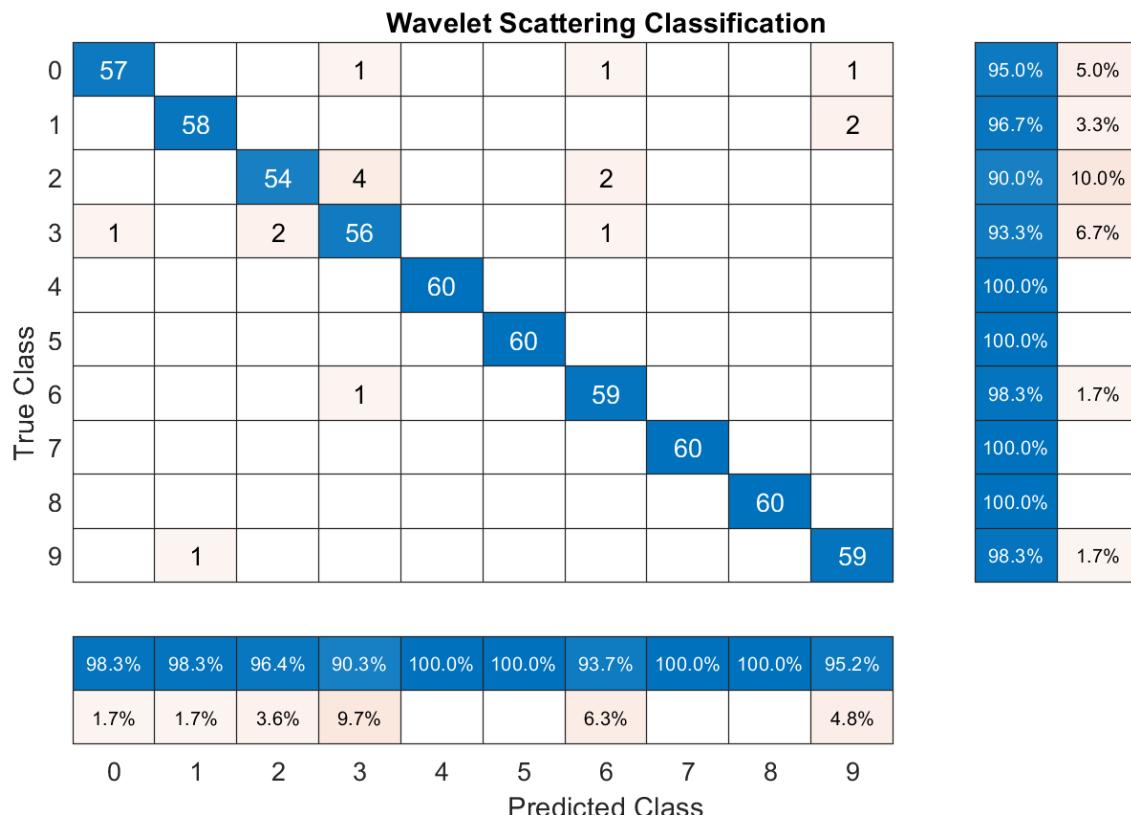
```
partitionedModel = crossval(classificationSVM, 'KFold', 5);
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);
validationAccuracy = (1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError'))*100  
validationAccuracy = 97.4167
```

估计的泛化准确度约为 97%。使用经过训练的 SVM 预测测试集中的口述数字类。

```
predLabels = predict(classificationSVM, TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100  
testAccuracy = 97.1667
```

用混淆图总结模型基于测试集的预测效果。使用列汇总和行汇总显示每个类的准确率和召回率。混淆图底部的表显示每个类的精确率值。混淆图右侧的表显示召回值。

```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
ccscat = confusionchart(adsTest.Labels,predLabels);
ccscat.Title = 'Wavelet Scattering Classification';
ccscat.ColumnSummary = 'column-normalized';
ccscat.RowSummary = 'row-normalized';
```



使用散射变换配合 SVM 分类器对测试集中的口述数字进行分类，准确度达到 98%（即错误率为 2%）。

### 长短期记忆 (LSTM) 网络

LSTM 网络是一种循环神经网络 (RNN)。RNN 是专门用于处理序列或时序数据（如语音数据）的神经网络。由于小波散射系数是序列，因此它们可以用作 LSTM 的输入。通过使用散射特征而不是原始数据，您可以减少网络需要学习的变异性。

修改用于 LSTM 网络的训练和测试散射特征。排除零阶散射系数，并将特征转换为元胞数组。

```
TrainFeatures = Strain(2:end,:,:);
TrainFeatures = squeeze(num2cell(TrainFeatures,[1 2]));
TestFeatures = Stest(2:end,:,:);
TestFeatures = squeeze(num2cell(TestFeatures, [1 2]));
```

构造一个包含 512 个隐含层的简单 LSTM 网络。

```
[inputSize, ~] = size(TrainFeatures{1});
YTrain = adsTrain.Labels;

numHiddenUnits = 512;
numClasses = numel(unique(YTrain));

layers = [
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
```

```
softmaxLayer
classificationLayer];
```

设置超参数。使用 Adam 优化和小批量大小为 50。将最大训练轮数设置为 300。使用 1e-4 作为学习率。如果您不想使用绘图跟踪进度，可以关闭训练进度绘图。默认情况下，训练使用 GPU（如果有）。否则，将使用 CPU。有关详细信息，请参阅 **trainingOptions**。

```
maxEpochs = 300;
miniBatchSize = 50;

options = trainingOptions('adam',...
    'InitialLearnRate',0.0001,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','shortest',...
    'Shuffle','every-epoch',...
    'Verbose', false, ...
    'Plots','training-progress');
```

训练网络。

```
net = trainNetwork(TrainFeatures,YTrain,layers,options);

predLabels = classify(net,TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100

testAccuracy = 96.3333
```

## 贝叶斯优化

确定合适的超参数设置通常是训练深度网络最困难的部分之一。您可以使用贝叶斯优化来帮助解决这一难题。在此示例中，您将使用贝叶斯方法优化隐藏层的数量和初始学习率。创建一个新目录来存储 MAT 文件，其中包含有关超参数设置和网络的信息以及相应的错误率。

```
YTrain = adsTrain.Labels;
YTest = adsTest.Labels;

if ~exist("results",'dir')
    mkdir results
end
```

初始化要优化的变量及其值范围。由于隐含层的数量必须为整数，请将 'type' 设置为 'integer'。

```
optVars = [
    optimizableVariable('InitialLearnRate',[1e-5, 1e-1],'Transform','log')
    optimizableVariable('NumHiddenUnits',[10, 1000],'Type','integer')
];
```

贝叶斯优化涉及大量计算，可能需要几个小时才能完成。出于此示例的目的，请将 **optimizeCondition** 设置为 **false** 以下载并使用预先确定的经过优化的超参数设置。如果您将 **optimizeCondition** 设置为 **true**，将使用贝叶斯优化求目标函数 **helperBayesOptLSTM** 最小值。附录中列出的目标函数是给定特定超参数设置时网络的错误率。加载的设置是针对目标函数最小值 0.02 (2% 错误率)。

```
ObjFcn = helperBayesOptLSTM(TrainFeatures,YTrain,TestFeatures,YTest);

optimizeCondition = false;
if optimizeCondition
    BayesObject = bayesopt(ObjFcn,optVars,...
```

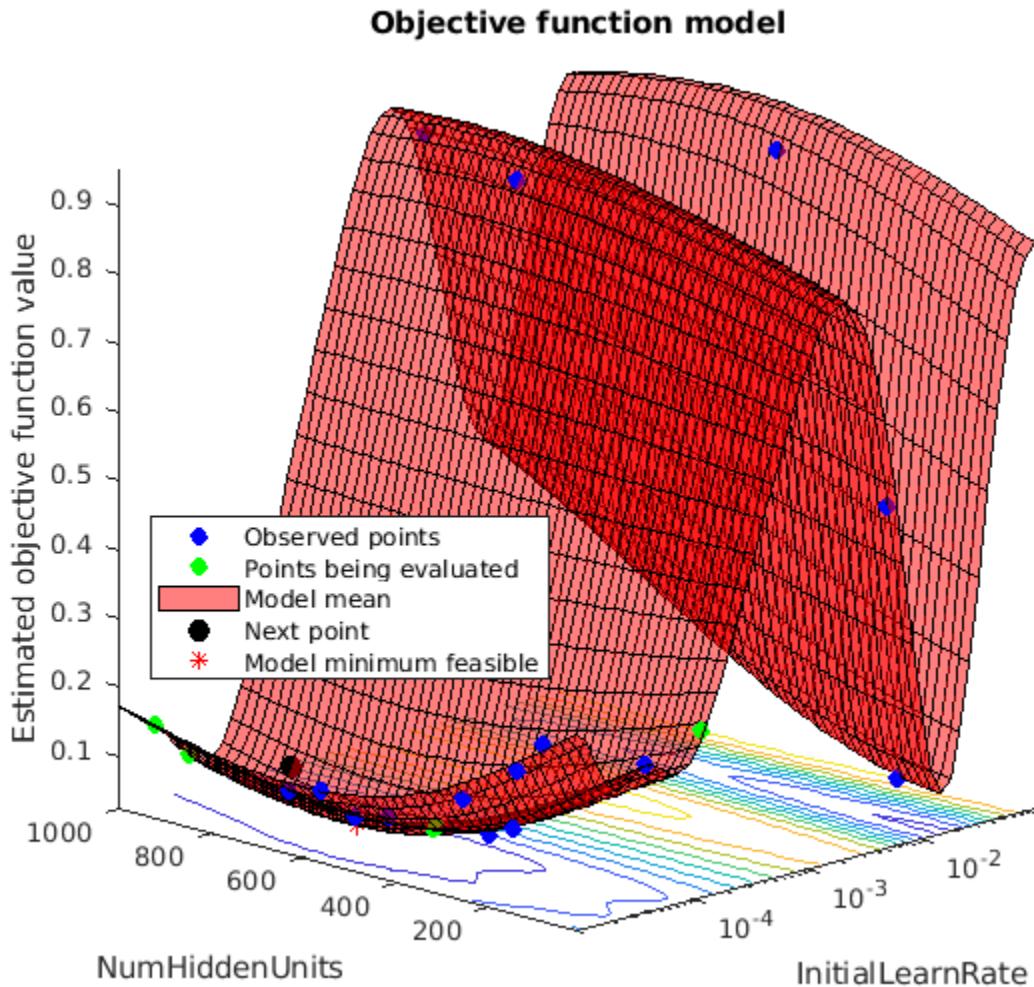
```

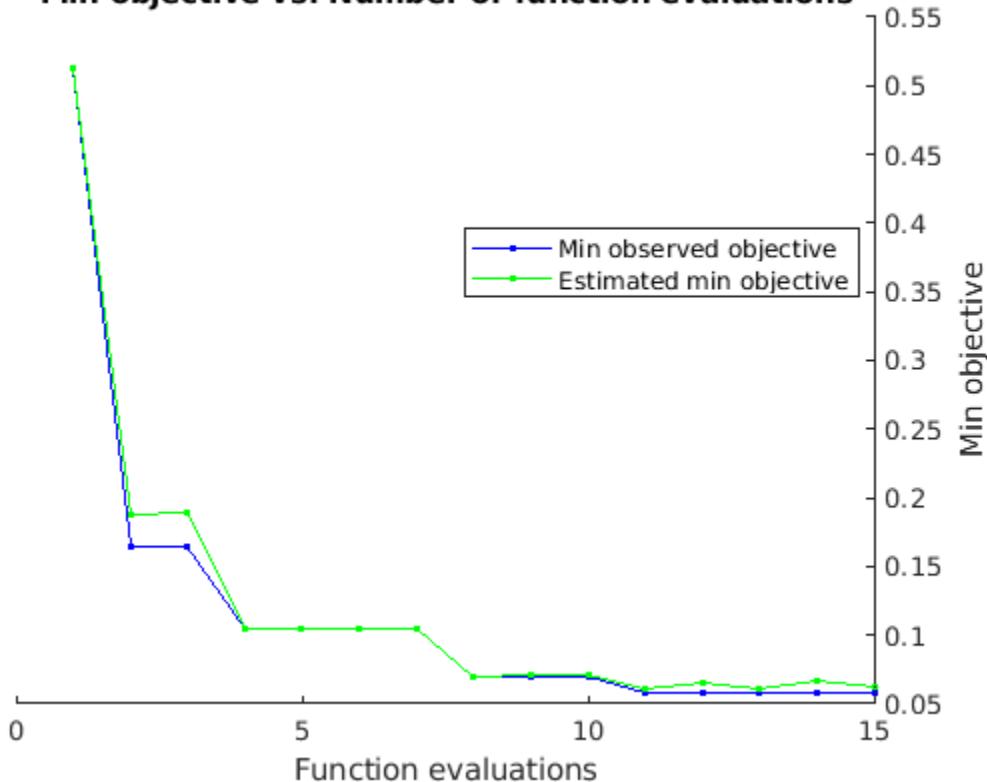
'MaxObjectiveEvaluations',15,...
'IsObjectiveDeterministic',false,...
'UseParallel',true);
else
url = 'http://ssd.mathworks.com/supportfiles/audio/SpokenDigitRecognition.zip';
downloadNetFolder = tempdir;
netFolder = fullfile(downloadNetFolder,'SpokenDigitRecognition');
if ~exist(netFolder,'dir')
    disp('Downloading pretrained network (1 file - 12 MB) ...')
    unzip(url,downloadNetFolder)
end
load(fullfile(netFolder,'0.02.mat'));
end

```

Downloading pretrained network (1 file - 12 MB) ...

如果执行贝叶斯优化，将生成类似以下的图形，以跟踪目标函数值以及对应的超参数值和迭代次数。您可以增加贝叶斯优化迭代的次数，以确保达到目标函数的全局最小值。



**Min objective vs. Number of function evaluations**

使用隐含单元数量和初始学习率的优化值，重新训练网络。

```

numHiddenUnits = 768;
numClasses = numel(unique(YTrain));

layers = [ ...
    sequenceInputLayer(inputSize)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

maxEpochs = 300;
miniBatchSize = 50;

options = trainingOptions('adam',...
    'InitialLearnRate',2.198827960269379e-04,...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'SequenceLength','shortest', ...
    'Shuffle','every-epoch',...
    'Verbose', false, ...
    'Plots','training-progress');

net = trainNetwork(TrainFeatures,YTrain,layers,options);
predLabels = classify(net,TestFeatures);
testAccuracy = sum(predLabels==adsTest.Labels)/numel(predLabels)*100

```

```
testAccuracy = 97.5000
```

如图所示，使用贝叶斯优化可以得到更高准确度的 LSTM。

### 使用 Mel 频谱图的深度卷积网络

下面用另一种方法来完成口述数字识别任务，使用基于 Mel 频谱图的深度卷积神经网络 (DCNN) 对 FSDD 数据集进行分类。使用与散射变换中相同的信号截断/填充过程。同样，用每个信号采样点除以最大绝对值来归一化每个录音。为了保持一致，使用与散射变换相同的训练集和测试集。

设置 Mel 频谱图的参数。使用与散射变换中相同的窗（即帧）长度，即 0.22 秒。将窗之间的帧移设置为 10 ms。使用 40 个频带。

```
segmentDuration = 8192*(1/8000);
frameDuration = 0.22;
hopDuration = 0.01;
numBands = 40;
```

重置训练和测试数据存储。

```
reset(adsTrain);
reset(adsTest);
```

在本示例末尾定义的辅助函数 `helperspeechSpectrograms` 使用 `melSpectrogram` 在标准化录音长度和归一化幅值后获得 Mel 频谱图。使用 Mel 频谱图的对数作为 DCNN 的输入。为了避免对零取对数，请对每个元素加一个小  $\epsilon$ 。

```
epsil = 1e-6;
XTrain = helperspeechSpectrograms(adsTrain,segmentDuration,frameDuration,hopDuration,numBands);
```

```
Computing speech spectrograms...
Processed 500 files out of 2400
Processed 1000 files out of 2400
Processed 1500 files out of 2400
Processed 2000 files out of 2400
...done
```

```
XTrain = log10(XTrain + epsil);
```

```
XTest = helperspeechSpectrograms(adsTest,segmentDuration,frameDuration,hopDuration,numBands);
```

```
Computing speech spectrograms...
Processed 500 files out of 600
...done
```

```
XTest = log10(XTest + epsil);
```

```
YTrain = adsTrain.Labels;
YTest = adsTest.Labels;
```

### 定义 DCNN 架构

构造一个小 DCNN 作为层的数组。使用卷积和批量归一化层，并使用最大池化层对特征图下采样。为了降低网络记住训练数据特定特征的可能性，可为最后一个全连接层的输入添加一个小的丢弃率。

```
sz = size(XTrain);
specSize = sz(1:2);
imageSize = [specSize 1];
```

```

numClasses = numel(categories(YTrain));

dropoutProb = 0.2;
numF = 12;
layers = [
    imageInputLayer(imageSize)

    convolution2dLayer(5,numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,2*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(3,'Stride',2,'Padding','same')

    convolution2dLayer(3,4*numF,'Padding','same')
    batchNormalizationLayer
    reluLayer

    maxPooling2dLayer(2)

    dropoutLayer(dropoutProb)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer('Classes',categories(YTrain));
];

```

设置用于训练网络的超参数。使用 50 作为小批量大小， $1e-4$  作为学习率。指定 Adam 优化。由于本示例中的数据量相对较小，为了可再现性，请将执行环境设置为 'cpu'。您还可以通过将执行环境设置为 'gpu' 或 'auto' 在可用的 GPU 上训练网络。有关详细信息，请参阅 [trainingOptions](#)。

```

miniBatchSize = 50;
options = trainingOptions('adam', ...
    'InitialLearnRate',1e-4, ...
    'MaxEpochs',30, ...
    'MiniBatchSize',miniBatchSize, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress', ...
    'Verbose',false, ...
    'ExecutionEnvironment','cpu');

```

训练网络。

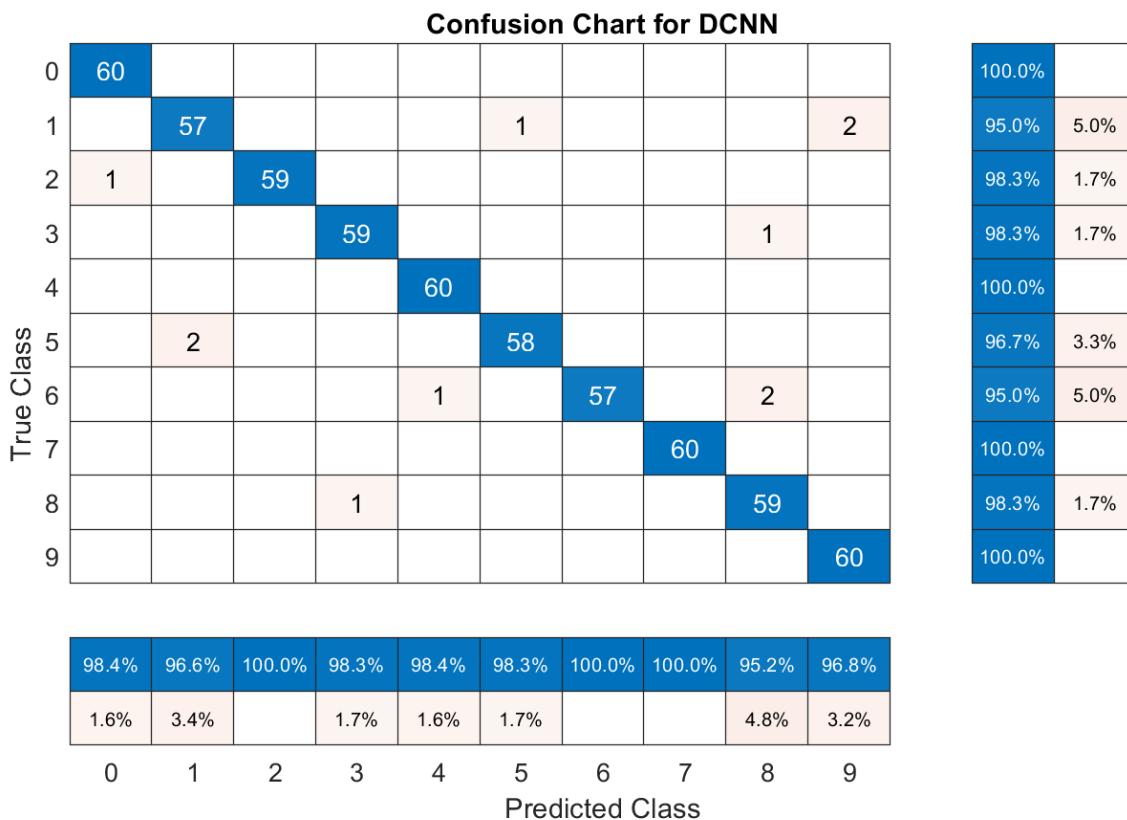
```
trainedNet = trainNetwork(XTrain,YTrain,layers,options);
```

使用经过训练的网络来预测测试集的数字标签。

```
[Ypredicted,probs] = classify(trainedNet,XTest,'ExecutionEnvironment','CPU');
cnnAccuracy = sum(Ypredicted==YTest)/numel(YTest)*100
cnnAccuracy = 98.1667
```

用混淆图总结经过训练的网络基于测试集的性能。使用列汇总和行汇总显示每个类的准确率和召回率。混淆图底部的表显示精确率值。混淆图右侧的表显示召回值。

```
figure('Units','normalized','Position',[0.2 0.2 0.5 0.5]);
ccDCNN = confusionchart(YTest,Ypredicted);
ccDCNN.Title = 'Confusion Chart for DCNN';
ccDCNN.ColumnSummary = 'column-normalized';
ccDCNN.RowSummary = 'row-normalized';
```



使用 Mel 频谱图作为输入的 DCNN 对测试集中的口述数字进行分类，准确度也约为 98%。

## 总结

此示例说明如何使用不同的机器和深度学习方法对 FSDD 中的语音数字进行分类。该示例演示了与 SVM 和 LSTM 配对的小波散射。贝叶斯方法用于优化 LSTM 超参数。最后，此示例说明如何将 CNN 和 Mel 频谱图结合使用。

该示例的目标是说明如何使用 MathWorks® 工具以完全不同但互补的方式来解决问题。所有工作流都使用 **audioDatastore** 来管理来自磁盘的数据流，并确保适当的随机化。

此示例中使用的所有方法在测试集上都表现良好。此示例的目的不是直接比较各种方法。例如，您还可以在 CNN 中使用贝叶斯优化进行超参数选择。适用于此版本 FSDD 之类的小型训练集深度学习的另一个策略是使用数据增强。操作如何影响类并不始终已知，因此数据增强并不始终可行。不过，对于语音，**audioDataAugmenter** 能提供可行的数据增强策略。

在使用小波时间散射的情况下，您也可以尝试一些修改。例如，您可以更改变换的不变性尺度，更改每个滤波器组的小波滤波器数量，并尝试不同分类器。

## 附录：辅助函数

```

function Labels = helperGenLabels(ads)
% This function is only for use in Wavelet Toolbox examples. It may be
% changed or removed in a future release.
tmp = cell(numel(ads.Files),1);
expression = "[0-9]+_";
for nf = 1:numel(ads.Files)
    idx = regexp(ads.Files{nf},expression);
    tmp{nf} = ads.Files{nf}(idx);
end
Labels = categorical(tmp);
end

function x = helperReadSPData(x)
% This function is only for use Wavelet Toolbox examples. It may change or
% be removed in a future release.

N = numel(x);
if N > 8192
    x = x(1:8192);
elseif N < 8192
    pad = 8192-N;
    prepad = floor(pad/2);
    postpad = ceil(pad/2);
    x = [zeros(prepad,1) ; x ; zeros(postpad,1)];
end
x = x./max(abs(x));

end

function x = helperBayesOptLSTM(X_train, Y_train, X_val, Y_val)
% This function is only for use in the
% "Spoken Digit Recognition with Wavelet Scattering and Deep Learning"
% example. It may change or be removed in a future release.
x = @valErrorFun;

function [valError,cons, fileName] = valErrorFun(optVars)
    %% LSTM Architecture
    [inputSize,~] = size(X_train{1});
    numClasses = numel(unique(Y_train));

    layers = [ ...
        sequenceInputLayer(inputSize)
        bilstmLayer(optVars.NumHiddenUnits,'OutputMode','last') % Using number of hidden layers value from optVars
        fullyConnectedLayer(numClasses)
        softmaxLayer
        classificationLayer];

    % Plots not displayed during training

```

```

options = trainingOptions('adam', ...
    'InitialLearnRate',optVars.InitialLearnRate, ... % Using initial learning rate value from optimizing variab
    'MaxEpochs',300, ...
    'MiniBatchSize',30, ...
    'SequenceLength','shortest', ...
    'Shuffle','never', ...
    'Verbose', false);

%% Train the network
net = trainNetwork(X_train, Y_train, layers, options);
%% Training accuracy
X_val_P = net.classify(X_val);
accuracy_training = sum(X_val_P == Y_val)./numel(Y_val);
valError = 1 - accuracy_training;
%% save results of network and options in a MAT file in the results folder along with the error value
fileName = fullfile('results', num2str(valError) + ".mat");
save(fileName,'net','valError','options')
cons = [];

end % end for inner function
end % end for outer function

function X = helperspeechSpectrograms(ads,segmentDuration,frameDuration,hopDuration,numBands)
% This function is only for use in the
% "Spoken Digit Recognition with Wavelet Scattering and Deep Learning"
% example. It may change or be removed in a future release.
%
% helperspeechSpectrograms(ads,segmentDuration,frameDuration,hopDuration,numBands)
% computes speech spectrograms for the files in the datastore ads.
% segmentDuration is the total duration of the speech clips (in seconds),
% frameDuration the duration of each spectrogram frame, hopDuration the
% time shift between each spectrogram frame, and numBands the number of
% frequency bands.
disp("Computing speech spectrograms...");

numHops = ceil((segmentDuration - frameDuration)/hopDuration);
numFiles = length(ads.Files);
X = zeros([numBands,numHops,1,numFiles],'single');

for i = 1:numFiles

    [x,info] = read(ads);
    x = normalizeAndResize(x);
    fs = info.SampleRate;
    frameLength = round(frameDuration*fs);
    hopLength = round(hopDuration*fs);

    spec = melSpectrogram(x,fs, ...
        'Window',hamming(frameLength,'periodic'), ...
        'OverlapLength',frameLength - hopLength, ...
        'FFTLength',2048, ...
        'NumBands',numBands, ...
        'FrequencyRange',[50,4000]);

    % If the spectrogram is less wide than numHops, then put spectrogram in
    % the middle of X.
    w = size(spec,2);
    left = floor((numHops-w)/2)+1;

```

```
ind = left:left+w-1;
X(:,ind,1,i) = spec;

if mod(i,500) == 0
    disp("Processed " + i + " files out of " + numFiles)
end

end

disp("...done");

end

%-----
function x = normalizeAndResize(x)
% This function is only for use in the
% "Spoken Digit Recognition with Wavelet Scattering and Deep Learning"
% example. It may change or be removed in a future release.

N = numel(x);
if N > 8192
    x = x(1:8192);
elseif N < 8192
    pad = 8192-N;
    prepad = floor(pad/2);
    postpad = ceil(pad/2);
    x = [zeros(prepad,1) ; x ; zeros(postpad,1)];
end
x = x./max(abs(x));
end
```

Copyright 2018, The MathWorks, Inc.

## 另请参阅

[trainingOptions](#) | [trainNetwork](#)

## 详细信息

- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 基于深度学习网络的鸡尾酒会信源分离

此示例说明如何使用深度学习网络隔离语音信号。

### 简介

鸡尾酒会效应指大脑专注于单个说话者而同时滤除其他声音和背景噪声的能力。人类在鸡尾酒会问题上表现很好。此示例说明如何使用深度学习网络将单个说话者从一男一女同时说话的混合语音中分离出来。

### 下载所需文件

在进入示例的具体步骤前，您需要下载预训练网络和 4 个音频文件。

```
url = 'http://ssd.mathworks.com/supportfiles/audio/CocktailPartySourceSeparation.zip';

downloadNetFolder = tempdir;
netFolder = fullfile(downloadNetFolder,'CocktailPartySourceSeparation');

if ~exist(netFolder,'dir')
    disp('Downloading pretrained network and audio files (5 files - 24.5 MB) ...')
    unzip(url,downloadNetFolder)
end
```

### 问题摘要

加载音频文件，其中包含以 4 kHz 采样的男性和女性语音。单独收听音频文件以供参考。

```
[mSpeech,Fs] = audioread(fullfile(netFolder,"MaleSpeech-16-4-mono-20secs.wav"));
sound(mSpeech,Fs)

[fSpeech] = audioread(fullfile(netFolder,"FemaleSpeech-16-4-mono-20secs.wav"));
sound(fSpeech,Fs)
```

将两个语音源合并。确保信号源在混音中具有相同的功率。对混音进行归一化，使其最大幅值为 1。

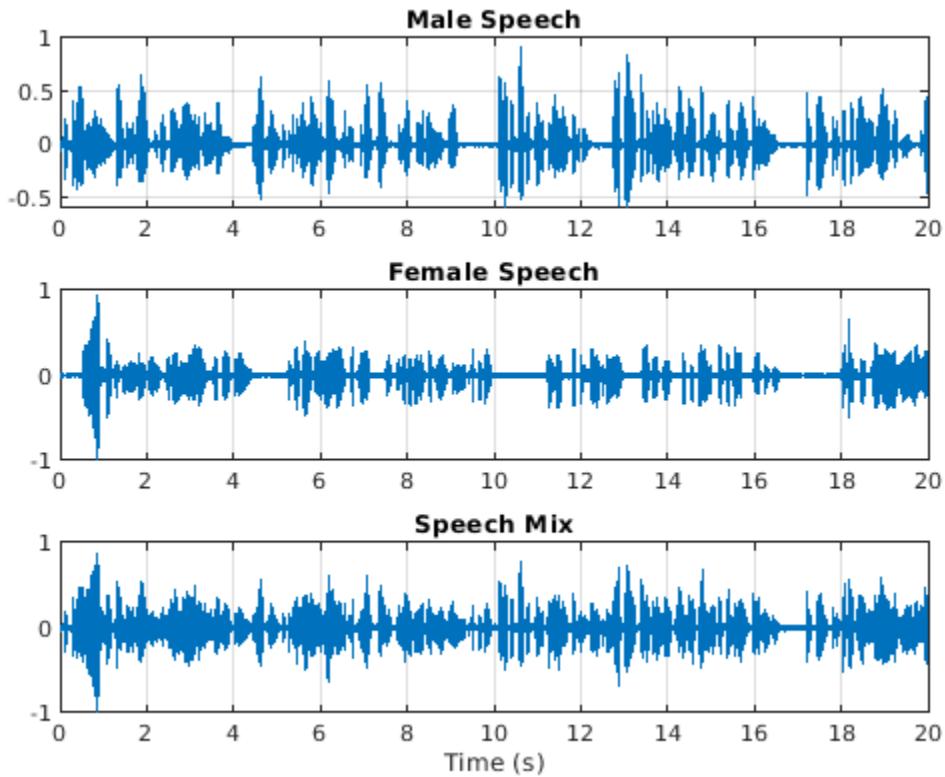
```
mSpeech = mSpeech/norm(mSpeech);
fSpeech = fSpeech/norm(fSpeech);
ampAdj = max(abs([mSpeech;fSpeech]));
mSpeech = mSpeech/ampAdj;
fSpeech = fSpeech/ampAdj;
mix = mSpeech + fSpeech;
mix = mix ./ max(abs(mix));
```

可视化原始信号和混音信号。收听混音信号。此示例说明从混合语音中提取男性和女性语音源的信源分离方案。

```
t = (0:numel(mix)-1)*(1/Fs);
```

```
figure(1)
subplot(3,1,1)
plot(t,mSpeech)
title("Male Speech")
grid on
subplot(3,1,2)
plot(t,fSpeech)
title("Female Speech")
grid on
subplot(3,1,3)
```

```
plot(t,mix)
title("Speech Mix")
xlabel("Time (s)")
grid on
```



收听混合音频。

```
sound(mix,Fs)
```

### 时频表示

使用 stft 可可视化男性语音、女性语音和混合语音信号的时频表示 (TF)。使用长度为 128、FFT 长度为 128 且重叠长度为 96 的 Hann 窗。

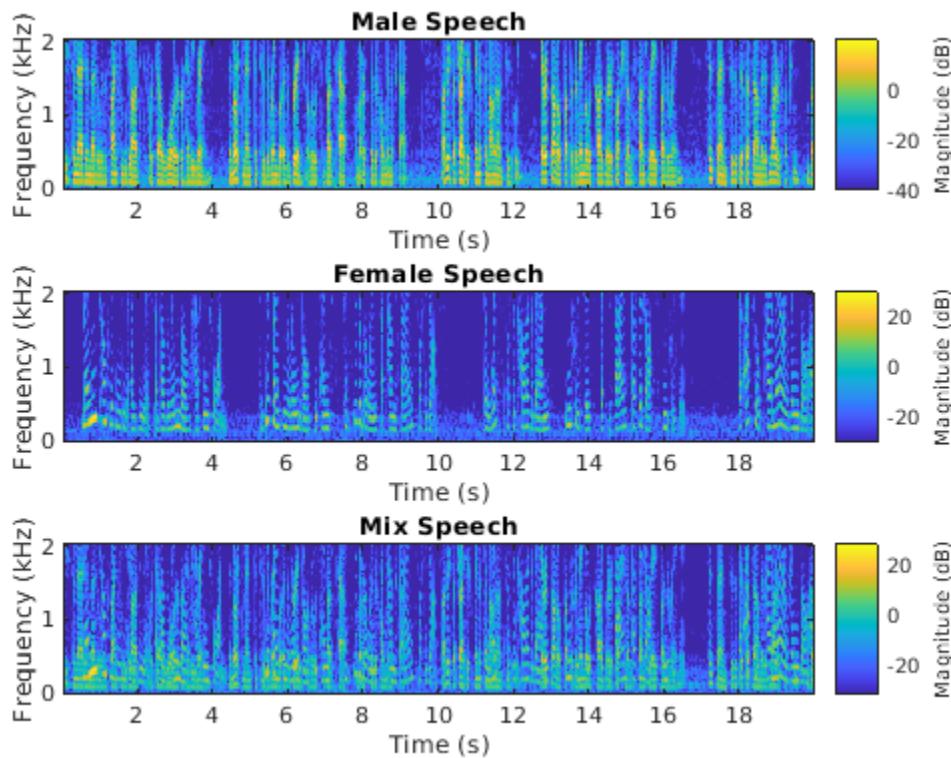
```
WindowLength = 128;
FFTLength = 128;
OverlapLength = 96;
win      = hann(WindowLength,"periodic");

figure(2)
subplot(3,1,1)
stft(mSpeech, Fs, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Male Speech")
subplot(3,1,2)
stft(fSpeech, Fs, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Female Speech")
```

```

subplot(3,1,3)
stft(mix, Fs, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Mix Speech")

```



### 使用理想时频掩膜的信源分离

TF 掩膜的应用已被证明是从多源声音中分离期望的音频信号的有效方法。TF 掩膜是与基础 STFT 大小相同的矩阵。该掩膜与基础 STFT 逐元素相乘以隔离期望的信源。TF 掩膜可以是二值掩膜或软掩膜。

### 使用理想二值掩膜的信源分离

在理想二值掩膜中，掩膜单元值为 0 或 1。如果期望信源的功率大于特定 TF 单元中其他信源合并后的功率，则该单元设置为 1。否则，单元设置为 0。

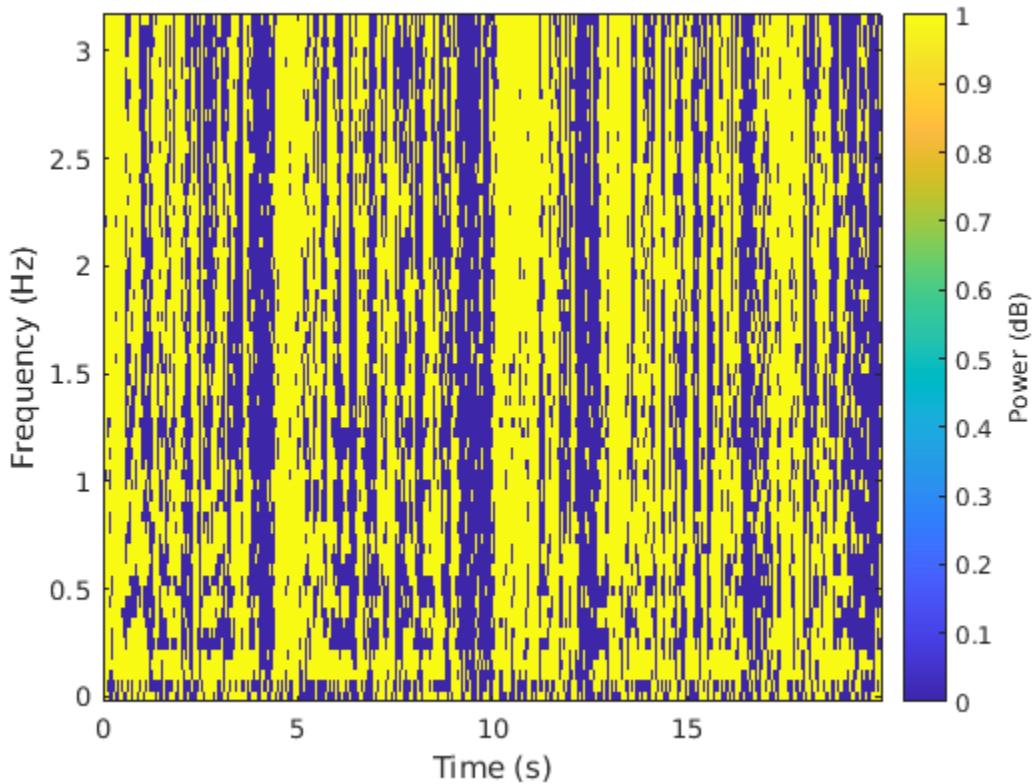
计算男性说话者的理想二值掩膜，然后将其可视化。

```

P_M      = stft(mSpeech, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
P_F      = stft(fSpeech, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
[P_mix,F] = stft(mix, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
binaryMask = abs(P_M) >= abs(P_F);

figure(3)
plotMask(binaryMask, WindowLength - OverlapLength, F, Fs)

```



通过将混音 STFT 乘以男性说话者的二值掩膜来估计男性语音的 STFT。通过将混音 STFT 乘以男性说话者的二值掩膜的倒数来估计女性语音的 STFT。

```
P_M_Hard = P_mix .* binaryMask;
P_F_Hard = P_mix .* (1-binaryMask);
```

使用逆短时 FFT (ISTFT) 估计男性和女性音频信号。可视化估计的信号和原始信号。收听估计的男性和女性语音信号。

```
mSpeech_Hard = istft(P_M_Hard , 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
fSpeech_Hard = istft(P_F_Hard , 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');

figure(4)
subplot(2,2,1)
plot(t,mSpeech)
axis([t(1) t(end) -1 1])
title("Original Male Speech")
grid on

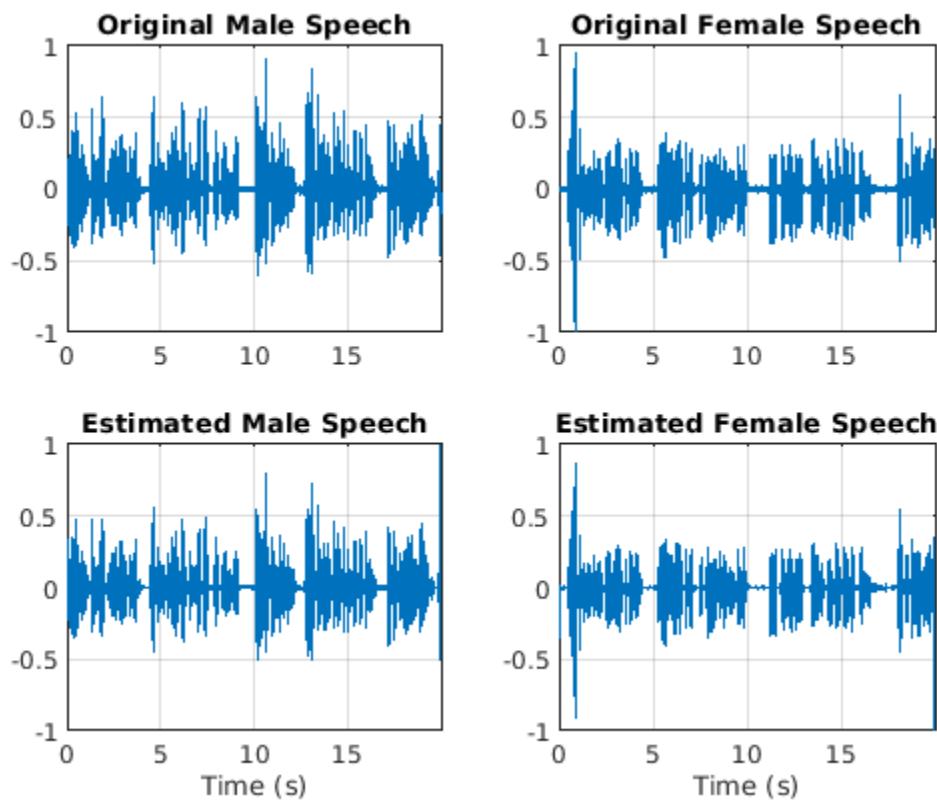
subplot(2,2,3)
plot(t,mSpeech_Hard)
axis([t(1) t(end) -1 1])
xlabel("Time (s)")
title("Estimated Male Speech")
grid on
```

```

subplot(2,2,2)
plot(t,fSpeech)
axis([t(1) t(end) -1 1])
title("Original Female Speech")
grid on

subplot(2,2,4)
plot(t,fSpeech_Hard)
axis([t(1) t(end) -1 1])
title("Estimated Female Speech")
xlabel("Time (s)")
grid on

```



```
sound(mSpeech_Hard,Fs)
```

```
sound(fSpeech_Hard,Fs)
```

### 使用理想软掩膜的信源分离

在软掩膜中，TF 掩膜单元值等于期望的信源功率与总混合功率的比率。TF 单元的值在 [0,1] 范围内。

计算男性说话者的软掩膜。通过将混音 STFT 乘以男性说话者的软掩膜来估计男性说话者的 STFT。通过将混音 STFT 乘以女性说话者的软掩膜来估计女性说话者的 STFT。

使用 ISTFT 估计男性和女性音频信号。

```
softMask = abs(P_M) ./ (abs(P_F) + abs(P_M) + eps);
```

```

P_M_Soft = P_mix .* softMask;
P_F_Soft = P_mix .* (1-softMask);

mSpeech_Soft = istft(P_M_Soft, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
fSpeech_Soft = istft(P_F_Soft, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');

```

可视化估计的信号和原始信号。收听估计的男性和女性语音信号。请注意，结果非常好，因为掩膜是在完全了解分离的男女音频信号的情况下创建的。

```

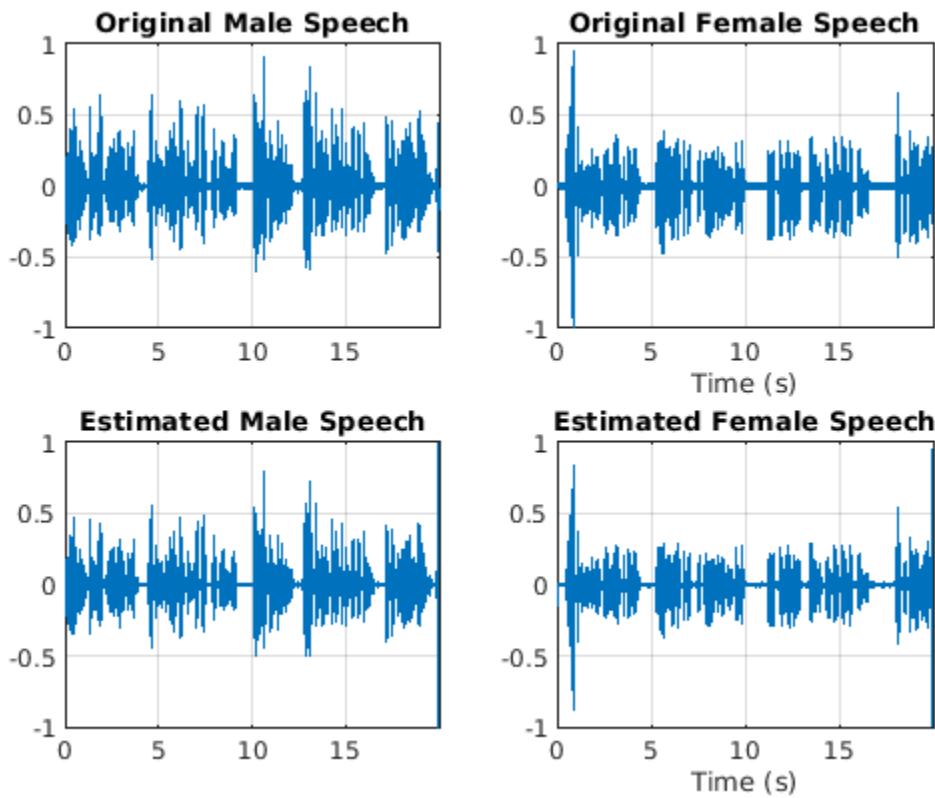
figure(5)
subplot(2,2,1)
plot(t,mSpeech)
axis([t(1) t(end) -1 1])
title("Original Male Speech")
grid on

subplot(2,2,3)
plot(t,mSpeech_Soft)
axis([t(1) t(end) -1 1])
title("Estimated Male Speech")
grid on

subplot(2,2,2)
plot(t,fSpeech)
axis([t(1) t(end) -1 1])
xlabel("Time (s)")
title("Original Female Speech")
grid on

subplot(2,2,4)
plot(t,fSpeech_Soft)
axis([t(1) t(end) -1 1])
xlabel("Time (s)")
title("Estimated Female Speech")
grid on

```



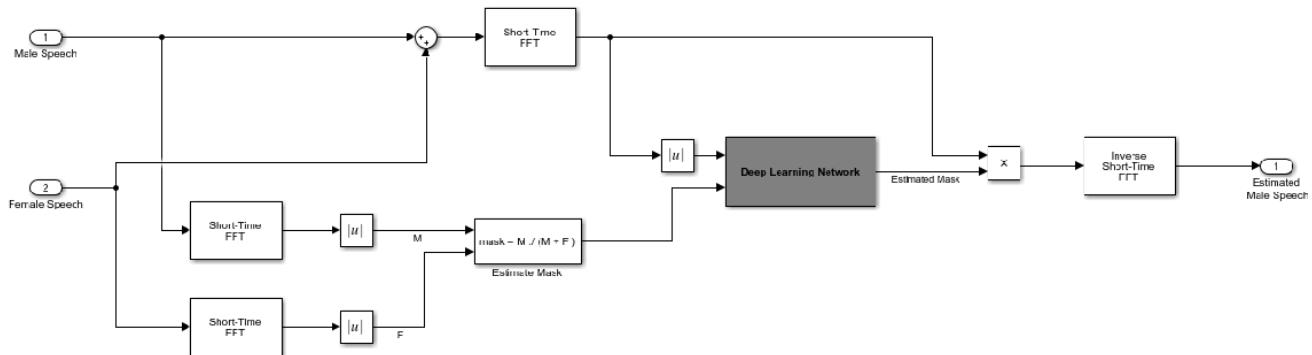
```
sound(mSpeech_Soft,Fs)
```

```
sound(fSpeech_Soft,Fs)
```

### 使用深度学习的掩膜估计

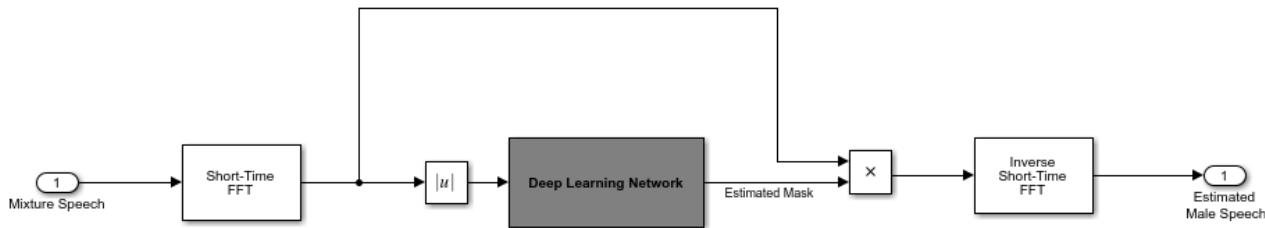
本示例中深度学习网络的目标是估计上述的理想软掩膜。网络估计对应于男性说话者的掩膜。女性说话者的掩膜直接派生自男性掩膜。

基本的深度学习训练方案如下所示。预测变量是混合（男性 + 女性）音频的幅值频谱。目标是与男性说话者对应的理想软掩膜。回归网络使用预测变量输入来最小化其输出和输入目标之间的均方误差。在输出端，使用输出幅值频谱和混音信号的相位将音频 STFT 转换回时域。



您可以使用短时傅里叶变换 (STFT) 将音频变换为频域，使用的窗口长度为 128 个样本、重叠部分为 127 个样本并使用 Hann 窗。通过丢弃对于负频率的频率样本，可以将频谱向量的大小减小到 65（因为时域语音信号是实信号，这样不会导致任何信息丢失）。预测变量输入由 20 个连续的 STFT 向量组成。输出为一个  $65 \times 20$  软掩膜。

您使用经过训练的网络来估计男性语音。经过训练的网络的输入是混合（男性 + 女性）语音音频。



## STFT 目标和预测变量

本节说明如何从训练数据集中生成目标和预测变量信号。

读入分别来自男性和女性说话者的约 400 秒语音的训练信号，采样频率为 4 kHz。低采样率用于加快训练速度。裁剪训练信号使其长度相同。

```

maleTrainingAudioFile = "MaleSpeech-16-4-mono-405secs.wav";
femaleTrainingAudioFile = "FemaleSpeech-16-4-mono-405secs.wav";

maleSpeechTrain = audioread(fullfile(netFolder,maleTrainingAudioFile));
femaleSpeechTrain = audioread(fullfile(netFolder,femaleTrainingAudioFile));

L = min(length(maleSpeechTrain),length(femaleSpeechTrain));
maleSpeechTrain = maleSpeechTrain(1:L);
femaleSpeechTrain = femaleSpeechTrain(1:L);
  
```

读入验证信号，该信号由分别来自男性和女性说话者的约 20 秒的语音组成，采样频率为 4 kHz。裁剪验证信号使其长度相同

```

maleValidationAudioFile = "MaleSpeech-16-4-mono-20secs.wav";
femaleValidationAudioFile = "FemaleSpeech-16-4-mono-20secs.wav";

maleSpeechValidate = audioread(fullfile(netFolder,maleValidationAudioFile));
femaleSpeechValidate = audioread(fullfile(netFolder,femaleValidationAudioFile));

L = min(length(maleSpeechValidate),length(femaleSpeechValidate));
maleSpeechValidate = maleSpeechValidate(1:L);
femaleSpeechValidate = femaleSpeechValidate(1:L);
  
```

将训练信号缩放到相同的功率。将验证信号缩放到相同的功率。

```

maleSpeechTrain = maleSpeechTrain/norm(maleSpeechTrain);
femaleSpeechTrain = femaleSpeechTrain/norm(femaleSpeechTrain);
ampAdj = max(abs([maleSpeechTrain;femaleSpeechTrain]));
maleSpeechTrain = maleSpeechTrain/ampAdj;
femaleSpeechTrain = femaleSpeechTrain/ampAdj;
  
```

```

maleSpeechValidate = maleSpeechValidate/norm(maleSpeechValidate);
femaleSpeechValidate = femaleSpeechValidate/norm(femaleSpeechValidate);
ampAdj      = max(abs([maleSpeechValidate;femaleSpeechValidate]));
maleSpeechValidate = maleSpeechValidate/ampAdj;
femaleSpeechValidate = femaleSpeechValidate/ampAdj;

```

创建训练和验证“鸡尾酒会”混音。

```

mixTrain = maleSpeechTrain + femaleSpeechTrain;
mixTrain = mixTrain / max(mixTrain);

mixValidate = maleSpeechValidate + femaleSpeechValidate;
mixValidate = mixValidate / max(mixValidate);

```

生成训练 STFT。

```

WindowLength = 128;
FFTLength   = 128;
OverlapLength = 128-1;
Fs          = 4000;
win         = hann(WindowLength,"periodic");

P_mix0 = stft(mixTrain, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
P_M   = abs(stft(maleSpeechTrain, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided'));
P_F   = abs(stft(femaleSpeechTrain, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided'));

```

对混音 STFT 取对数。通过均值和标准差对这些值进行归一化。

```

P_mix = log(abs(P_mix0) + eps);
MP   = mean(P_mix(:));
SP   = std(P_mix(:));
P_mix = (P_mix - MP) / SP;

```

生成验证 STFT。对混音 STFT 取对数。通过均值和标准差对这些值进行归一化。

```

P_Val_mix0 = stft(mixValidate, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
P_Val_M   = abs(stft(maleSpeechValidate, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided'));
P_Val_F   = abs(stft(femaleSpeechValidate, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided'));

P_Val_mix = log(abs(P_Val_mix0) + eps);
MP       = mean(P_Val_mix(:));
SP       = std(P_Val_mix(:));
P_Val_mix = (P_Val_mix - MP) / SP;

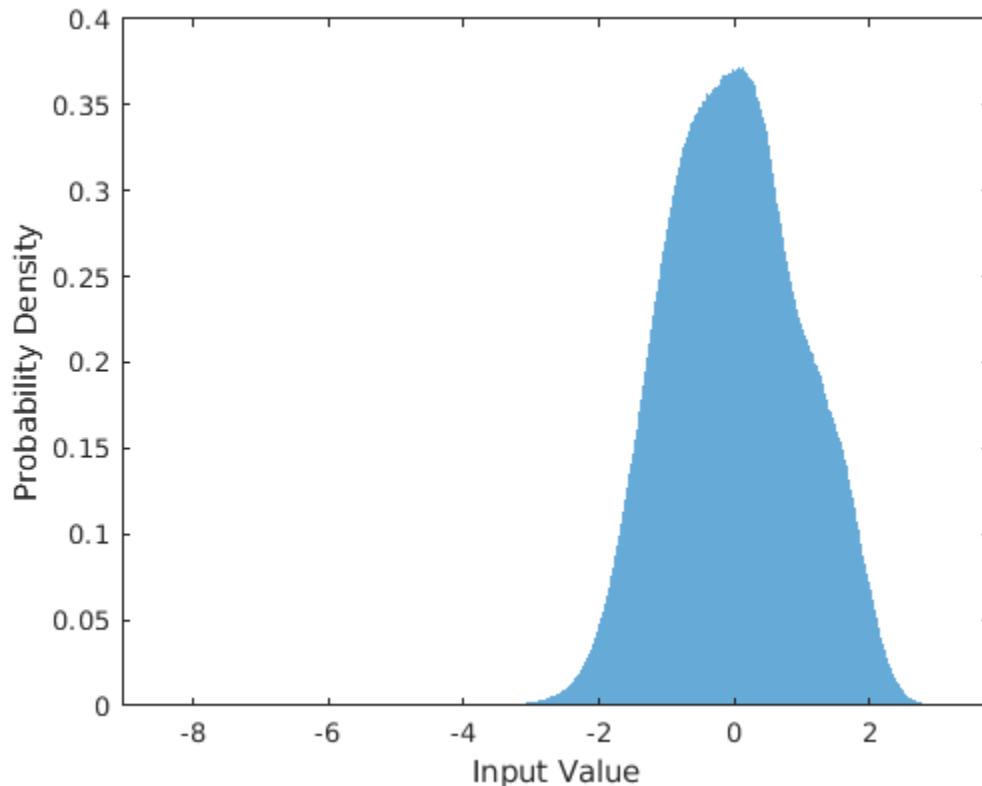
```

当网络的输入经过归一化且具有适度平滑的分布时，最易于训练神经网络。要检查数据分布是否平滑，请绘制训练数据的 STFT 值的直方图。

```

figure(6)
histogram(P_mix,"EdgeColor","none","Normalization","pdf")
xlabel("Input Value")
ylabel("Probability Density")

```



计算训练软掩膜。在训练网络时，使用此掩膜作为目标信号。

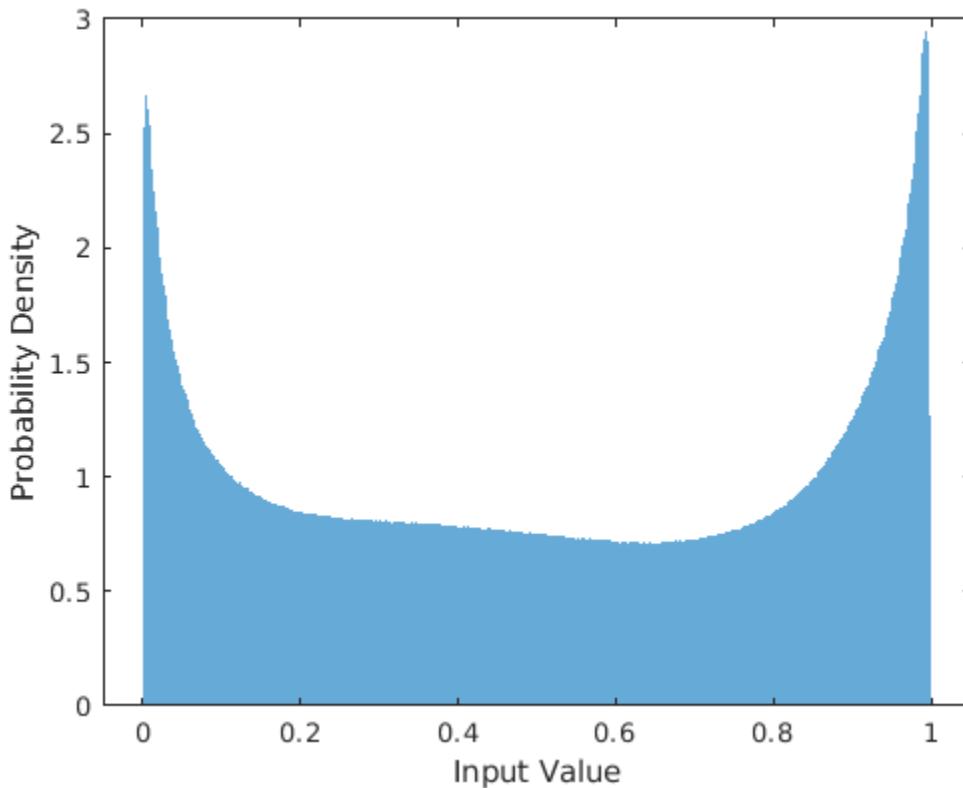
```
maskTrain = P_M ./ (P_M + P_F + eps);
```

计算验证软掩膜。使用此掩膜评估由经过训练的网络发出的掩膜。

```
maskValidate = P_Val_M ./ (P_Val_M + P_Val_F + eps);
```

要检查目标数据分布是否平滑，请绘制训练数据封装值的直方图。

```
figure(7)
histogram(maskTrain,"EdgeColor","none","Normalization","pdf")
xlabel("Input Value")
ylabel("Probability Density")
```



根据预测变量和目标信号创建大小为 (65, 20) 的分块。为了获得更多训练样本，在连续分块之间使用 10 个段作为重叠量。

```

seqLen      = 20;
seqOverlap  = 10;
mixSequences = zeros(1 + FFTLength/2,seqLen,1,0);
maskSequences = zeros(1 + FFTLength/2,seqLen,1,0);

loc = 1;
while loc < size(P_mix,2) - seqLen
    mixSequences(:,:,end+1) = P_mix(:,:,loc:loc+seqLen-1); %#ok
    maskSequences(:,:,end+1) = maskTrain(:,:,loc:loc+seqLen-1); %#ok
    loc                  = loc + seqOverlap;
end

```

根据验证预测变量和目标信号创建大小为 (65,20) 的分块。

```

mixValSequences = zeros(1 + FFTLength/2,seqLen,1,0);
maskValSequences = zeros(1 + FFTLength/2,seqLen,1,0);
seqOverlap      = seqLen;

loc = 1;
while loc < size(P_Val_mix,2) - seqLen
    mixValSequences(:,:,end+1) = P_Val_mix(:,:,loc:loc+seqLen-1); %#ok
    maskValSequences(:,:,end+1) = maskValidate(:,:,loc:loc+seqLen-1); %#ok
    loc                  = loc + seqOverlap;
end

```

重构训练和验证信号。

```
mixSequencesT = reshape(mixSequences, [1 1 (1 + FFTLength/2) * seqLen size(mixSequences,4)]);
mixSequencesV = reshape(mixValSequences, [1 1 (1 + FFTLength/2) * seqLen size(mixValSequences,4)]);
maskSequencesT = reshape(maskSequences, [1 1 (1 + FFTLength/2) * seqLen size(maskSequences,4)]);
maskSequencesV = reshape(maskValSequences,[1 1 (1 + FFTLength/2) * seqLen size(maskValSequences,4)]);
```

## 定义深度学习网络

定义网络的各层。将输入大小指定为  $1 \times 1 \times 1300$  大小的图像。定义两个隐藏的全连接层，每个层有 1300 个神经元。每个隐藏的全连接层后跟一个 sigmoid 层。批量归一化层对输出的均值和标准差进行归一化。添加一个包含 1300 个神经元的全连接层，后跟一个回归层。

```
numNodes = (1 + FFTLength/2) * seqLen;
layers = [ ...
    imageInputLayer([1 1 (1 + FFTLength/2)*seqLen],"Normalization","None")
    fullyConnectedLayer(numNodes)
    BiasedSigmoidLayer(6)
    batchNormalizationLayer
    dropoutLayer(0.1)

    fullyConnectedLayer(numNodes)
    BiasedSigmoidLayer(6)
    batchNormalizationLayer
    dropoutLayer(0.1)

    fullyConnectedLayer(numNodes)
    BiasedSigmoidLayer(0)

    regressionLayer
];
```

指定网络的训练选项。将 **MaxEpochs** 设置为 3 以便基于训练数据对网络进行 3 轮训练。将 **MiniBatchSize** 设置为 64 以便网络可以一次查看 64 个训练信号。将 **Plots** 设置为 **training-progress** 以生成显示训练进度随着迭代次数增加而变化的图。将 **Verbose** 设置为 **false** 以禁止将对应于图中所示数据的表输出打印到命令行窗口中。将 **Shuffle** 设置为 **every-epoch** 以便在每轮开始时对训练序列进行乱序处理。将 **LearnRateSchedule** 设置为 **piecewise** 以便每经过一定数量的轮次 (1) 后，按指定的因子 (0.1) 降低学习率。将 **ValidationData** 设置为验证预测变量和目标。设置 **ValidationFrequency** 以便每轮计算一次验证均方误差。此示例使用自适应矩估计 (ADAM) 求解器。

```
maxEpochs = 3;
miniBatchSize = 64;

options = trainingOptions("adam", ...
    "MaxEpochs",maxEpochs, ...
    "MiniBatchSize",miniBatchSize, ...
    "SequenceLength","longest", ...
    "Shuffle","every-epoch",...
    "Verbose",0, ...
    "Plots","training-progress",...
    "ValidationFrequency",floor(size(mixSequencesT,4)/miniBatchSize),...
    "ValidationData",{mixSequencesV,maskSequencesV},...
    "LearnRateSchedule","piecewise",...
```

```
"LearnRateDropFactor",0.9, ...
"LearnRateDropPeriod",1);
```

### 训练深度学习网络

使用 `trainNetwork` 用指定的训练选项和层架构训练网络。由于训练集很大，训练过程可能需要几分钟。要加载预训练的网络，请将 `doTraining` 设置为 `false`。

```
doTraining = true;
if doTraining
    CocktailPartyNet = trainNetwork(mixSequencesT,maskSequencesT,layers,options);
else
    s = load("CocktailPartyNet.mat");
    CocktailPartyNet = s.CocktailPartyNet;
end
```

将验证预测变量传递给网络。输出是估计的掩膜。重构估计的掩膜。

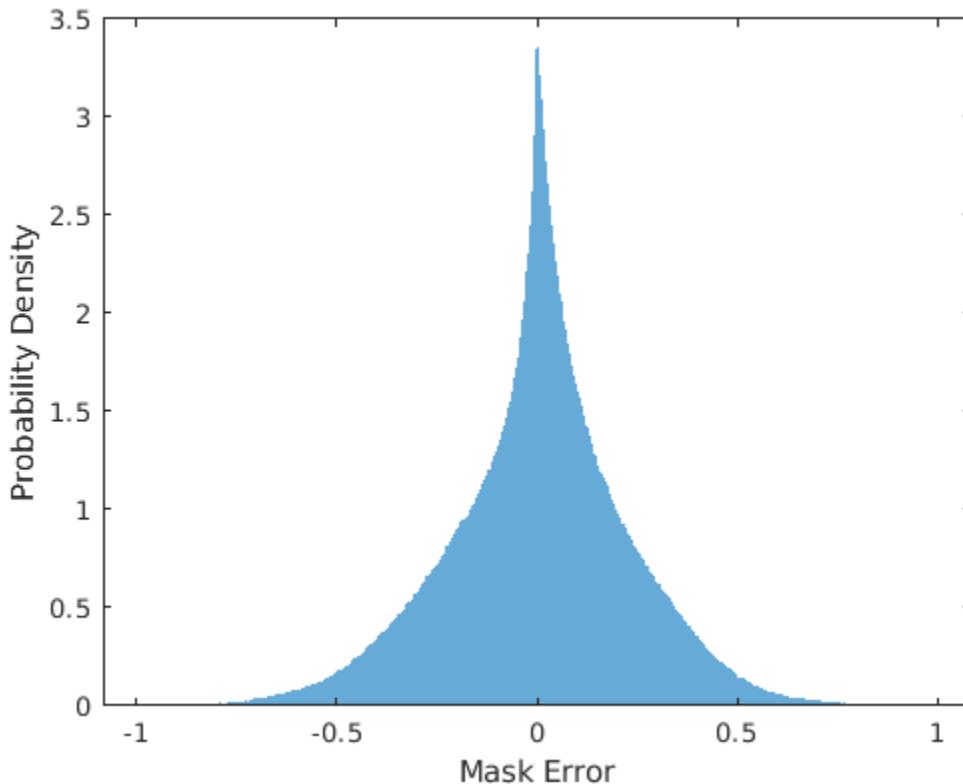
```
estimatedMasks0 = predict(CocktailPartyNet,mixSequencesV);

estimatedMasks0 = estimatedMasks0.';
estimatedMasks0 = reshape(estimatedMasks0,1 + FFTLength/2,numel(estimatedMasks0)/(1 + FFTLength/2));
```

### 评估深度学习网络

绘制实际掩膜和预期掩膜之间误差的直方图。

```
figure(8)
histogram(maskValSequences(:) - estimatedMasks0(:),"EdgeColor","none","Normalization","pdf")
xlabel("Mask Error")
ylabel("Probability Density")
```



### 评估软掩膜估计

估计男声和女声软掩膜。通过为软掩膜设置阈值来估计男声和女声二值掩膜。

```
SoftMaleMask = estimatedMasks0;
SoftFemaleMask = 1 - SoftMaleMask;
```

缩短混音 STFT 以匹配估计的掩膜的大小。

```
P_Val_mix0 = P_Val_mix0(:,1:size(SoftMaleMask,2));
```

将混音 STFT 乘以男声软掩膜，得到估计的男性语音 STFT。

```
P_Male = P_Val_mix0 .* SoftMaleMask;
```

使用 ISTFT 获得估计的男性音频信号。缩放音频。

```
maleSpeech_est_soft = istft(P_Male, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'ConjugateSymmetric', true, ...
    'FrequencyRange', 'onesided');
maleSpeech_est_soft = maleSpeech_est_soft / max(abs(maleSpeech_est_soft));
```

可视化估计的和原始的男性语音信号。收听估计的软掩膜男性语音。

```
range = (numel(win):numel(maleSpeech_est_soft)-numel(win));
t = range * (1/Fs);
```

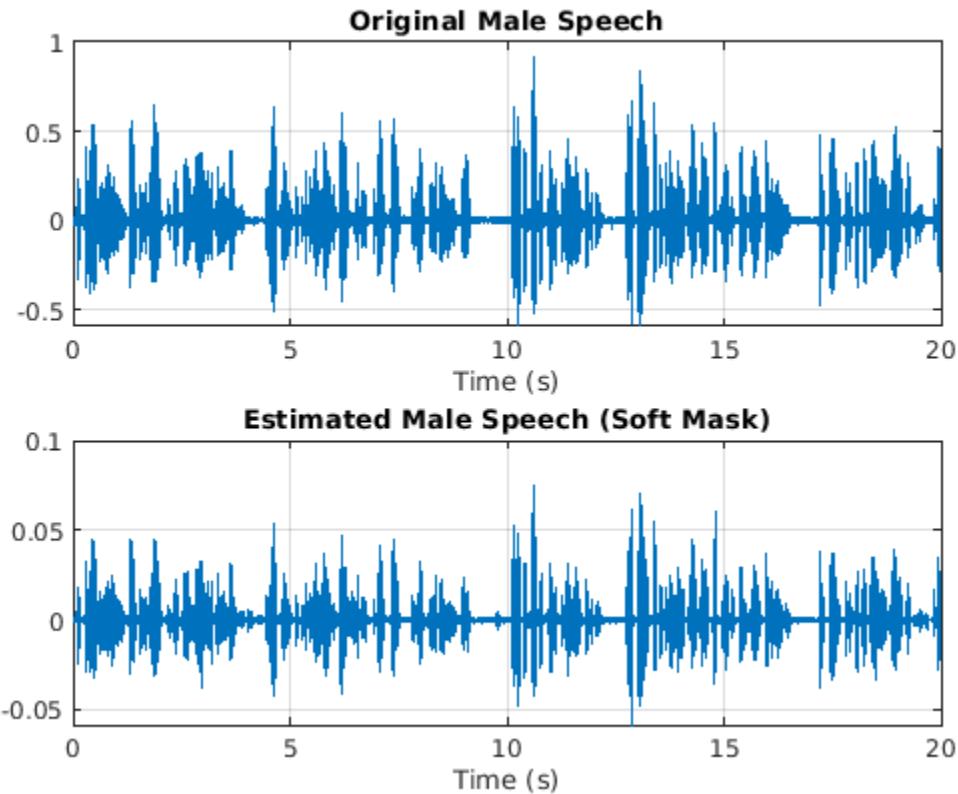
```
figure(9)
```

```

subplot(2,1,1)
plot(t,maleSpeechValidate(range))
title("Original Male Speech")
xlabel("Time (s)")
grid on

subplot(2,1,2)
plot(t,maleSpeech_est_soft(range))
xlabel("Time (s)")
title("Estimated Male Speech (Soft Mask)")
grid on

```



```
sound(maleSpeech_est_soft(range),Fs)
```

将混音 STFT 乘以女性软掩膜，得到估计的女性语音 STFT。使用 ISTFT 获得估计的男性音频信号。缩放音频。

```

P_Female = P_Val_mix0 .* SoftFemaleMask;

femaleSpeech_est_soft = istft(P_Female, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'ConjugateSymmetric', true, ...
    'FrequencyRange', 'onesided');
femaleSpeech_est_soft = femaleSpeech_est_soft / max(femaleSpeech_est_soft);

```

可视化估计的和原始的女性语音信号。收听估计的女性语音。

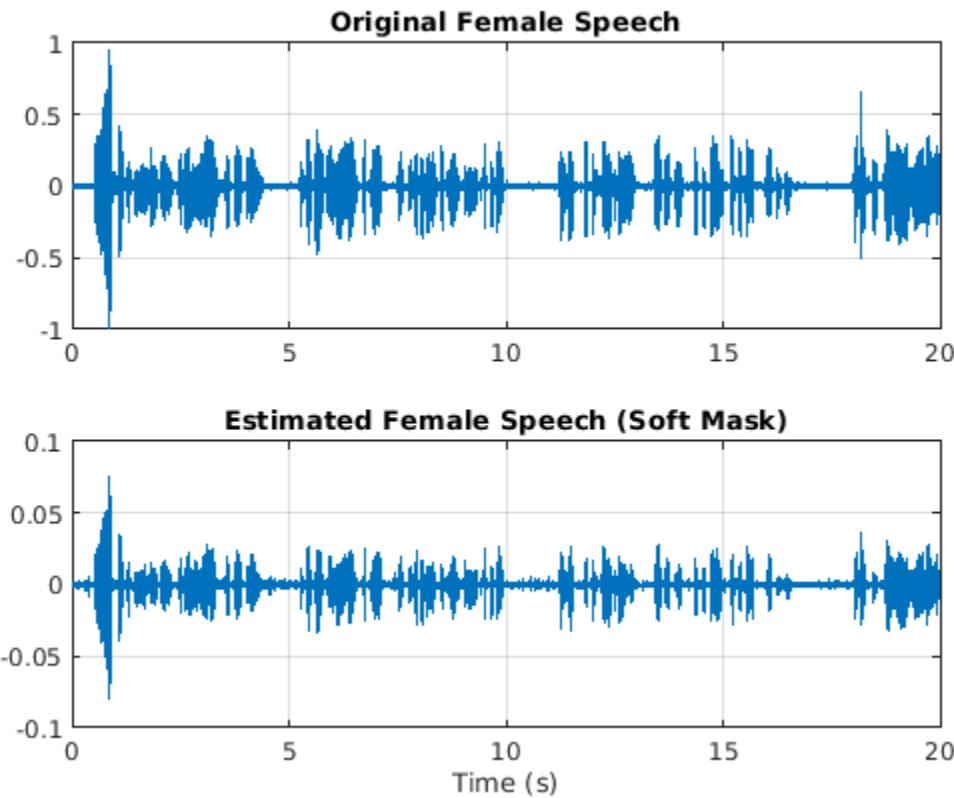
```
range = (numel(win):numel(maleSpeech_est_soft) - numel(win));
t = range * (1/Fs);
```

```

figure(10)
subplot(2,1,1)
plot(t,femaleSpeechValidate(range))
title("Original Female Speech")
grid on

subplot(2,1,2)
plot(t,femaleSpeech_est_soft(range))
xlabel("Time (s)")
title("Estimated Female Speech (Soft Mask)")
grid on

```



```
sound(femaleSpeech_est_soft(range),Fs)
```

### 评估二值掩膜估计

通过为软掩膜设置阈值来估计男声和女声二值掩膜。

```

HardMaleMask = SoftMaleMask >= 0.5;
HardFemaleMask = SoftMaleMask < 0.5;

```

将混音 STFT 乘以男声二值掩膜，得到估计的男性语音 STFT。使用 ISTFT 获得估计的男性音频信号。缩放音频。

```

P_Male = P_Val_mix0 .* HardMaleMask;

maleSpeech_est_hard = istft(P_Male, 'Window', win, 'OverlapLength', OverlapLength, ...

```

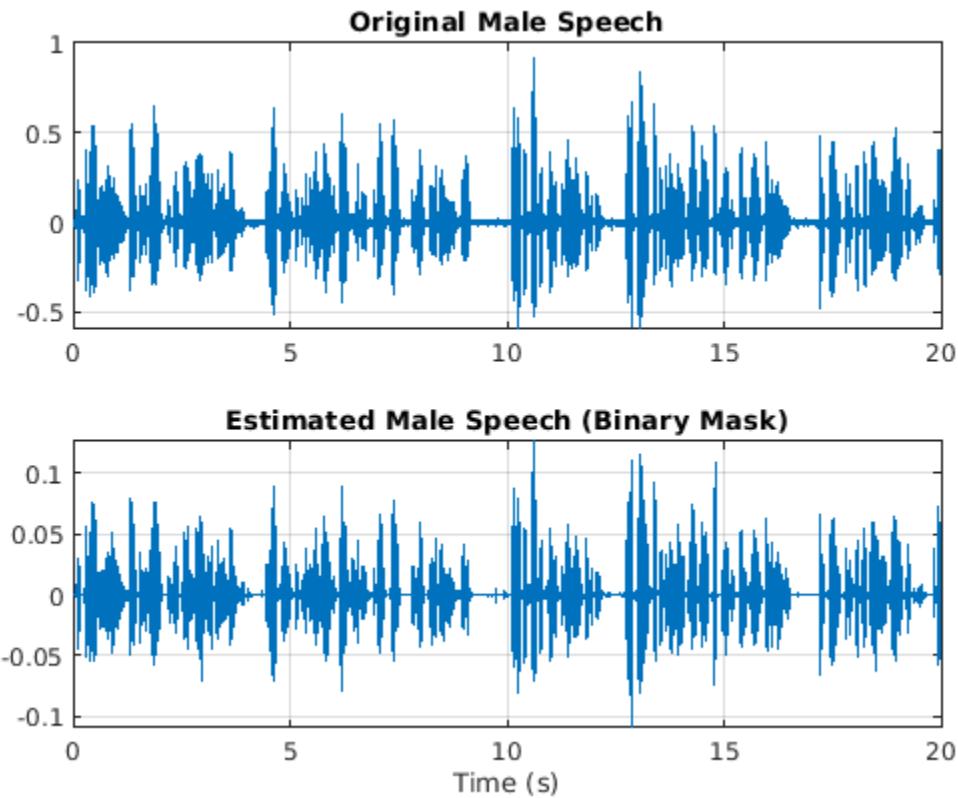
```
'FFTLength', FFTLength, 'ConjugateSymmetric', true, ...
'FrequencyRange', 'onesided');
maleSpeech_est_hard = maleSpeech_est_hard / max(maleSpeech_est_hard);
```

可视化估计的和原始的男性语音信号。收听估计的二值掩膜男性语音。

```
range = (numel(win):numel(maleSpeech_est_soft)-numel(win));
t = range * (1/Fs);
```

```
figure(11)
subplot(2,1,1)
plot(t,maleSpeechValidate(range))
title("Original Male Speech")
grid on

subplot(2,1,2)
plot(t,maleSpeech_est_hard(range))
xlabel("Time (s)")
title("Estimated Male Speech (Binary Mask)")
grid on
```



```
sound(maleSpeech_est_hard(range),Fs)
```

将混音 STFT 乘以女声二值掩膜，得到估计的男性语音 STFT。使用 ISTFT 获得估计的男性音频信号。缩放音频。

```
P_Female = P_Val_mix0 .* HardFemaleMask;
```

```

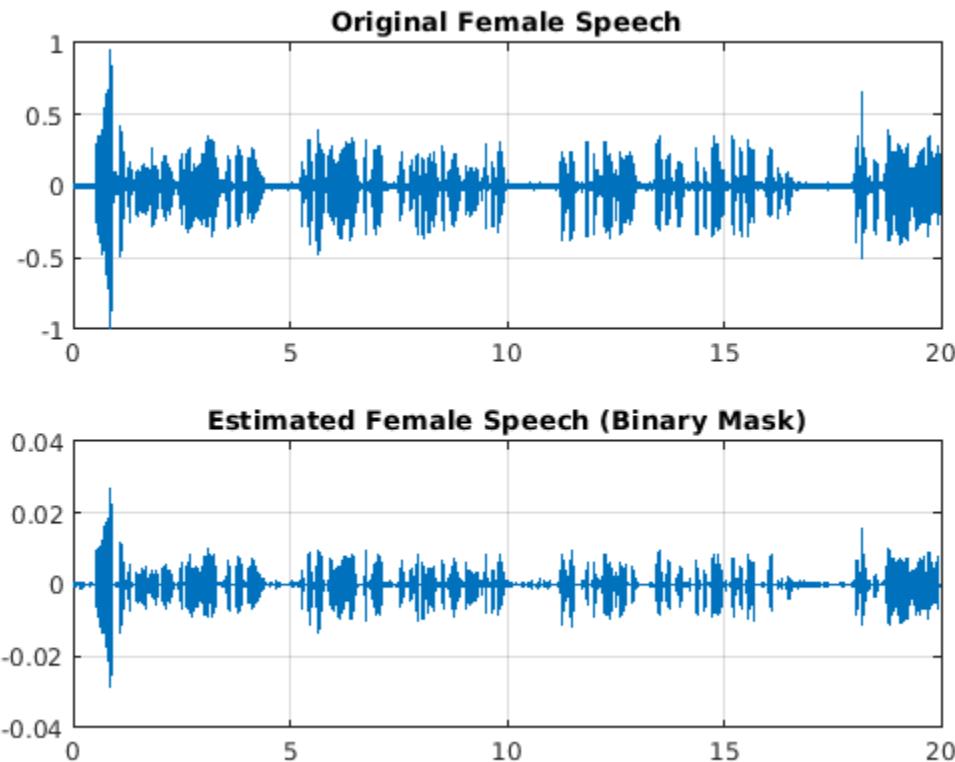
femaleSpeech_est_hard = istft(P_Female, 'Window', win, 'OverlapLength', OverlapLength, ...
    'FFTLength', FFTLength, 'ConjugateSymmetric', true, ...
    'FrequencyRange', 'onesided');
femaleSpeech_est_hard = femaleSpeech_est_hard / max(femaleSpeech_est_hard);

可视化估计的和原始的女性语音信号。收听估计的女性语音。
range = (numel(win):numel(maleSpeech_est_soft)-numel(win));
t = range * (1/Fs);

figure(12)
subplot(2,1,1)
plot(t,femaleSpeechValidate(range))
title("Original Female Speech")
grid on

subplot(2,1,2)
plot(t,femaleSpeech_est_hard(range))
title("Estimated Female Speech (Binary Mask)")
grid on

```



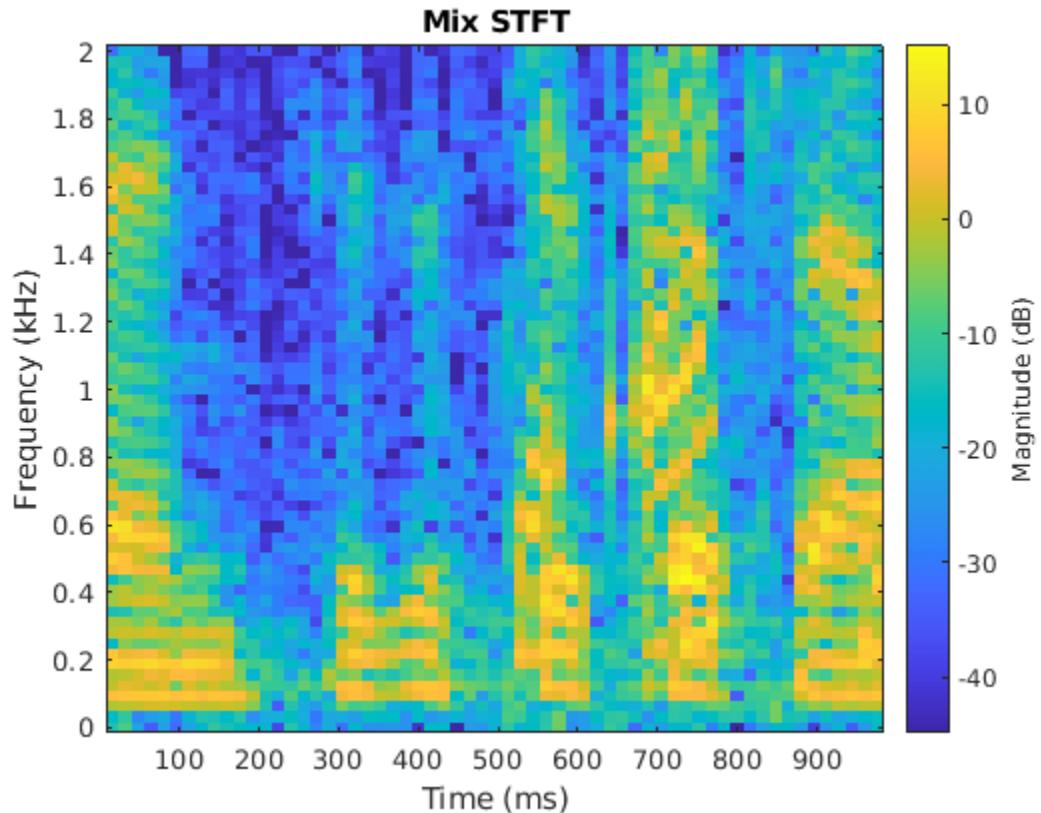
```
sound(femaleSpeech_est_hard(range),Fs)
```

对混音、原始女性语音和男性语音以及估计的女性和男性语音，分别比较其一秒语音段的 STFT。

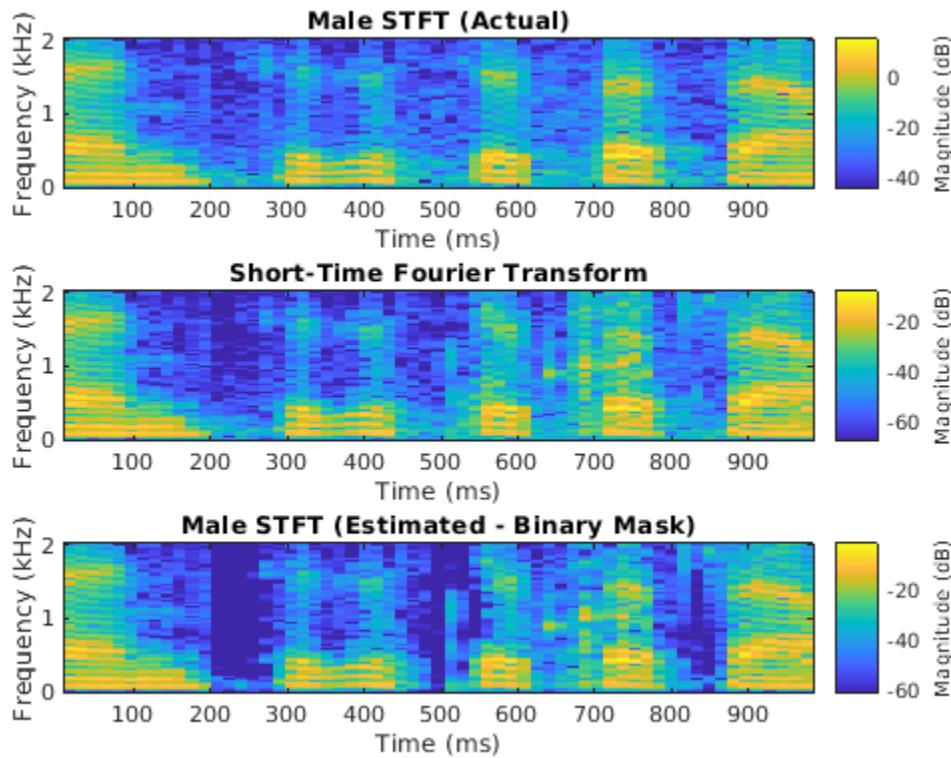
```
range = 7e4:7.4e4;
```

```
figure(13)
```

```
stft(mixValidate(range), Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Mix STFT")
```



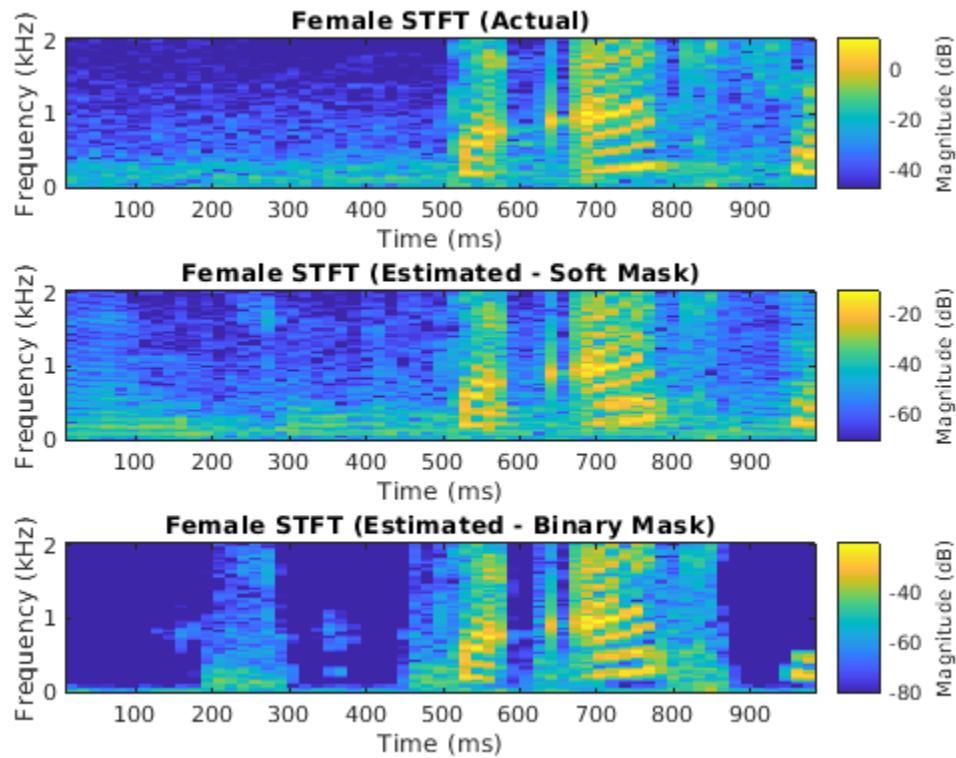
```
figure(14)
subplot(3,1,1)
stft(maleSpeechValidate(range),Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Male STFT (Actual)")
subplot(3,1,2)
stft(maleSpeech_est_soft(range),Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
subplot(3,1,3)
stft(maleSpeech_est_hard(range),Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Male STFT (Estimated - Binary Mask)")
```



```

figure(15)
subplot(3,1,1)
stft(femaleSpeechValidate(range),Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Female STFT (Actual)")
subplot(3,1,2)
stft(femaleSpeech_est_soft(range),Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Female STFT (Estimated - Soft Mask)")
subplot(3,1,3)
stft(femaleSpeech_est_hard(range),Fs, 'Window', win, 'OverlapLength', 64, ...
    'FFTLength', FFTLength, 'FrequencyRange', 'onesided');
title("Female STFT (Estimated - Binary Mask)")

```



## 参考资料

[1] "Probabilistic Binary-Mask Cocktail-Party Source Separation in a Convolutional Deep Neural Network", Andrew J.R. Simpson, 2015.

## 另请参阅

[trainingOptions](#) | [trainNetwork](#)

## 详细信息

- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 使用深度学习检测噪声中的语音活动

此示例说明如何使用深度学习在低信噪比环境中检测语音区域。该示例使用语音命令数据集来训练一个双向长短期记忆 (BiLSTM) 网络来检测语音活动。

## 简介

语音活动检测是许多音频系统的重要组成部分，例如自动语音识别和说话者识别。在低信噪比 (SNR) 情况下，语音活动检测尤其具有挑战性，因为这种情况下语音会受到噪声的干扰。

本示例使用长短期记忆 (LSTM) 网络，这是一种循环神经网络 (RNN)，非常适合研究序列和时序数据。LSTM 网络可以学习序列的时间步之间的长期相关性。LSTM 层 (`lstmLayer`) 可以前向分析时间序列，而双向 LSTM 层 (`bilstmLayer`) 可以前向和后向分析时间序列。此示例使用双向 LSTM 层。

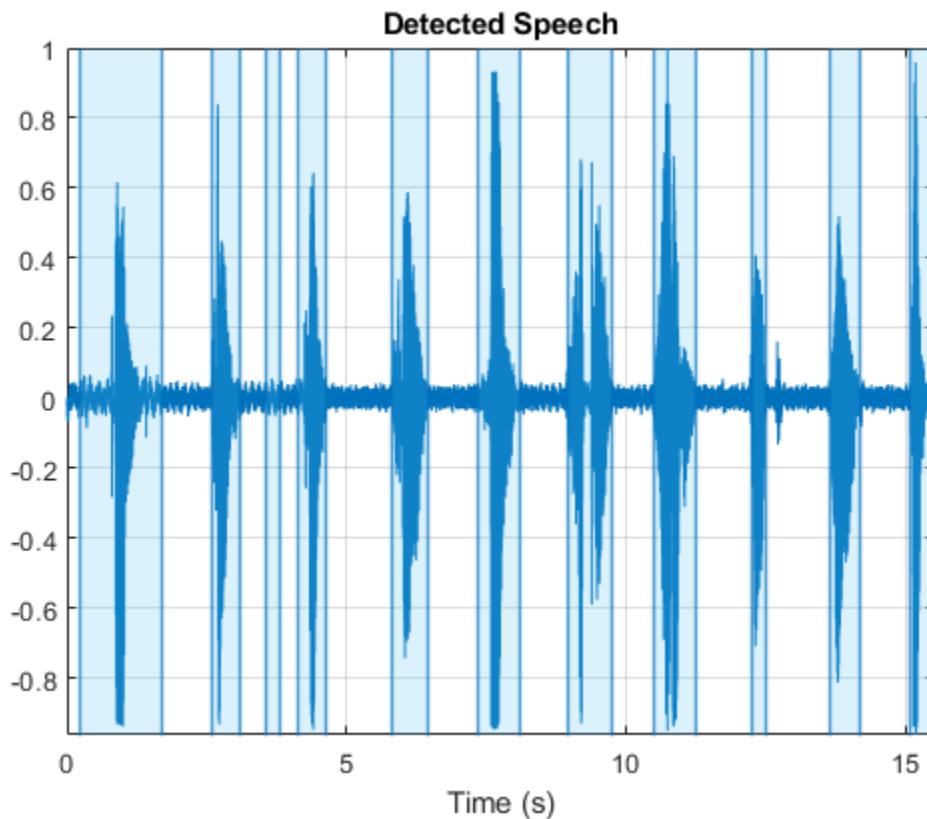
此示例使用包含频谱特性与谐波比指标的特征序列训练一个用于语音活动检测的双向 LSTM 网络。

在高 SNR 情况下，传统的语音检测算法性能就足够好了。读入由单词组成的音频文件，单词之间有停顿。以 16 kHz 的频率对音频进行重采样。收听音频。

```
fs = 16e3;
[speech,fileFs] = audioread('Counting-16-44p1-mono-15secs.wav');
speech = resample(speech,fs,fileFs);
speech = speech/max(abs(speech));
sound(speech,fs)
```

使用 `detectSpeech` (Audio Toolbox) 函数定位语音区域。`detectSpeech` 函数可正确识别所有语音区域。

```
win = hamming(50e-3 * fs,'periodic');
detectSpeech(speech,fs,'Window',win)
```



用 SNR 为 -20 dB 的洗衣机噪声破坏音频信号。收听损坏的音频。

```
[noise,fileFs] = audioread('WashingMachine-16-8-mono-200secs.mp3');
noise = resample(noise,fs,fileFs);

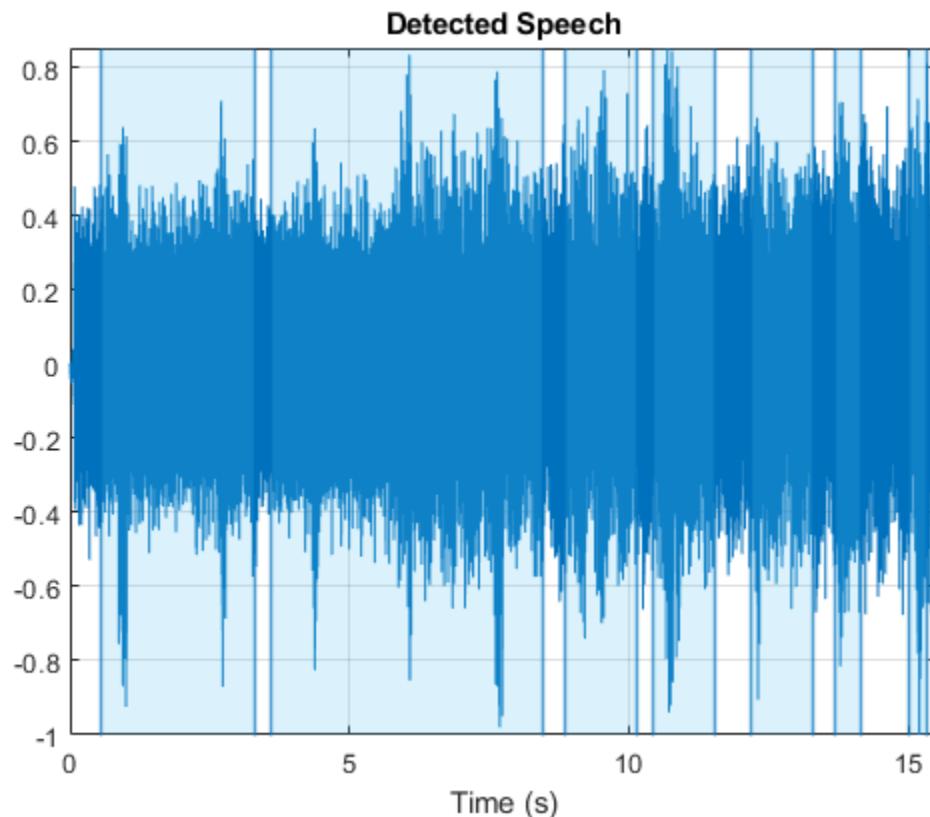
SNR = -20;
noiseGain = 10^(-SNR/20) * norm(speech) / norm(noise);

noisySpeech = speech + noiseGain*noise(1:numel(speech));
noisySpeech = noisySpeech./max(abs(noisySpeech));

sound(noisySpeech,fs)
```

对含噪音音频信号调用 `detectSpeech`。在 SNR 很低的情况下，该函数无法检测到语音区域。

```
detectSpeech(noisySpeech,fs,'Window',win)
```



下载并加载一个预训练网络和一个已配置的 `audioFeatureExtractor` (Audio Toolbox) 对象。该网络经过训练，能够在低 SNR 环境中基于由 `audioFeatureExtractor` 对象输出的特征检测语音。

```

url = 'http://ssd.mathworks.com/supportfiles/audio/VoiceActivityDetection.zip';
downloadNetFolder = tempdir;
netFolder = fullfile(downloadNetFolder,'VoiceActivityDetection');
if ~exist(netFolder,'dir')
    disp('Downloading pretrained network (1 file - 8 MB) ...')
    unzip(url,downloadNetFolder)
end
load(fullfile(netFolder,'voiceActivityDetectionExample.mat'));
speechDetectNet

speechDetectNet =
SeriesNetwork with properties:
    Layers: [6×1 nnet.cnn.layer.Layer]
    InputNames: {'sequenceinput'}
    OutputNames: {'classoutput'}

afe
afe =
audioFeatureExtractor with properties:

Properties

```

```

    Window: [256x1 double]
    OverlapLength: 128
    SampleRate: 16000
    FFTLength: []
    SpectralDescriptorInput: 'linearSpectrum'

Enabled Features
spectralCentroid, spectralCrest, spectralEntropy, spectralFlux, spectralKurtosis, spectralRolloffPoint
spectralSkewness, spectralSlope, harmonicRatio

Disabled Features
linearSpectrum, melSpectrum, barkSpectrum, erbSpectrum, mfcc, mfccDelta
mfccDeltaDelta, gtcc, gtccDelta, gtccDeltaDelta, spectralDecrease, spectralFlatness
spectralSpread, pitch

```

To extract a feature, set the corresponding property to true.  
For example, `obj.mfcc = true`, adds mfcc to the list of enabled features.

从语音数据中提取特征，然后对其进行归一化。调整特征方向，使时间跨列。

```

features = extract(afe,noisySpeech);
features = (features - mean(features,1)) ./ std(features,[],1);
features = features';

```

使特征通过语音检测网络，以将每个特征向量分类为是否属于一个语音帧。

```
decisionsCategorical = classify(speechDetectNet,features);
```

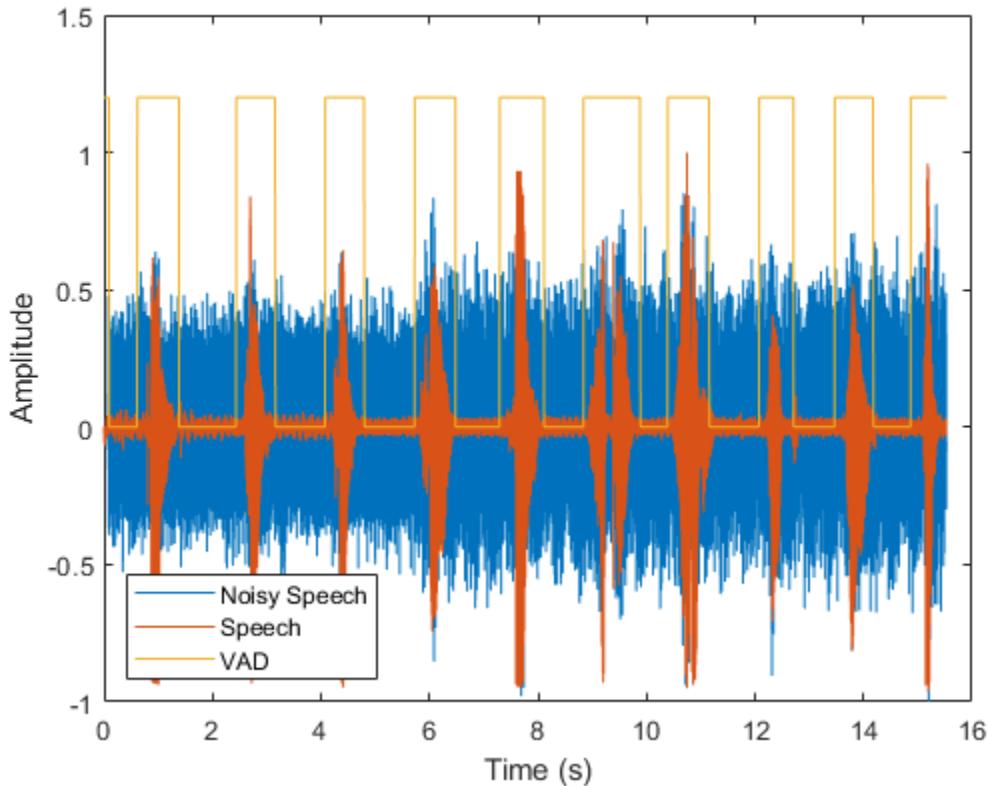
每个决策都对应一个由 `audioFeatureExtractor` 分析的分析窗口。复制决策，使它们与音频采样一一对应。对语音、含噪语音和 VAD 决策进行绘图。

```

decisionsWindow = 1.2*(double(decisionsCategorical)-1);
decisionsSample = [repelem(decisionsWindow(1),numel(afe.Window)), ...
    repelem(decisionsWindow(2:end),numel(afe.Window)-afe.OverlapLength)];

t = (0:numel(decisionsSample)-1)/afe.SampleRate;
plot(t,noisySpeech(1:numel(t)), ...
    t,speech(1:numel(t)), ...
    t,decisionsSample);
xlabel('Time (s)')
ylabel('Amplitude')
legend('Noisy Speech','Speech','VAD','Location','southwest')

```



您还可以在流式环境中使用经过训练的 VAD 网络。要模拟流式环境，首先将语音和含噪信号保存为 WAV 文件。要模拟流式输入，您将从文件中读取帧，并以所需的 SNR 混合它们。

```
audiowrite('Speech.wav',speech,fs)
audiowrite('Noise.wav',noise,fs)
```

要将 VAD 网络应用于流音频，您必须在延迟和准确度之间进行权衡。定义噪声演示中流语音活动检测的参数。您可以设置测试的持续时间、馈送到网络的序列长度、序列跳跃长度和要测试的 SNR。一般情况下，增加序列长度会提高准确度，但也会增加滞后。您还可以选择输出到设备的信号作为原始信号或含噪信号。

```
testDuration = 20 ;  
sequenceLength = 400 ;  
sequenceHop = 20 ;  
SNR = -20 ;  
noiseGain = 10^(-SNR/20) * norm(speech) / norm(noise);  
  
signalToListenTo = "noisy" ;
```

调用流式演示辅助函数，观察 VAD 网络对流音频的性能。您使用实时控制设置的参数不会中断流示例。流式演示完成后，您可以修改演示的参数，然后再次运行流式演示。您可以在支持函数（第 14-0 页）中找到流式演示的代码。

```
helperStreamingDemo(speechDetectNet,afe, ...
    'Speech.wav','Noise.wav', ...
    testDuration,sequenceLength,sequenceHop,signalToListenTo,noiseGain);
```

此示例的其余部分介绍 VAD 网络的训练和评估。

### 训练和评估 VAD 网络

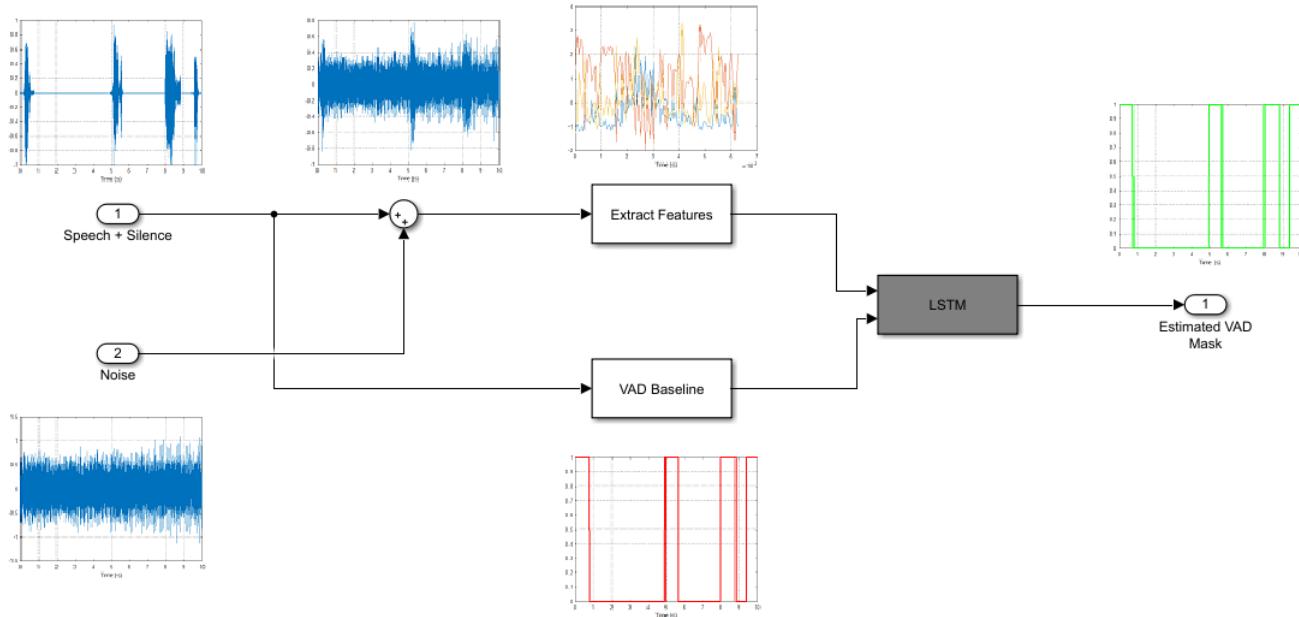
训练：

- 1 创建一个 **audioDatastore** (Audio Toolbox)，它指向用于训练 LSTM 网络的音频语音文件。
- 2 创建由语音段组成的训练信号，这些语音段由持续时间不同的静默段分隔开。
- 3 用洗衣机噪声损坏语音加静默信号 ( $\text{SNR} = -10 \text{ dB}$ )。
- 4 从含噪信号中提取由频谱特性和谐波比组成的特征序列。
- 5 使用特征序列训练 LSTM 网络以识别语音活动区域。

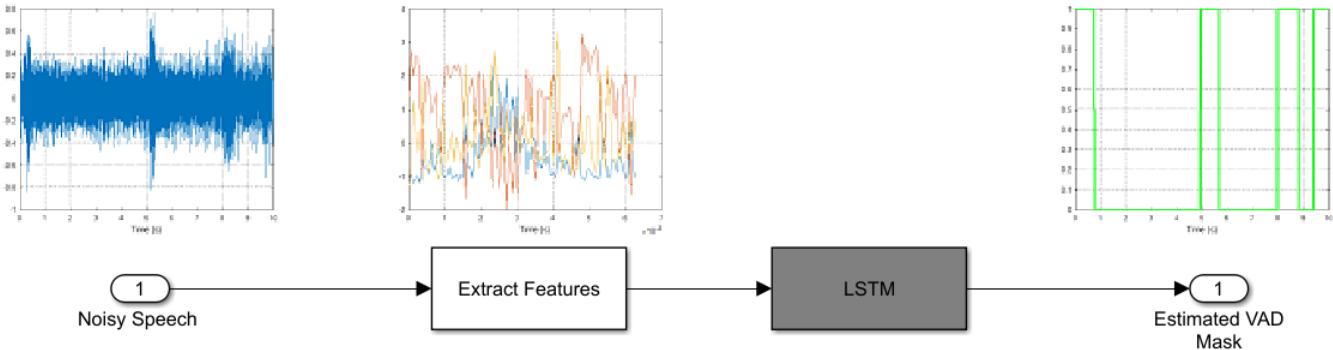
预测：

- 1 创建一个由语音文件组成的 **audioDatastore**，用于测试经过训练的网络，并创建一个测试信号，该信号由以静默段分隔的语音组成。
- 2 用洗衣机噪声损坏测试信号 ( $\text{SNR} = -10 \text{ dB}$ )。
- 3 从含噪测试信号中提取特征序列。
- 4 通过对测试特征应用经过训练的网络来识别语音活动区域。
- 5 将网络的准确度与信号加静默测试信号中的语音活动基线进行比较。

这是训练过程的草图。



这是预测过程的草图。您使用经过训练的网络进行预测。



### 加载语音命令数据集

下载并提取 Google Speech Commands Dataset [1] (第 14-0 页)。

```

url = 'https://ssd.mathworks.com/supportfiles/audio/google_speech.zip';

downloadFolder = tempdir;
datasetFolder = fullfile(downloadFolder,'google_speech');

if ~exist(datasetFolder,'dir')
    disp('Downloading Google speech commands data set (1.9 GB)...')
    unzip(url,downloadFolder)
end
  
```

Downloading Google speech commands data set (1.9 GB)...

创建一个指向该训练数据集的 **audioDatastore** (Audio Toolbox)。

```
adsTrain = audioDatastore(fullfile(datasetFolder, 'train'), "Includesubfolders",true);
```

创建一个指向该验证数据集的 **audioDatastore** (Audio Toolbox)。

```
adsValidation = audioDatastore(fullfile(datasetFolder, 'validation'), "Includesubfolders",true);
```

### 创建语音加静默训练信号

使用 **read** (Audio Toolbox) 读取音频文件的内容。从 **adsInfo** 结构体获取采样率。

```
[data,adsInfo] = read(adsTrain);
Fs = adsInfo.SampleRate;
```

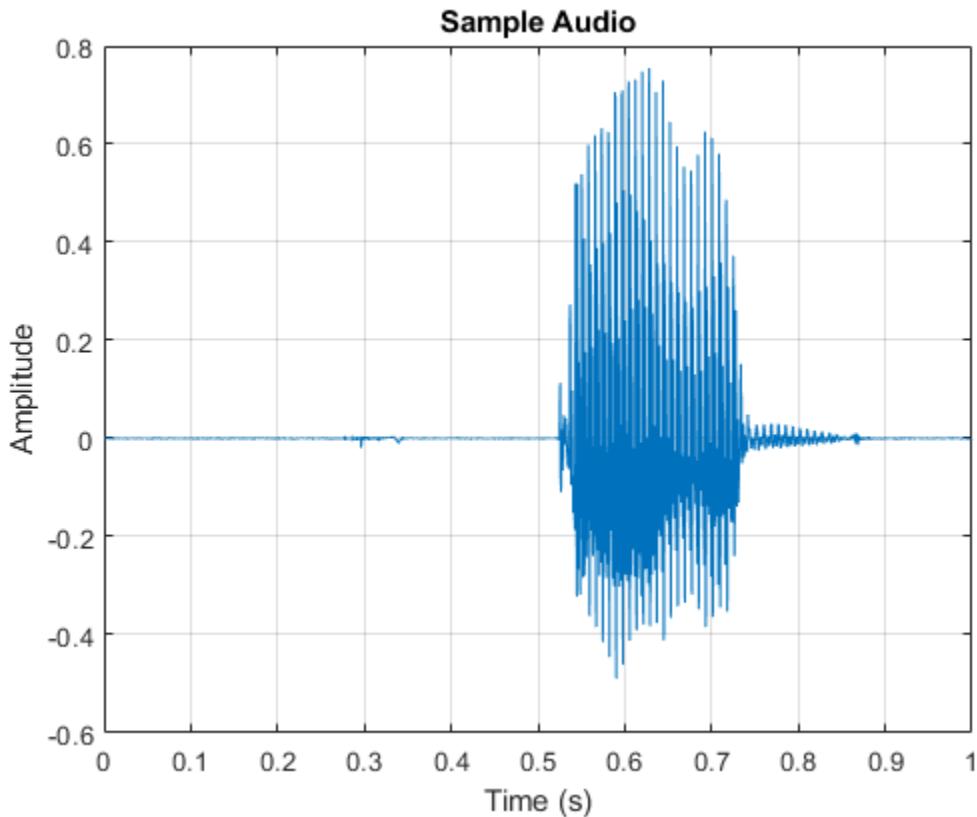
使用 **sound** 命令收听音频信号。

```
sound(data,Fs)
```

绘制音频信号。

```
timeVector = (1/Fs) * (0:numel(data)-1);
plot(timeVector,data)
```

```
ylabel("Amplitude")
xlabel("Time (s)")
title("Sample Audio")
grid on
```

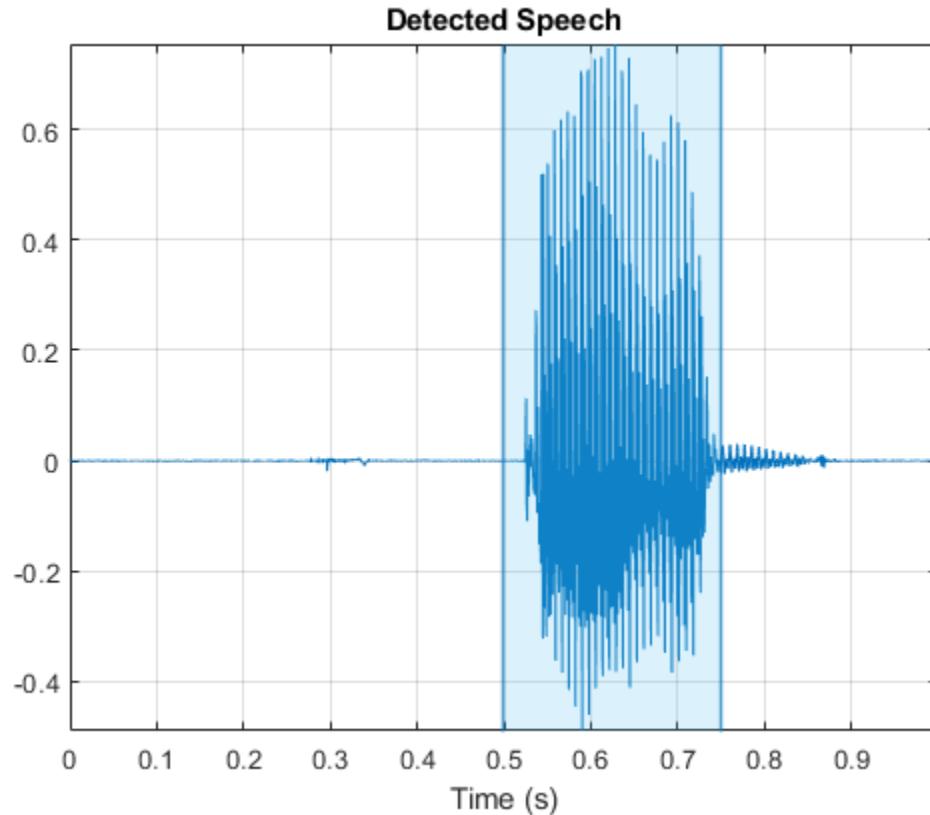


该信号具有不包含有用语音信息的非语音部分（静默、背景噪声等）。此示例使用 **detectSpeech** (Audio Toolbox) 函数消除静默。

提取数据的有用部分。定义一个以 50 毫秒为周期的 Hamming 窗进行分析。不带任何输出参数调用 **detectSpeech** 来绘制检测到的语音区域。再次调用 **detectSpeech** 以返回检测到的语音的索引。隔离检测到的语音区域，然后使用 **sound** 命令收听音频。

```
win = hamming(50e-3 * Fs,'periodic');

detectSpeech(data,Fs,'Window',win);
```



```
speechIndices = detectSpeech(data,Fs,'Window',win);
```

```
sound(data(speechIndices(1,1):speechIndices(1,2)),Fs)
```

**detectSpeech** 函数返回紧紧围绕检测到的语音区域的索引。根据经验确定，在任一侧将检测到的语音的索引扩展若干帧（对于此示例为五帧）可提高最终模型的性能。将语音索引扩展五帧，然后收听语音。

```
speechIndices(1,1) = max(speechIndices(1,1) - 5*numel(win),1);
speechIndices(1,2) = min(speechIndices(1,2) + 5*numel(win),numel(data));
```

```
sound(data(speechIndices(1,1):speechIndices(1,2)),Fs)
```

重置训练数据存储并打乱数据存储中文件的顺序。

```
reset(adsTrain)
adsTrain = shuffle(adsTrain);
adsValidation = shuffle(adsValidation);
```

**detectSpeech** 函数计算基于统计的阈值来确定语音区域。通过直接指定阈值，您可以跳过阈值计算并加快 **detectSpeech** 函数的执行速度。要确定数据集的阈值，请对文件的采样调用 **detectSpeech** 并获取它计算的阈值。取阈值的均值。

```
TM = [];
for index1 = 1:500
    data = read(adsTrain);
    [~,T] = detectSpeech(data,Fs,'Window',win);
    TM = [TM;T];
end
```

```
T = mean(TM);
```

```
reset(adsTrain)
```

通过合并来自训练数据集的多个语音文件，创建一个 1000 秒的训练信号。使用 `detectSpeech` 删除每个文件中不需要的部分。在语音段之间插入一段随机长度的静默时间。

预分配训练信号。

```
duration = 2000*Fs;
audioTraining = zeros(duration,1);
```

预分配语音活动训练掩膜。掩膜中的值 1 对应于具有语音活动的区域中的样本。值 0 对应于没有语音活动的区域。

```
maskTraining = zeros(duration,1);
```

指定最大静默段持续时间为 2 秒。

```
maxSilenceSegment = 2;
```

通过对数据存储循环调用 `read` 来构造训练信号。

```
numSamples = 1;
```

```
while numSamples < duration
    data = read(adsTrain);
    data = data ./ max(abs(data)); % Normalize amplitude
```

```
% Determine regions of speech
idx = detectSpeech(data,Fs,'Window',win,'Thresholds',T);
```

```
% If a region of speech is detected
if ~isempty(idx)
```

```
% Extend the indices by five frames
idx(1,1) = max(1,idx(1,1) - 5*numel(win));
idx(1,2) = min(length(data),idx(1,2) + 5*numel(win));
```

```
% Isolate the speech
data = data(idx(1,1):idx(1,2));
```

```
% Write speech segment to training signal
audioTraining(numSamples:numSamples+numel(data)-1) = data;
```

```
% Set VAD baseline
maskTraining(numSamples:numSamples+numel(data)-1) = true;
```

```
% Random silence period
numSilenceSamples = randi(maxSilenceSegment*Fs,1,1);
numSamples = numSamples + numel(data) + numSilenceSamples;
```

```
end
end
```

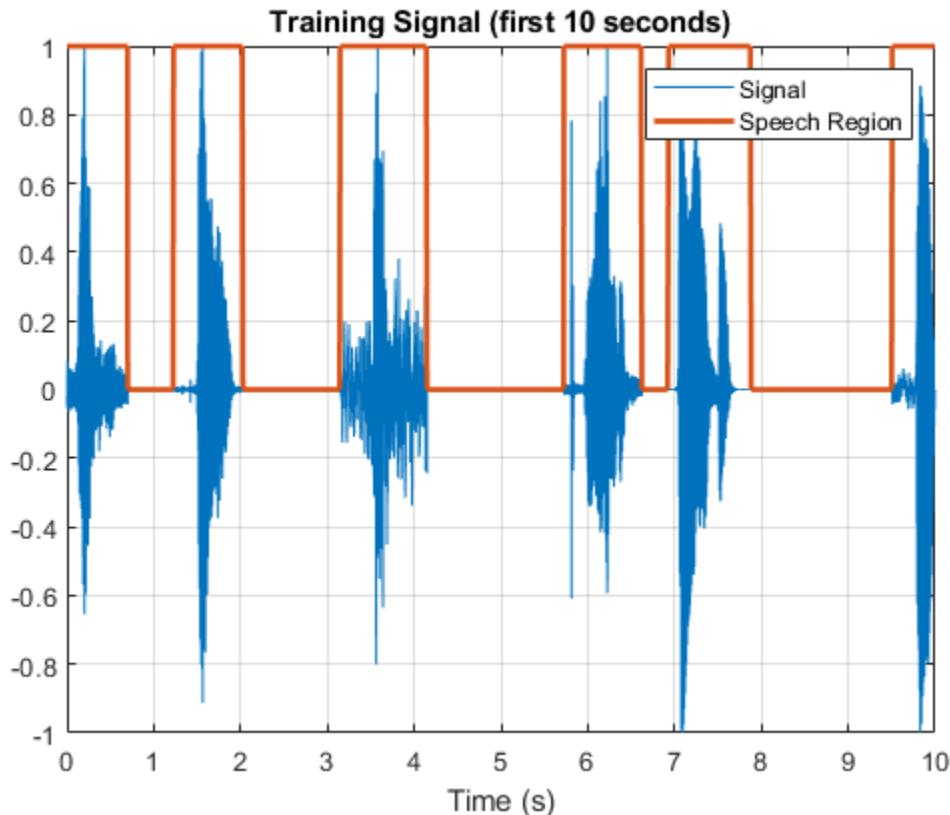
对训练信号的一个 10 秒部分进行可视化。绘制基线语音活动掩膜。

```
figure
range = 1:10*Fs;
```

```

plot((1/Fs)*(range-1),audioTraining(range));
hold on
plot((1/Fs)*(range-1),maskTraining(range));
grid on
lines = findall(gcf,"Type","Line");
lines(1).LineWidth = 2;
xlabel("Time (s)")
legend("Signal","Speech Region")
title("Training Signal (first 10 seconds)");

```



收听训练信号的前 10 秒。

```
sound(audioTraining(range),Fs);
```

### 向训练信号添加噪声

向语音信号添加洗衣机噪声以破坏训练信号，使其信噪比为 -10 dB。

读取 8 kHz 噪声，并将其转换为 16 kHz。

```

noise = audioread("WashingMachine-16-8-mono-1000secs.mp3");
noise = resample(noise,2,1);

```

用噪声损坏训练信号。

```

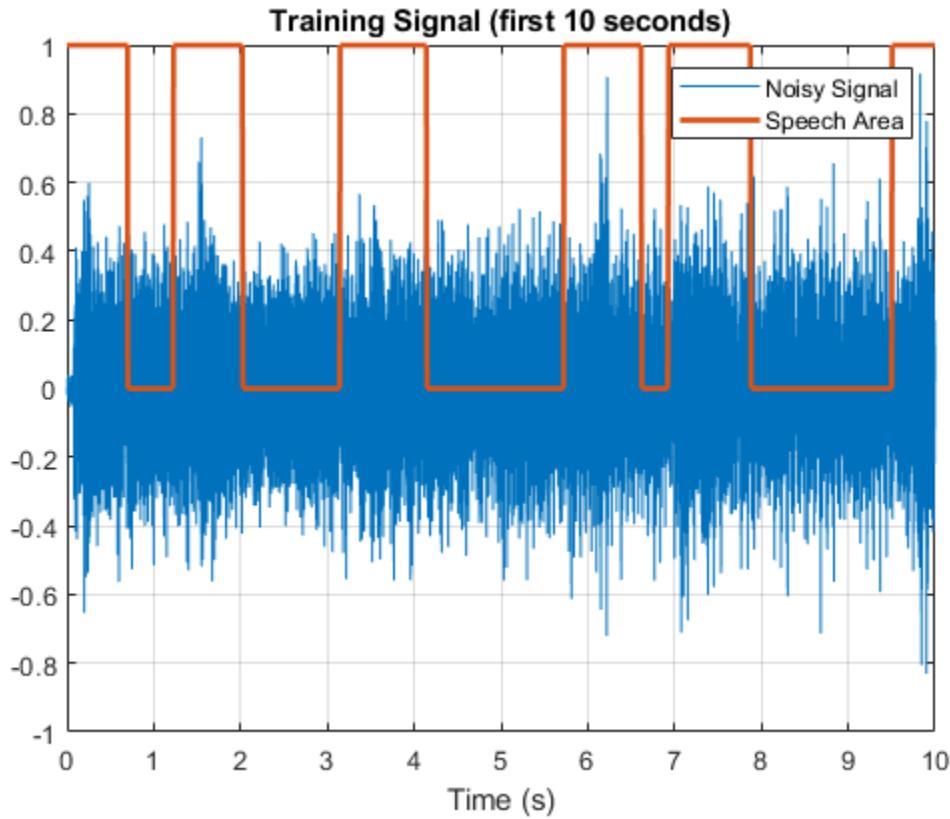
audioTraining = audioTraining(1:numel(noise));
SNR = -10;
noise = 10^(-SNR/20) * noise * norm(audioTraining) / norm(noise);

```

```
audioTrainingNoisy = audioTraining + noise;
audioTrainingNoisy = audioTrainingNoisy / max(abs(audioTrainingNoisy));
```

对含噪训练信号的一个 10 秒部分进行可视化。绘制基线语音活动掩膜。

```
figure
plot((1/Fs)*(range-1),audioTrainingNoisy(range));
hold on
plot((1/Fs)*(range-1),maskTraining(range));
grid on
lines = findall(gcf,"Type","Line");
lines(1).LineWidth = 2;
xlabel("Time (s)")
legend("Noisy Signal","Speech Area")
title("Training Signal (first 10 seconds)");
```



收听含噪训练信号的前 10 秒。

```
sound(audioTrainingNoisy(range),Fs)
```

请注意，您使用无噪声语音加静默信号获得了基线语音活动掩膜。验证对被噪声损坏的信号使用 **detectSpeech** 不会产生良好的结果。

```
speechIndices = detectSpeech(audioTrainingNoisy,Fs,'Window',win);
speechIndices(:,1) = max(1,speechIndices(:,1) - 5*numel(win));
speechIndices(:,2) = min(numel(audioTrainingNoisy),speechIndices(:,2) + 5*numel(win));
```

```

noisyMask = zeros(size(audioTrainingNoisy));
for ii = 1:size(speechIndices)
    noisyMask(speechIndices(ii,1):speechIndices(ii,2)) = 1;
end

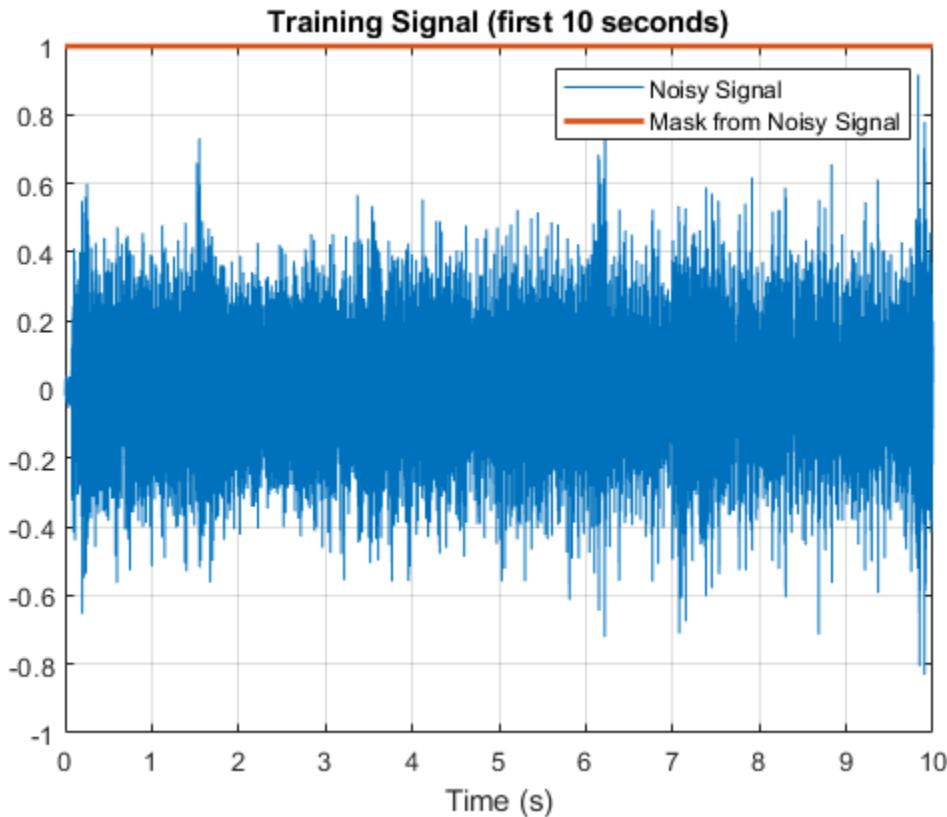
```

对含噪训练信号的一个 10 秒部分进行可视化。绘制通过解析含噪信号获得的语音活动掩膜。

```

figure
plot((1/Fs)*(range-1),audioTrainingNoisy(range));
hold on
plot((1/Fs)*(range-1),noisyMask(range));
grid on
lines = findall(gcf,"Type","Line");
lines(1).LineWidth = 2;
xlabel("Time (s)")
legend("Noisy Signal","Mask from Noisy Signal")
title("Training Signal (first 10 seconds)");

```



### 创建语音加静默验证信号

创建一个 200 秒的含噪语音信号来验证经过训练的网络。使用验证数据存储。请注意，验证数据存储和训练数据存储包含不同的说话者。

预分配验证信号和验证掩膜。您将使用此掩膜来评估经过训练的网络的准确度。

```

duration = 200*Fs;
audioValidation = zeros(duration,1);
maskValidation = zeros(duration,1);

```

通过对数据存储循环调用 `read` 来构造验证信号。

```

numSamples = 1;
while numSamples < duration
    data = read(adsValidation);
    data = data ./ max(abs(data)); % Normalize amplitude

    % Determine regions of speech
    idx = detectSpeech(data,Fs,'Window',win,'Thresholds',T);

    % If a region of speech is detected
    if ~isempty(idx)

        % Extend the indices by five frames
        idx(1,1) = max(1,idx(1,1) - 5*numel(win));
        idx(1,2) = min(length(data),idx(1,2) + 5*numel(win));

        % Isolate the speech
        data = data(idx(1,1):idx(1,2));

        % Write speech segment to training signal
        audioValidation(numSamples:numSamples+numel(data)-1) = data;

        % Set VAD Baseline
        maskValidation(numSamples:numSamples+numel(data)-1) = true;

        % Random silence period
        numSilenceSamples = randi(maxSilenceSegment*Fs,1,1);
        numSamples = numSamples + numel(data) + numSilenceSamples;
    end
end

```

通过向语音信号添加洗衣机噪声，用洗衣机噪声损坏验证信号，使信噪比为 -10 dB。对验证信号使用不同于训练信号所用的噪声文件。

```

noise = audioread("WashingMachine-16-8-mono-200secs.mp3");
noise = resample(noise,2,1);
noise = noise(1:duration);
audioValidation = audioValidation(1:numel(noise));

noise = 10^(-SNR/20) * noise * norm(audioValidation) / norm(noise);
audioValidationNoisy = audioValidation + noise;
audioValidationNoisy = audioValidationNoisy / max(abs(audioValidationNoisy));

```

### 提取训练特征

此示例使用以下特征训练 LSTM 网络：

- 1 `spectralCentroid` (Audio Toolbox)
- 2 `spectralCrest` (Audio Toolbox)
- 3 `spectralEntropy` (Audio Toolbox)
- 4 `spectralFlux` (Audio Toolbox)
- 5 `spectralKurtosis` (Audio Toolbox)
- 6 `spectralRolloffPoint` (Audio Toolbox)
- 7 `spectralSkewness` (Audio Toolbox)

- 8 **spectralSlope** (Audio Toolbox)
- 9 **harmonicRatio** (Audio Toolbox)

此示例使用 **audioFeatureExtractor** (Audio Toolbox) 为特征集创建一个最佳特征提取管道。创建一个 **audioFeatureExtractor** 对象来提取特征集。使用一个重叠率为 50% 的 256 点 Hann 窗。

```
afe = audioFeatureExtractor('SampleRate',Fs, ...
    'Window',hann(256,"Periodic"), ...
    'OverlapLength',128, ...
    ...
    'spectralCentroid',true, ...
    'spectralCrest',true, ...
    'spectralEntropy',true, ...
    'spectralFlux',true, ...
    'spectralKurtosis',true, ...
    'spectralRolloffPoint',true, ...
    'spectralSkewness',true, ...
    'spectralSlope',true, ...
    'harmonicRatio',true);
```

```
featuresTraining = extract(afe, audioTrainingNoisy);
```

显示特征矩阵的维度。第一个维度对应于信号所分成窗口的数量（它取决于窗口长度和重叠长度）。第二个维度是本示例中使用的特征的数量。

```
[numWindows,numFeatures] = size(featuresTraining)
numWindows = 125009
numFeatures = 9
```

在分类应用中，将所有特征归一化为具有零均值和单位标准差是很好的做法。

计算每个系数的均值和标准差，并使用它们来归一化数据。

```
M = mean(featuresTraining,1);
S = std(featuresTraining,[],1);
featuresTraining = (featuresTraining - M) ./ S;
```

使用相同的过程从验证信号中提取特征。

```
featuresValidation = extract(afe, audioValidationNoisy);
featuresValidation = (featuresValidation - mean(featuresValidation,1)) ./ std(featuresValidation,[],1);
```

每个特征对应于 128 个数据样本（跳跃长度）。对于每个跳跃，将预期的语音/无语音值设置为对应于这 128 个样本的基线掩膜值的模式。将语音/无语音掩膜转换为分类。

```
windowLength = numel(afe.Window);
hopLength = windowLength - afe.OverlapLength;
range = (hopLength) * (1:size(featuresTraining,1)) + hopLength;
maskMode = zeros(size(range));
for index = 1:numel(range)
    maskMode(index) = mode(maskTraining( (index-1)*hopLength+1:(index-1)*hopLength+windowLength ));
end
maskTraining = maskMode.';

maskTrainingCat = categorical(maskTraining);
```

对验证掩膜执行同样的操作。

```

range = (hopLength) * (1:size(featuresValidation,1)) + hopLength;
maskMode = zeros(size(range));
for index = 1:numel(range)
    maskMode(index) = mode(maskValidation( (index-1)*hopLength+1:(index-1)*hopLength+windowLength ));
end
maskValidation = maskMode.';

maskValidationCat = categorical(maskValidation);

将训练特征和掩膜分成长为 800 的序列，连续序列之间有 75% 的重叠。

sequenceLength = 800;
sequenceOverlap = round(0.75*sequenceLength);

trainFeatureCell = helperFeatureVector2Sequence(featuresTraining',sequenceLength,sequenceOverlap);
trainLabelCell = helperFeatureVector2Sequence(maskTrainingCat',sequenceLength,sequenceOverlap);

```

### 定义 LSTM 网络架构

LSTM 网络可以学习序列数据的时间步之间的长期相关性。此示例使用双向 LSTM 层 `bilstmLayer` 前向和后向分析序列。

将输入大小指定为长度为 9 的序列（特征数量）。指定输出大小为 200 的一个隐藏双向 LSTM 层，并输出序列。此命令指示双向 LSTM 层将输入时序映射为 200 个特征，这些特征会传递到下一个层。然后，指定输出大小为 200 的一个双向 LSTM 层，并输出序列的最后一个元素。此命令指示双向 LSTM 层将其输入映射到 200 个特征，然后准备好输出到全连接层。最后，通过包含大小为 2 的全连接层，后跟 softmax 层和分类层，来指定两个类。

```

layers = [ ...
    sequenceInputLayer( size(featuresValidation,2) )
    bilstmLayer(200,"OutputMode","sequence")
    bilstmLayer(200,"OutputMode","sequence")
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer
];

```

接下来，指定分类器的训练选项。将 `MaxEpochs` 设置为 20 以便基于训练数据对网络进行 20 轮训练。将 `MiniBatchSize` 设置为 64 以便网络可以一次查看 64 个训练信号。将 `Plots` 设置为 "training-progress" 以生成显示训练进度随着迭代次数增加而变化的图。将 `Verbose` 设置为 `false` 以禁止打印对应于图中所示数据的表输出。将 `Shuffle` 设置为 "every-epoch" 以便在每轮开始时使训练序列变为乱序。将 `LearnRateSchedule` 设置为 "piecewise" 以便每经过一定的轮数 (10) 时，按指定的因子 (0.1) 降低学习率。将 `ValidationData` 设置为验证预测变量和目标。

此示例使用自适应矩估计 (ADAM) 求解器。与默认的具有动量的随机梯度下降 (SGDM) 求解器相比，ADAM 在使用 LSTM 之类的循环神经网络 (RNN) 时性能更好。

```

maxEpochs = 20;
miniBatchSize = 64;
options = trainingOptions("adam", ...
    "MaxEpochs",maxEpochs, ...
    "MiniBatchSize",miniBatchSize, ...
    "Shuffle","every-epoch", ...
    "Verbose",0, ...
    "SequenceLength",sequenceLength, ...
    "ValidationFrequency",floor(numel(trainFeatureCell)/miniBatchSize), ...
    "ValidationData",{featuresValidation.',maskValidationCat.'}, ...

```

```
"Plots","training-progress", ...
"LearnRateSchedule","piecewise", ...
"LearnRateDropFactor",0.1, ...
"LearnRateDropPeriod",5);
```

## 训练 LSTM 网络

使用 `trainNetwork` 用指定的训练选项和层架构训练 LSTM 网络。由于训练集很大，训练过程可能需要几分钟。

```
doTraining = true;
if doTraining
    [speechDetectNet,netInfo] = trainNetwork(trainFeatureCell,trainLabelCell,layers,options);
    fprintf("Validation accuracy: %f percent.\n", netInfo.FinalValidationAccuracy);
else
    load speechDetectNet
end
```

Validation accuracy: 91.320312 percent.

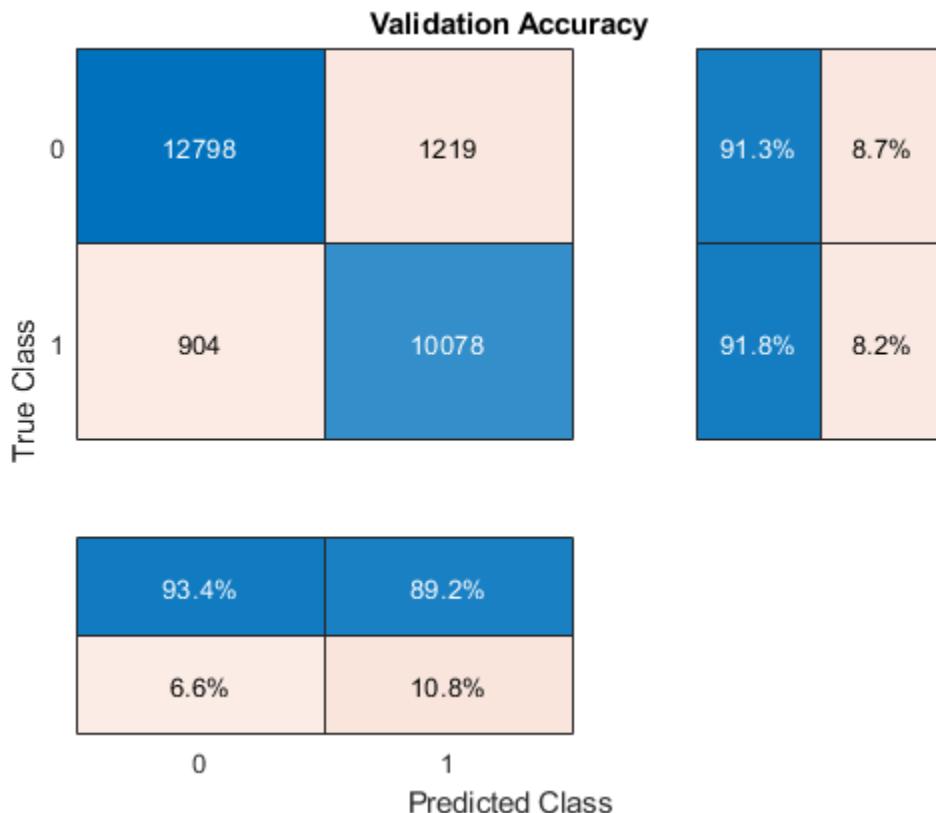
## 使用经过训练的网络检测语音活动

使用经过训练的网络估计验证信号中的语音活动。将估计的 VAD 掩膜从分类转换为双精度。

```
EstimatedVADMask = classify(speechDetectNet,featuresValidation.');
EstimatedVADMask = double(EstimatedVADMask);
EstimatedVADMask = EstimatedVADMask.' - 1;
```

根据由实际和估计标签组成的向量来计算并绘制验证混淆矩阵。

```
figure
cm = confusionchart(maskValidation,EstimatedVADMask,"title","Validation Accuracy");
cm.ColumnSummary = "column-normalized";
cm.RowSummary = "row-normalized";
```



如果您更改网络或特征提取管道的参数，请考虑使用新网络和 `audioFeatureExtractor` 对象重新保存 MAT 文件。

```
resaveNetwork = ;  
if resaveNetwork  
    save('Audio_VoiceActivityDetectionExample.mat','speechDetectNet','afe');  
end
```

### 支持函数

#### 将特征向量转换为序列

```
function [sequences,sequencePerFile] = helperFeatureVector2Sequence(features,featureVectorsPerSequence,fe  
    if featureVectorsPerSequence <= featureVectorOverlap  
        error("The number of overlapping feature vectors must be less than the number of feature vectors per sequence");  
    end  
  
    if ~iscell(features)  
        features = {features};  
    end  
    hopLength = featureVectorsPerSequence - featureVectorOverlap;  
    idx1 = 1;  
    sequences = {};  
    sequencePerFile = cell(numel(features),1);  
    for ii = 1:numel(features)  
        sequencePerFile{ii} = floor((size(features{ii},2) - featureVectorsPerSequence)/hopLength) + 1;  
        idx2 = 1;
```

```

for j = 1:sequencePerFile{ii}
    sequences{idx1,1} = features{ii}(:,idx2:idx2 + featureVectorsPerSequence - 1); %#ok<AGROW>
    idx1 = idx1 + 1;
    idx2 = idx2 + hopLength;
end
end
end

```

## 流式演示

```
function helperStreamingDemo(speechDetectNet,afe,cleanSpeech,noise,testDuration,sequenceLength,sequence
```

创建 `dsp.AudioFileReader` (DSP System Toolbox) 对象，逐帧从语音和噪声文件中读取。

```

speechReader = dsp.AudioFileReader(cleanSpeech,'PlayCount',inf);
noiseReader = dsp.AudioFileReader(noise,'PlayCount',inf);
fs = speechReader.SampleRate;

```

创建一个 `dsp.MovingStandardDeviation` (DSP System Toolbox) 对象和一个 `dsp.MovingAverage` (DSP System Toolbox) 对象。您将使用这些对象来确定音频特征的标准差和均值以进行归一化。统计量应随着时间的推移逐步改善。

```

movSTD = dsp.MovingStandardDeviation('Method','Exponential weighting','ForgettingFactor',1);
movMean = dsp.MovingAverage('Method','Exponential weighting','ForgettingFactor',1);

```

创建三个 `dsp.AsyncBuffer` (DSP System Toolbox) 对象。一个用于缓冲输入音频，一个用于缓冲提取的特征，一个用于缓冲输出缓冲区。只有实时可视化决策才需要输出缓冲区。

```

audioInBuffer = dsp.AsyncBuffer;
featureBuffer = dsp.AsyncBuffer;
audioOutBuffer = dsp.AsyncBuffer;

```

对于音频缓冲区，您将缓冲原始的干净语音信号和含噪信号。您将只播放指定的 `signalToListenTo`。将 `signalToListenTo` 变量转换为您要收听的通道。

```

channelToListenTo = 1;
if strcmp(signalToListenTo,"clean")
    channelToListenTo = 2;
end

```

创建一个 Time Scope 来可视化原始语音信号、网络所应用于的含噪信号以及网络的决策输出。

```

scope = timescope('SampleRate',fs, ...
    'TimeSpanSource','property', ...
    'TimeSpan',3, ...
    'BufferLength',fs*3*3, ...
    'YLimits',[-1 1], ...
    'TimeSpanOverrunAction','Scroll', ...
    'ShowGrid',true, ...
    'NumInputPorts',3, ...
    'LayoutDimensions',[3,1], ...
    'Title','Noisy Speech');
scope.ActiveDisplay = 2;
scope.Title = 'Clean Speech (Original)';
scope.YLimits = [-1 1];
scope.ActiveDisplay = 3;
scope.Title = 'Detected Speech';
scope.YLimits = [-1 1];

```

创建一个 `audioDeviceWriter` (Audio Toolbox) 对象来从扬声器播放原始音频或含噪音频。

```
deviceWriter = audioDeviceWriter('SampleRate',fs);
```

初始化循环中所使用的变量。

```
windowLength = numel(afe.Window);
hopLength = windowLength - afe.OverlapLength;
myMax = 0;
audioBufferInitialized = false;
featureBufferInitialized = false;
```

运行流式演示。

```
tic
while toc < testDuration

    % Read a frame of the speech signal and a frame of the noise signal
    speechIn = speechReader();
    noiseIn = noiseReader();

    % Mix the speech and noise at the specified SNR
    noisyAudio = speechIn + noiseGain*noiseIn;

    % Update a running max for normalization
    myMax = max(myMax,max(abs(noisyAudio)));

    % Write the noisy audio and speech to buffers
    write(audioInBuffer,[noisyAudio,speechIn]);

    % If enough samples are buffered,
    % mark the audio buffer as initialized and push the read pointer
    % for the audio buffer up a window length.
    if audioInBuffer.NumUnreadSamples >= windowLength && ~audioBufferInitialized
        audioBufferInitialized = true;
        read(audioInBuffer,windowLength);
    end

    % If enough samples are in the audio buffer to calculate a feature
    % vector, read the samples, normalize them, extract the feature vectors, and write
    % the latest feature vector to the features buffer.
    while (audioInBuffer.NumUnreadSamples >= hopLength) && audioBufferInitialized
        x = read(audioInBuffer,windowLength + hopLength,windowLength);
        write(audioOutBuffer,x(end-hopLength+1:end,:));
        noisyAudio = x(:,1);
        noisyAudio = noisyAudio/myMax;
        features = extract(afe,noisyAudio);
        write(featureBuffer,features(2,:));
    end

    % If enough feature vectors are buffered, mark the feature buffer
    % as initialized and push the read pointer for the feature buffer
    % and the audio output buffer (so that they are in sync).
    if featureBuffer.NumUnreadSamples >= (sequenceLength + sequenceHop) && ~featureBufferInitialized
        featureBufferInitialized = true;
        read(featureBuffer,sequenceLength - sequenceHop);
        read(audioOutBuffer,(sequenceLength - sequenceHop)*windowLength);
    end
```

```

while featureBuffer.NumUnreadSamples >= sequenceHop && featureBufferInitialized
    features = read(featureBuffer,sequenceLength,sequenceLength - sequenceHop);
    features(isnan(features)) = 0;

    % Use only the new features to update the
    % standard deviation and mean. Normalize the features.
    localSTD = movSTD(features(end-sequenceHop+1:end,:));
    localMean = movMean(features(end-sequenceHop+1:end,:));
    features = (features - localMean(end,:)) ./ localSTD(end,:);

    decision = classify(speechDetectNet,features');
    decision = decision(end-sequenceHop+1:end);
    decision = double(decision)' - 1;
    decision = repelem(decision,hopLength);

    audioHop = read(audioOutBuffer,sequenceHop*hopLength);

    % Listen to the speech or speech+noise
    deviceWriter(audioHop(:,channelToListenTo));

    % Visualize the speech+noise, the original speech, and the
    % voice activity detection.
    scope(audioHop(:,1),audioHop(:,2),audioHop(:,1).*decision)
end
end
release(deviceWriter)
release(audioInBuffer)
release(audioOutBuffer)
release(featureBuffer)
release(movSTD)
release(movMean)
release(scope)
end

```

## 参考资料

[1] Warden P."Speech Commands:A public dataset for single-word speech recognition", 2017.可从 [https://storage.googleapis.com/download.tensorflow.org/data/speech\\_commands\\_v0.01.tar.gz](https://storage.googleapis.com/download.tensorflow.org/data/speech_commands_v0.01.tar.gz) 获得。Copyright Google 2017.Speech Commands Dataset 是根据 Creative Commons Attribution 4.0 许可证授权的

## 另请参阅

[trainingOptions](#) | [trainNetwork](#)

## 详细信息

- “在 MATLAB 中进行深度学习”（第 1-2 页）

## 使用深度学习网络对语音去噪

此示例说明如何使用深度学习网络对语音信号去噪。该示例比较应用于同一任务的两种类型的网络：全连接网络和卷积网络。

### 简介

语音去噪的目的是去除语音信号中的噪声，同时提高语音的质量和清晰度。此示例说明如何使用深度学习网络从语音信号中去除洗衣机噪声。该示例比较应用于同一任务的两种类型的网络：全连接网络和卷积网络。

### 问题摘要

假设有以下以 8 kHz 采样的语音信号。

```
[cleanAudio,fs] = audioread("SpeechDFT-16-8-mono-5secs.wav");
sound(cleanAudio,fs)

给语音信号增加洗衣机噪声。设置噪声功率，使信噪比 (SNR) 为零 dB。
noise = audioread("WashingMachine-16-8-mono-1000secs.mp3");

% Extract a noise segment from a random location in the noise file
ind = randi(numel(noise) - numel(cleanAudio) + 1, 1, 1);
noiseSegment = noise(ind:ind + numel(cleanAudio) - 1);

speechPower = sum(cleanAudio.^2);
noisePower = sum(noiseSegment.^2);
noisyAudio = cleanAudio + sqrt(speechPower/noisePower) * noiseSegment;
```

收听含噪的语音信号。

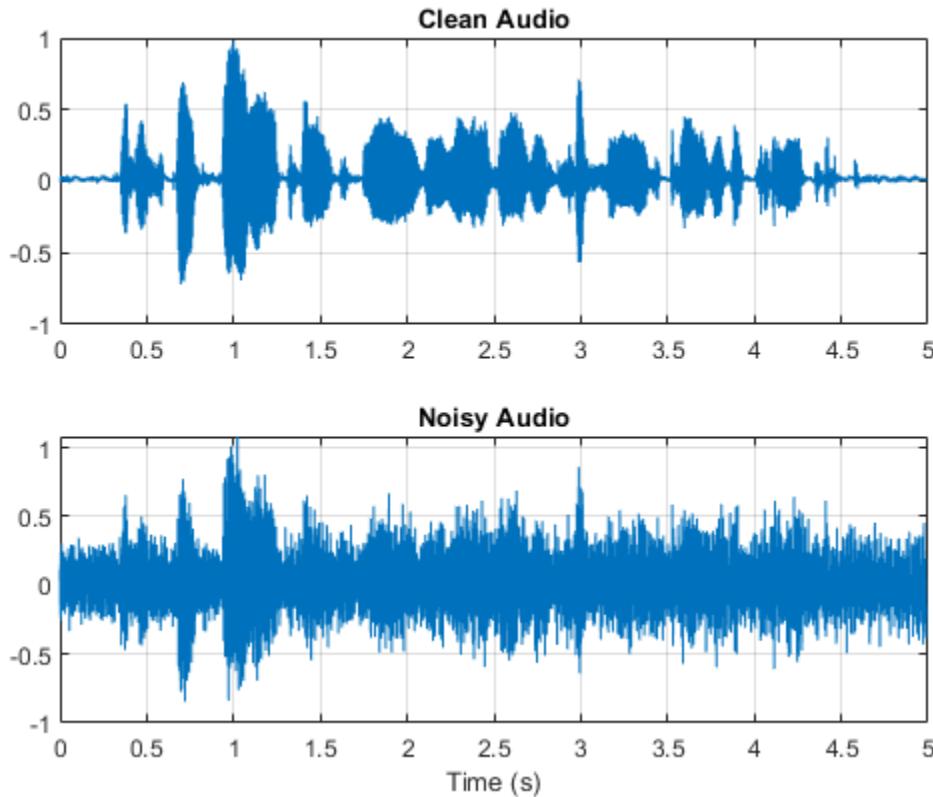
```
sound(noisyAudio,fs)
```

可视化原始信号和含噪信号。

```
t = (1/fs) * (0:numel(cleanAudio)-1);
```

```
subplot(2,1,1)
plot(t,cleanAudio)
title("Clean Audio")
grid on

subplot(2,1,2)
plot(t,noisyAudio)
title("Noisy Audio")
xlabel("Time (s)")
grid on
```



语音去噪的目标是从语音信号中去除洗衣机噪声，同时最小化输出语音中不需要的内容。

### 检查数据集

此示例使用 Mozilla 通用语音数据集 [1 (第 14-0 页) ] 的一部分来训练和测试深度学习网络。该数据集包含受试者口述短句的 48 kHz 录音。下载该数据集并解压缩下载的文件。

```
url = 'http://ssd.mathworks.com/supportfiles/audio/commonvoice.zip';
downloadFolder = tempdir;
dataFolder = fullfile(downloadFolder,'commonvoice');

if ~exist(dataFolder,'dir')
    disp('Downloading data set (956 MB) ...')
    unzip(url,downloadFolder)
end
```

使用 `audioDatastore` 为训练集创建数据存储。要以牺牲性能为代价来加快示例的运行时间，请将 `reduceDataset` 设置为 `true`。

```
adsTrain = audioDatastore(fullfile(dataFolder,'train'),'IncludeSubfolders',true);
```

```
reduceDataset = true;
if reduceDataset
    adsTrain = shuffle(adsTrain);
    adsTrain = subset(adsTrain,1:1000);
end
```

使用 `read` 获取数据存储中第一个文件的内容。

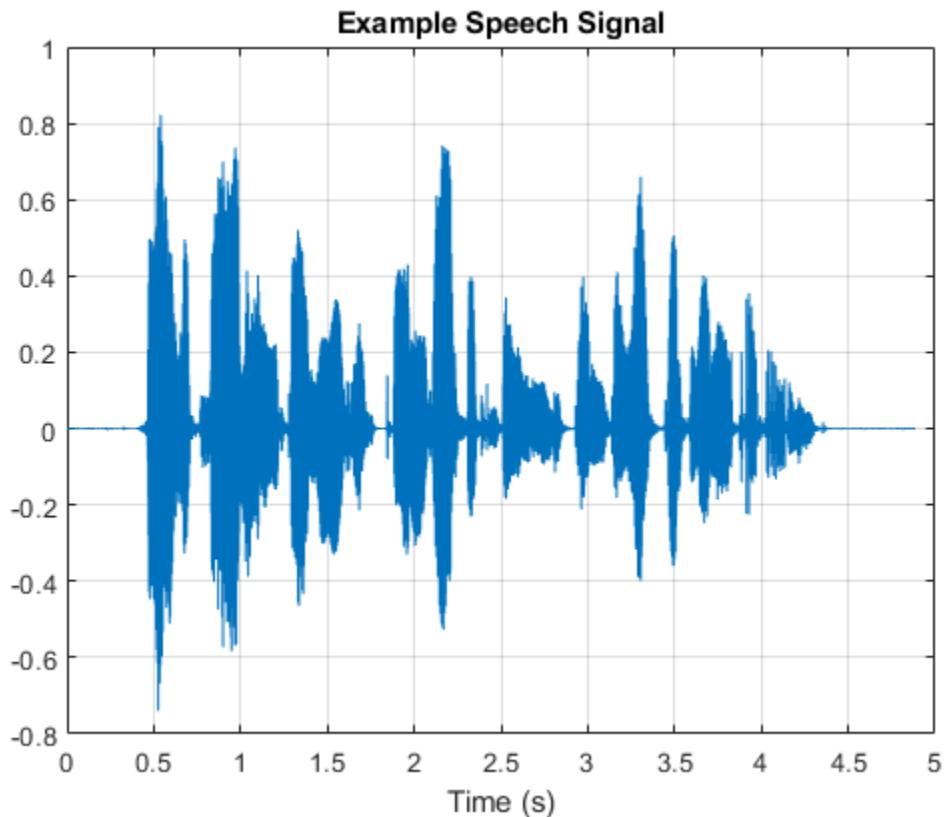
```
[audio,adsTrainInfo] = read(adsTrain);
```

收听语音信号。

```
sound(audio,adsTrainInfo.SampleRate)
```

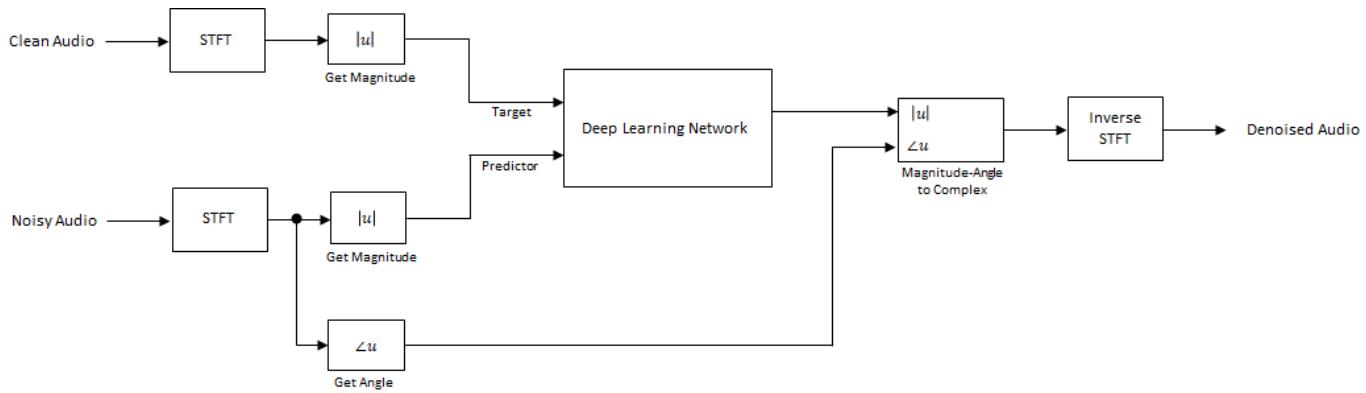
对语音信号绘图。

```
figure
t = (1/adsTrainInfo.SampleRate) * (0:numel(audio)-1);
plot(t, audio)
title("Example Speech Signal")
xlabel("Time (s)")
grid on
```



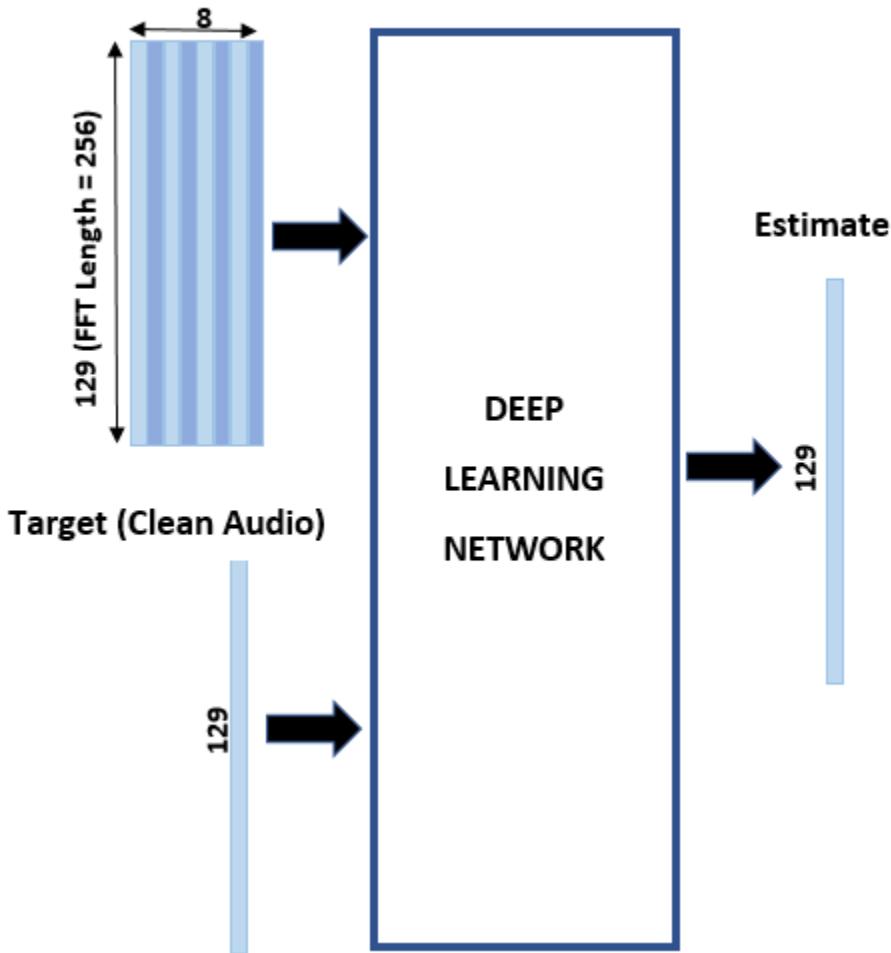
### 深度学习系统概述

基本的深度学习训练方案如下所示。请注意，因为语音通常低于 4 kHz，您首先将干净和含噪音音频信号下采样到 8 kHz，以减轻网络的计算负荷。预测变量信号和网络目标信号分别是含噪音音频信号和干净音频信号的幅值频谱。网络的输出是去噪信号的幅值频谱。回归网络使用预测变量输入来最小化其输出和输入目标之间的均方误差。使用输出的幅值频谱和含噪信号的相位将去噪后的音频转换回时域 [2 (第 14-0 页)]。



您可以使用短时傅里叶变换 (STFT) 将音频变换为频域，使用的窗口长度为 256 个样本、重叠率为 75% 并使用 Hamming 窗。通过丢弃对应于负频率的频率样本，可以将频谱向量的大小减小到 129（因为时域语音信号是实信号，这样不会导致任何信息丢失）。预测变量输入由 8 个连续含噪 STFT 向量组成，因此每个 STFT 输出估计是基于当前含噪 STFT 和 7 个前面的含噪 STFT 向量计算的。

### Predictor (Noisy Audio)



## STFT 目标和预测变量

本节说明如何从一个训练文件中生成目标和预测变量信号。

首先，定义系统参数：

```
windowLength = 256;
win = hamming(windowLength,"periodic");
overlap = round(0.75 * windowLength);
ffTLength = windowLength;
inputFs = 48e3;
fs = 8e3;
numFeatures = ffTLength/2 + 1;
numSegments = 8;
```

创建一个 `dsp.SampleRateConverter` (DSP System Toolbox) 对象以将 48 kHz 音频转换为 8 kHz。

```
src = dsp.SampleRateConverter("InputSampleRate",inputFs, ...
    "OutputSampleRate",fs, ...
    "Bandwidth",7920);
```

使用 `read` 从数据存储中获取音频文件的内容。

```
audio = read(adsTrain);
```

确保音频长度是采样率转换器抽取因子的倍数。

```
decimationFactor = inputFs/fs;
L = floor(numel(audio)/decimationFactor);
audio = audio(1:decimationFactor*L);
```

将音频信号转换为 8 kHz。

```
audio = src(audio);
reset(src)
```

使用洗衣机噪声向量创建一个随机噪声段。

```
randind = randi(numel(noise) - numel(audio),[1 1]);
noiseSegment = noise(randind : randind + numel(audio) - 1);
```

向语音信号添加噪声，使 SNR 为 0 dB。

```
noisePower = sum(noiseSegment.^2);
cleanPower = sum(audio.^2);
noiseSegment = noiseSegment .* sqrt(cleanPower/noisePower);
noisyAudio = audio + noiseSegment;
```

使用 `stft` (Signal Processing Toolbox) 基于原始和含噪音音频信号生成幅值 STFT 向量。

```
cleanSTFT = stft(audio,'Window',win,'OverlapLength',overlap,'FFTLength',ffTLength);
cleanSTFT = abs(cleanSTFT(numFeatures-1:end,:));
noisySTFT = stft(noisyAudio,'Window',win,'OverlapLength',overlap,'FFTLength',ffTLength);
noisySTFT = abs(noisySTFT(numFeatures-1:end,:));
```

基于含噪 STFT 生成包含 8 个段的训练预测变量信号。连续预测变量之间的重叠是 7 个段。

```
noisySTFT = [noisySTFT(:,1:numSegments - 1), noisySTFT];
stftSegments = zeros(numFeatures, numSegments , size(noisySTFT,2) - numSegments + 1);
```

```
for index = 1:size(noisySTFT,2) - numSegments + 1
    stftSegments(:, :, index) = (noisySTFT(:, index:index + numSegments - 1));
end
```

设置目标和预测变量。两个变量的最后一个维度对应于由音频文件生成的非重复预测变量/目标对组的数量。每个预测变量为  $129 \times 8$ , 每个目标为  $129 \times 1$ 。

```
targets = cleanSTFT;
size(targets)
```

```
ans = 1×2
```

```
129 544
```

```
predictors = stftSegments;
size(predictors)
```

```
ans = 1×3
```

```
129 8 544
```

## 使用 tall 数组提取特征

为了加快处理速度, 使用 tall 数组从数据存储中所有音频文件的语音段中提取特征序列。与内存数组不同, 在您调用 **gather** 函数之前, tall 数组通常不会实际进行计算。这种延迟计算使您能够快速处理大型数据集。当使用 **gather** 最终请求输出时, MATLAB 会尽可能地合并排队的计算, 并执行最少次数的数据遍历。如果您有 Parallel Computing Toolbox™, 您可以在本地 MATLAB 会话中或在本地并行池中使用 tall 数组。如果安装了 MATLAB® Parallel Server™, 您还可以在群集上运行 tall 数组计算。

首先, 将数据存储转换为 tall 数组。

```
reset(adsTrain)
T = tall(adsTrain)
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).
```

```
T =
```

```
M×1 tall cell array
```

```
{234480×1 double}
{210288×1 double}
{282864×1 double}
{292080×1 double}
{410736×1 double}
{303600×1 double}
{326640×1 double}
{233328×1 double}
:
:
:
```

上面的输出内容指示行数 (对应于数据存储中的文件数) M 未知。M 是占位符, 直到计算完成才会填充该值。

从 tall 表中提取目标幅值 STFT 和预测变量幅值 STFT。此操作会创建新 tall 数组变量以用于后续计算。函数 **HelperGenerateSpeechDenoisingFeatures** 执行在 STFT 目标和预测变量 (第 14-0 页) 部分

中已着重介绍的步骤。`cellfun` 命令将 `HelperGenerateSpeechDenoisingFeatures` 应用于数据存储中每个音频文件的内容。

```
[targets,predictors] = cellfun(@(x)HelperGenerateSpeechDenoisingFeatures(x,noise,src),T,"UniformOutput",false)
```

使用 `gather` 计算目标和预测变量。

```
[targets,predictors] = gather(targets,predictors);
```

Evaluating tall expression using the Parallel Pool 'local':

```
- Pass 1 of 1: Completed in 42 sec
Evaluation completed in 1 min 36 sec
```

将所有特征归一化为具有零均值和单位标准差是很好的做法。

分别计算预测变量和目标的均值和标准差，并使用它们来归一化数据。

```
predictors = cat(3,predictors{:});
noisyMean = mean(predictors(:));
noisyStd = std(predictors(:));
predictors(:) = (predictors(:) - noisyMean)/noisyStd;

targets = cat(2,targets{:});
cleanMean = mean(targets(:));
cleanStd = std(targets(:));
targets(:) = (targets(:) - cleanMean)/cleanStd;
```

将预测变量和目标重构为深度学习网络需要的维度。

```
predictors = reshape(predictors,size(predictors,1),size(predictors,2),1,size(predictors,3));
targets = reshape(targets,1,1,size(targets,1),size(targets,2));
```

在训练期间，您将使用 1% 的数据进行验证。验证对于检测网络过拟合训练数据的情况非常有用。

将数据随机分成训练集和验证集。

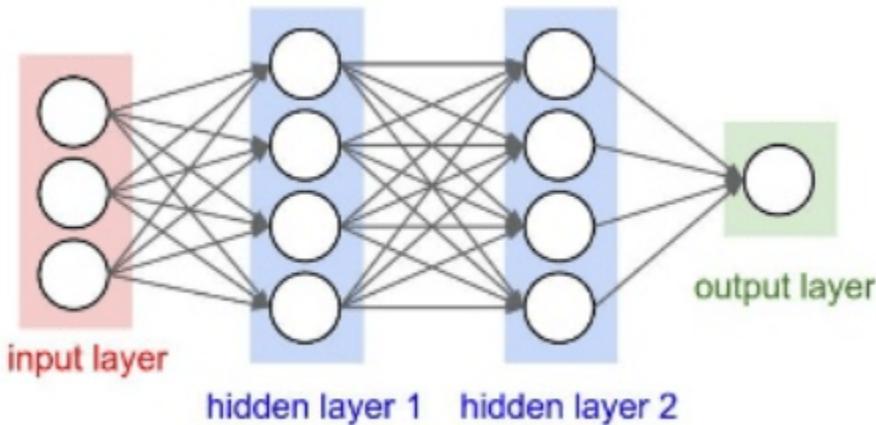
```
inds = randperm(size(predictors,4));
L = round(0.99 * size(predictors,4));

trainPredictors = predictors(:,:,:,:inds(1:L));
trainTargets = targets(:,:,:,:inds(1:L));

validatePredictors = predictors(:,:,:,:inds(L+1:end));
validateTargets = targets(:,:,:,:inds(L+1:end));
```

### 使用全连接层进行语音去噪

首先以由全连接层组成的网络的去噪为例。全连接层中的每个神经元都连接到前一个层的所有激活。全连接层将输入乘以权重矩阵，然后添加偏置向量。权重矩阵和偏置向量的维度由层中神经元的数量和前一个层中激活的数量决定。



定义网络的各层。将输入大小指定为大小为 NumFeatures×NumSegments（本示例中为  $129 \times 8$ ）的图像。定义两个隐藏的全连接层，每个层有 1024 个神经元。由于是纯线性系统，每个隐藏的全连接层后面都有一个修正线性单元（ReLU）层。批量归一化层对输出的均值和标准差进行归一化。添加一个包含 129 个神经元的全连接层，后跟一个回归层。

```
layers = [
    imageInputLayer([numFeatures,numSegments])
    fullyConnectedLayer(1024)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(1024)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(numFeatures)
    regressionLayer
];
```

接下来，指定网络的训练选项。将 **MaxEpochs** 设置为 3 以便基于训练数据对网络进行 3 轮训练。将 **MiniBatchSize** 设置为 128 以便网络可以一次查看 128 个训练信号。将 **Plots** 指定为 "training-progress" 以生成随着迭代次数增加显示训练进度的图。将 **Verbose** 设置为 `false` 以禁止将对应于图中所示数据的表输出打印到命令行窗口中。将 **Shuffle** 指定为 "every-epoch" 以便在每轮开始时打乱训练序列。将 **LearnRateSchedule** 指定为 "piecewise" 以便每经过一定数量的轮次 (1) 时，按指定的因子 (0.9) 降低学习率。将 **ValidationData** 设置为验证预测变量和目标。设置 **ValidationFrequency** 以便每轮计算一次验证均方误差。此示例使用自适应矩估计 (Adam) 求解器。

```
miniBatchSize = 128;
options = trainingOptions("adam", ...
    "MaxEpochs",3, ...
    "InitialLearnRate",1e-5, ...
    "MiniBatchSize",miniBatchSize, ...
    "Shuffle","every-epoch", ...
    "Plots","training-progress", ...
    "Verbose",false, ...
    "ValidationFrequency",floor(size(trainPredictors,4)/miniBatchSize), ...
    "LearnRateSchedule","piecewise", ...
    "LearnRateDropFactor",0.9, ...
    "LearnRateDropPeriod",1, ...
    "ValidationData",{validatePredictors,validateTargets});
```

使用 `trainNetwork` 用指定的训练选项和层架构训练网络。由于训练集很大，训练过程可能需要几分钟。要下载并加载预训练网络而不是从头开始训练网络，请将 `doTraining` 设置为 `false`。

```
doTraining = true;
if doTraining
    denoiseNetFullyConnected = trainNetwork(trainPredictors,trainTargets,layers,options);
else
    url = 'http://ssd.mathworks.com/supportfiles/audio/SpeechDenoising.zip';
    downloadNetFolder = tempdir;
    netFolder = fullfile(downloadNetFolder,'SpeechDenoising');
    if ~exist(netFolder,'dir')
        disp('Downloading pretrained network (1 file - 8 MB ...)')
        unzip(url,downloadNetFolder)
    end
    s = load(fullfile(netFolder,"denoisenet.mat"));
    denoiseNetFullyConnected = s.denoiseNetFullyConnected;
    cleanMean = s.cleanMean;
    cleanStd = s.cleanStd;
    noisyMean = s.noisyMean;
    noisyStd = s.noisyStd;
end
```

计算网络中全连接层的权重数。

```
numWeights = 0;
for index = 1:numel(denoiseNetFullyConnected.Layers)
    if isa(denoiseNetFullyConnected.Layers(index),"nnet.cnn.layer.FullyConnectedLayer")
        numWeights = numWeights + numel(denoiseNetFullyConnected.Layers(index).Weights);
    end
end
fprintf("The number of weights is %d.\n",numWeights);
```

The number of weights is 2237440.

### 使用卷积层进行语音去噪

以使用卷积层而不是全连接层的网络为例 [3 (第 14-0 页)]。二维卷积层对输入应用滑动滤波器。该层通过沿输入的纵向和横向移动滤波器来对输入进行卷积，并计算权重和输入的点积，然后加上偏置项。卷积层通常具有由比全连接层更少的参数。

定义由 16 个卷积层组成的 [3 (第 14-0 页)] 中所述的全卷积网络的层。前 15 个卷积层是重复 5 次的含 3 层的组，滤波器宽度分别为 9、5 和 9，滤波器数量分别为 18、30 和 8。最后一个卷积层具有 129 的滤波器宽度和 1 个滤波器。在此网络中，卷积仅在一个方向（沿频率维度）上执行，并且对于除第一层之外的所有层，沿时间维度的滤波器宽度设置为 1。与全连接网络相似，卷积层后是 ReLu 和批量归一化层。

```
layers = [imageInputLayer([numFeatures,numSegments])
    convolution2dLayer([9 8],18,"Stride",[1 100],"Padding","same")
    batchNormalizationLayer
    reluLayer

    repmat( ...
    [convolution2dLayer([5 1],30,"Stride",[1 100],"Padding","same")]
    batchNormalizationLayer
    reluLayer
    convolution2dLayer([9 1],8,"Stride",[1 100],"Padding","same")]
    batchNormalizationLayer
```

```

reluLayer
convolution2dLayer([9 1],18,"Stride",[1 100],"Padding","same")
batchNormalizationLayer
reluLayer],4,1)

convolution2dLayer([5 1],30,"Stride",[1 100],"Padding","same")
batchNormalizationLayer
reluLayer
convolution2dLayer([9 1],8,"Stride",[1 100],"Padding","same")
batchNormalizationLayer
reluLayer

convolution2dLayer([129 1],1,"Stride",[1 100],"Padding","same")

regressionLayer
];

```

除验证目标信号的大小置换为与回归层预期的大小一致之外，训练选项与完全连接网络的选项相同。

```

options = trainingOptions("adam", ...
    "MaxEpochs",3, ...
    "InitialLearnRate",1e-5, ...
    "MiniBatchSize",miniBatchSize, ...
    "Shuffle","every-epoch", ...
    "Plots","training-progress", ...
    "Verbose",false, ...
    "ValidationFrequency",floor(size(trainPredictors,4)/miniBatchSize), ...
    "LearnRateSchedule","piecewise", ...
    "LearnRateDropFactor",0.9, ...
    "LearnRateDropPeriod",1, ...
    "ValidationData",validatePredictors,permute(validateTargets,[3 1 2 4]));

```

使用 `trainNetwork` 用指定的训练选项和层架构训练网络。由于训练集很大，训练过程可能需要几分钟。要下载并加载预训练网络而不是从头开始训练网络，请将 `doTraining` 设置为 `false`。

```

doTraining = true;
if doTraining
    denoiseNetFullyConvolutional = trainNetwork(trainPredictors,permute(trainTargets,[3 1 2 4]),layers,options)
else
    url = 'http://ssd.mathworks.com/supportfiles/audio/SpeechDenoising.zip';
    downloadNetFolder = tempdir;
    netFolder = fullfile(downloadNetFolder,'SpeechDenoising');
    if ~exist(netFolder,'dir')
        disp('Downloading pretrained network (1 file - 8 MB) ...')
        unzip(url,downloadNetFolder)
    end
    s = load(fullfile(netFolder,"denoisenet.mat"));
    denoiseNetFullyConvolutional = s.denoiseNetFullyConvolutional;
    cleanMean = s.cleanMean;
    cleanStd = s.cleanStd;
    noisyMean = s.noisyMean;
    noisyStd = s.noisyStd;
end

```

计算网络中全连接层的权重数。

```

numWeights = 0;
for index = 1:numel(denoiseNetFullyConvolutional.Layers)

```

```

if isa(denoiseNetFullyConvolutional.Layers(index),"nnet.cnn.layer.Convolution2DLayer")
    numWeights = numWeights + numel(denoiseNetFullyConvolutional.Layers(index).Weights);
end
end
fprintf("The number of weights in convolutional layers is %d\n",numWeights);

```

The number of weights in convolutional layers is 31812

### 测试去噪网络

读入测试数据集。

```
adsTest = audioDatastore(fullfile(dataFolder,'test'),'IncludeSubfolders',true);
```

从数据存储中读取一个文件的内容。

```
[cleanAudio,adsTestInfo] = read(adsTest);
```

确保音频长度是采样率转换器抽取因子的倍数。

```
L = floor(numel(cleanAudio)/decimationFactor);
cleanAudio = cleanAudio(1:decimationFactor*L);
```

将音频信号转换为 8 kHz。

```
cleanAudio = src(cleanAudio);
reset(src)
```

在此测试阶段，您用在训练阶段没有使用的洗衣机噪声干扰语音。

```
noise = audioread("WashingMachine-16-8-mono-200secs.mp3");
```

使用洗衣机噪声向量创建一个随机噪声段。

```
randind = randi(numel(noise) - numel(cleanAudio), [1 1]);
noiseSegment = noise(randind : randind + numel(cleanAudio) - 1);
```

向语音信号添加噪声，使 SNR 为 0 dB。

```
noisePower = sum(noiseSegment.^2);
cleanPower = sum(cleanAudio.^2);
noiseSegment = noiseSegment .* sqrt(cleanPower/noisePower);
noisyAudio = cleanAudio + noiseSegment;
```

使用 stft 基于含噪音频信号生成幅值 STFT 向量。

```
noisySTFT = stft(noisyAudio,'Window',win,'OverlapLength',overlap,'FFTLength',ffTLength);
noisyPhase = angle(noisySTFT(numFeatures-1:end,:));
noisySTFT = abs(noisySTFT(numFeatures-1:end,:));
```

基于含噪 STFT 生成包含 8 个段的训练预测变量信号。连续预测变量之间的重叠是 7 个段。

```
noisySTFT = [noisySTFT(:,1:numSegments-1) noisySTFT];
predictors = zeros( numFeatures, numSegments , size(noisySTFT,2) - numSegments + 1);
for index = 1:(size(noisySTFT,2) - numSegments + 1)
    predictors(:,:,index) = noisySTFT(:,index:index + numSegments - 1);
end
```

通过训练阶段计算的均值和标准差将预测变量归一化。

```
predictors(:) = (predictors(:) - noisyMean) / noisyStd;
```

通过使用 `predict` 和两个经过训练的网络计算去噪后的 STFT 幅值。

```
predictors = reshape(predictors, [numFeatures,numSegments,1,size(predictors,3)]);
STFTFullyConnected = predict(denoiseNetFullyConnected, predictors);
STFTFullyConvolutional = predict(denoiseNetFullyConvolutional, predictors);
```

通过在训练阶段使用的均值和标准差来缩放输出。

```
STFTFullyConnected(:) = cleanStd * STFTFullyConnected(:) + cleanMean;
STFTFullyConvolutional(:) = cleanStd * STFTFullyConvolutional(:) + cleanMean;
```

将单侧 STFT 转换为居中 STFT。

```
STFTFullyConnected = STFTFullyConnected.'.* exp(1j*noisyPhase);
STFTFullyConnected = [conj(STFTFullyConnected(end-1:-1:2,:)); STFTFullyConnected];
STFTFullyConvolutional = squeeze(STFTFullyConvolutional).'.* exp(1j*noisyPhase);
STFTFullyConvolutional = [conj(STFTFullyConvolutional(end-1:-1:2,:)); STFTFullyConvolutional];
```

计算去噪后的语音信号。 `istft` 执行 STFT 逆变换。 使用含噪 STFT 向量的相位来重新构造时域信号。

```
denoisedAudioFullyConnected = istft(STFTFullyConnected, ...
'Window',win,'OverlapLength',overlap, ...
'FFTLength',ffTLength,'ConjugateSymmetric',true);

denoisedAudioFullyConvolutional = istft(STFTFullyConvolutional, ...
'Window',win,'OverlapLength',overlap, ...
'FFTLength',ffTLength,'ConjugateSymmetric',true);
```

分别绘制干净、含噪和去噪的音频信号。

```
t = (1/fs) * (0:numel(denoisedAudioFullyConnected)-1);
```

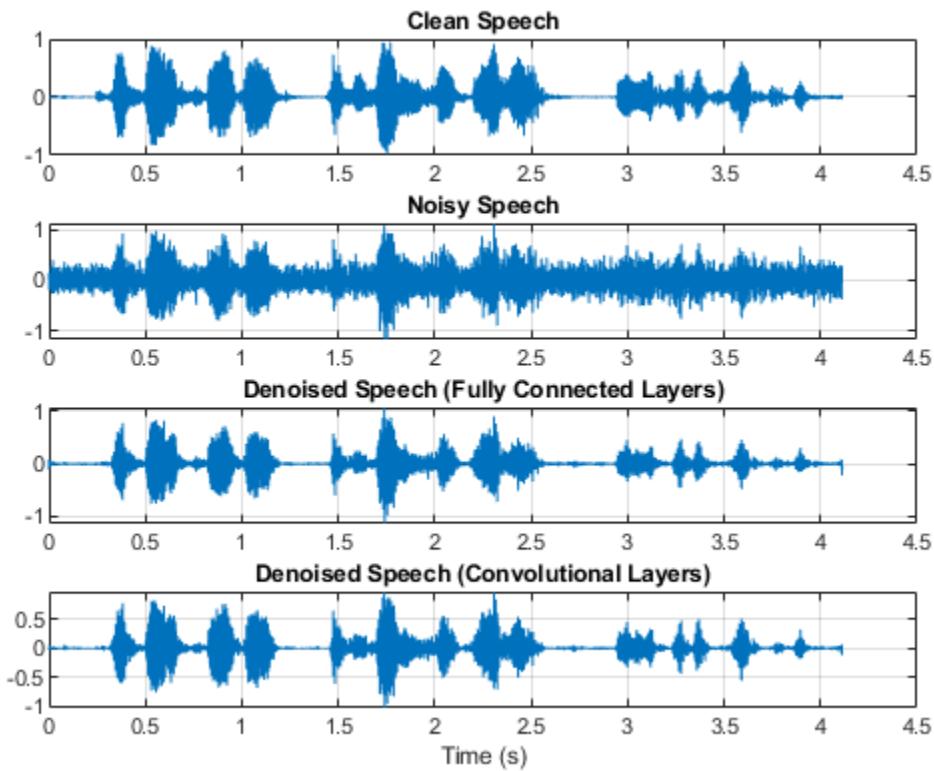
```
figure
```

```
subplot(4,1,1)
plot(t,cleanAudio(1:numel(denoisedAudioFullyConnected)))
title("Clean Speech")
grid on
```

```
subplot(4,1,2)
plot(t,noisyAudio(1:numel(denoisedAudioFullyConnected)))
title("Noisy Speech")
grid on
```

```
subplot(4,1,3)
plot(t,denoisedAudioFullyConnected)
title("Denoised Speech (Fully Connected Layers)")
grid on
```

```
subplot(4,1,4)
plot(t,denoisedAudioFullyConvolutional)
title("Denoised Speech (Convolutional Layers)")
grid on
xlabel("Time (s)")
```



分别绘制干净、含噪和去噪的频谱图。

```

h = figure;

subplot(4,1,1)
spectrogram(cleanAudio,win,overlap,fftLength,fs);
title("Clean Speech")
grid on

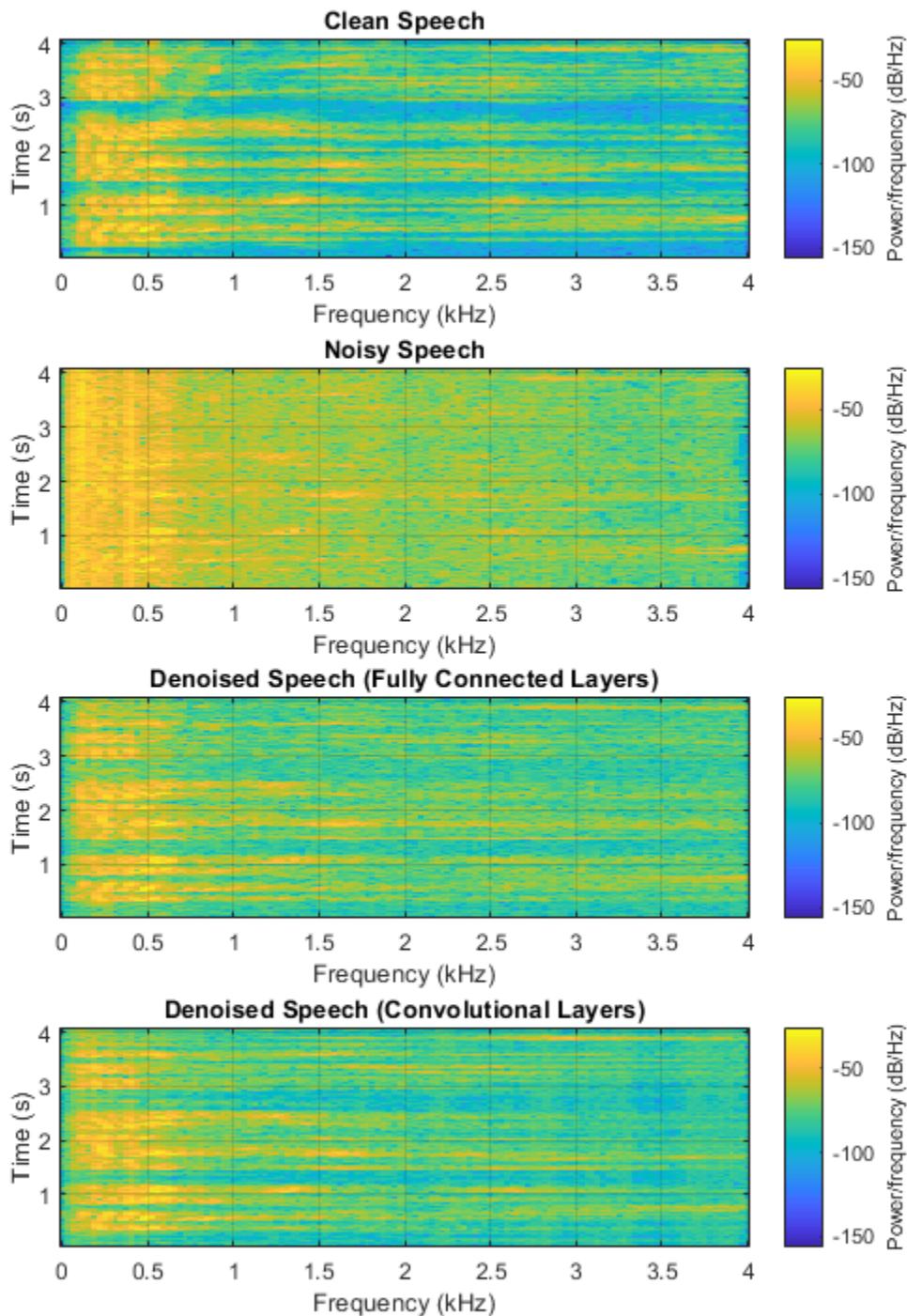
subplot(4,1,2)
spectrogram(noisyAudio,win,overlap,fftLength,fs);
title("Noisy Speech")
grid on

subplot(4,1,3)
spectrogram(denoisedAudioFullyConnected,win,overlap,fftLength,fs);
title("Denoised Speech (Fully Connected Layers)")
grid on

subplot(4,1,4)
spectrogram(denoisedAudioFullyConvolutional,win,overlap,fftLength,fs);
title("Denoised Speech (Convolutional Layers)")
grid on

p = get(h,'Position');
set(h,'Position',[p(1) 65 p(3) 800]);

```



收听含噪语音。

```
sound(noisyAudio,fs)
```

收听经过具有全连接层的网络处理的去噪语音。

```
sound(denoisedAudioFullyConnected,fs)
```

收听经过具有卷积层的网络处理的去噪语音。

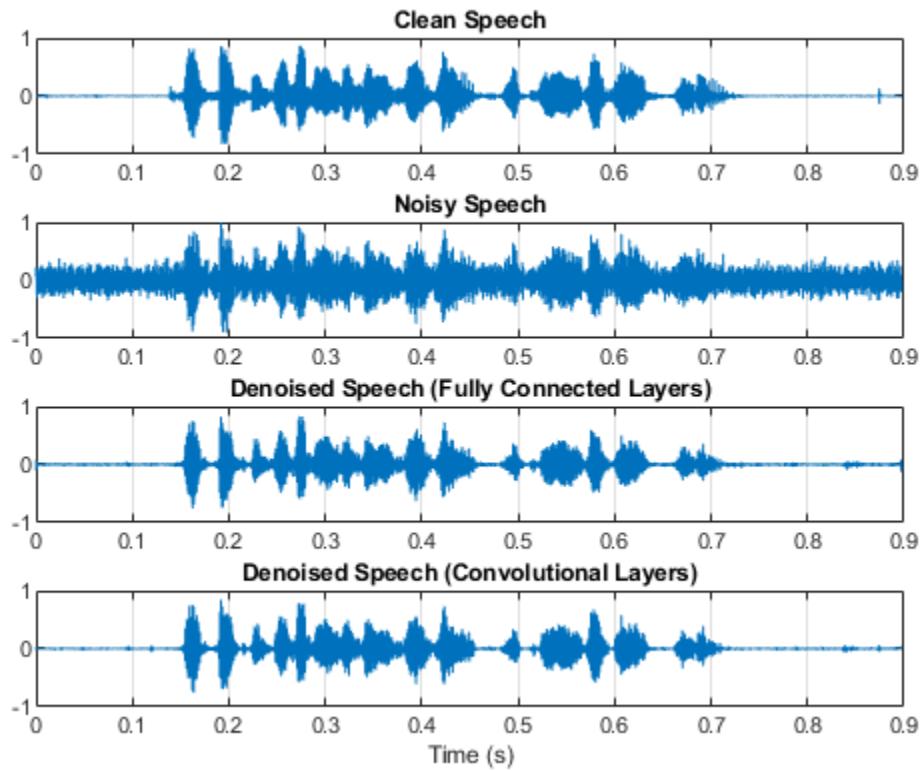
```
sound(denoisedAudioFullyConvolutional,fs)
```

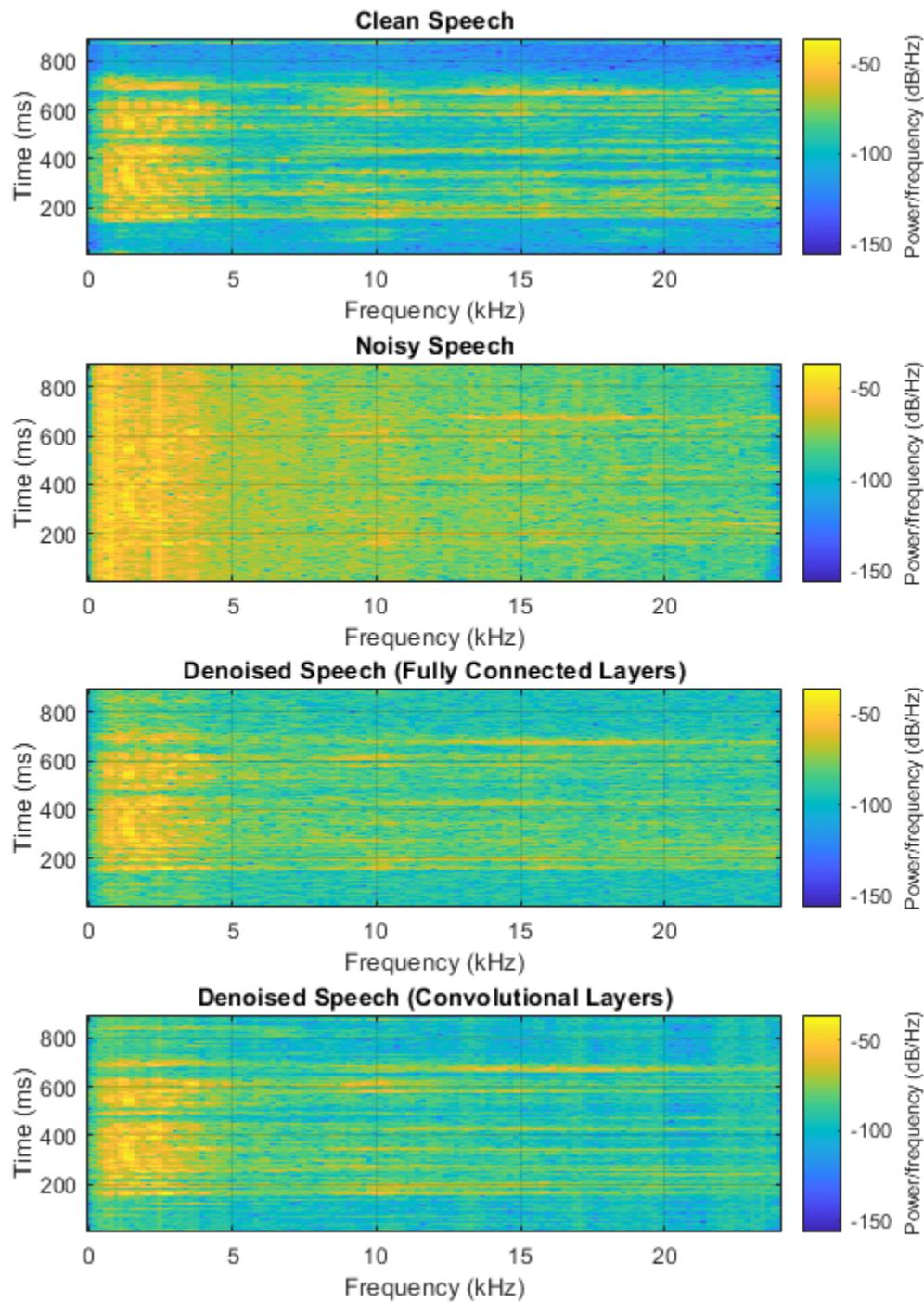
收听干净的语音。

```
sound(cleanAudio,fs)
```

您可以通过调用 `testDenoisingNets` 来测试数据存储中的更多文件。该函数生成上面突出显示的时域绘图和频域绘图，同时返回干净、含噪和去噪的音频信号。

```
[cleanAudio,noisyAudio,denoisedAudioFullyConnected,denoisedAudioFullyConvolutional] = testDenoisingNets
```



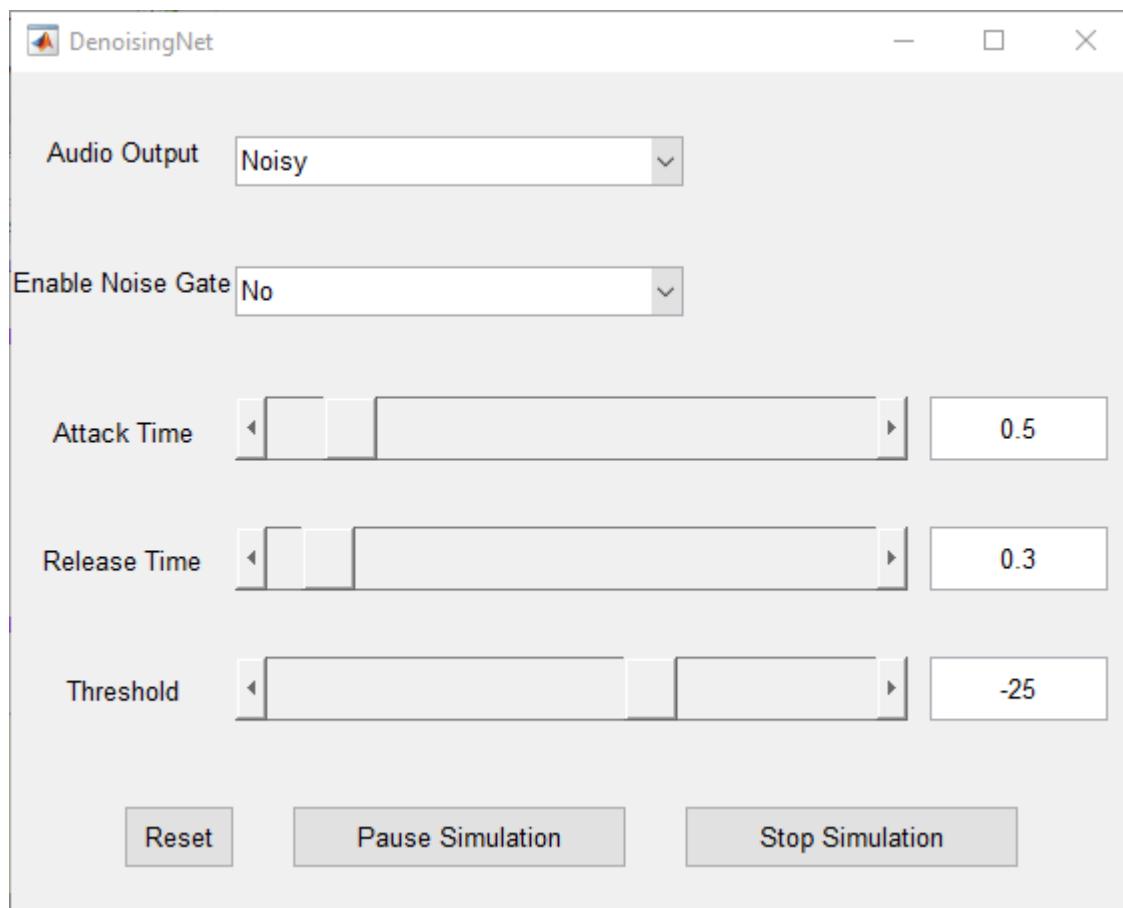


## 实时应用

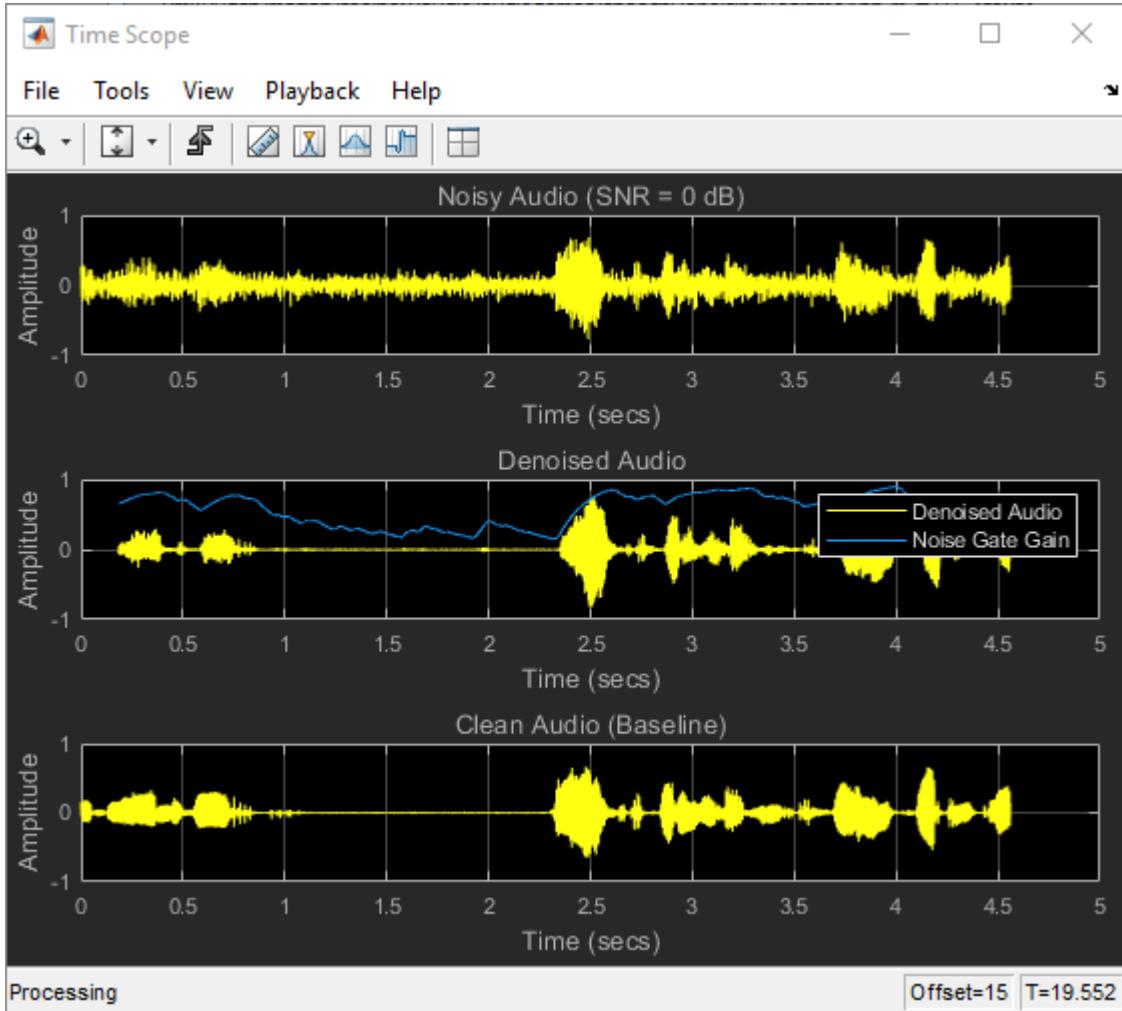
上一节中的过程将含噪信号的整个频谱传递给 `predict`。这不适用于要求低延迟的实时应用。

这里我们将运行 `speechDenoisingRealtimeApp` 以通过一个示例来了解如何仿真一个实时的流式去噪网络。该 App 使用具有全连接层的网络。音频帧长度等于 STFT 跳跃大小，即  $0.25 * 256 = 64$  个样本。

`speechDenoisingRealtimeApp` 会启动一个用于与仿真交互的用户界面 (UI)。该 UI 使您能够调整参数，结果会即时反映在仿真中。您还可以启用/禁用对去噪后的输出进行处理的噪声门，以进一步降低噪声，并调整噪声门的启动时间、释放时间和阈值。您可以从该 UI 收听含噪、干净或去噪的音频。



以下示波器绘制了干净、含噪和去噪的信号，以及噪声门的增益。



## 参考资料

- [1] <https://voice.mozilla.org/en>
- [2] "Experiments on Deep Learning for Speech Denoising", Ding Liu, Paris Smaragdis, Minje Kim, INTERSPEECH, 2014.
- [3] "A Fully Convolutional Neural Network for Speech Enhancement", Se Rim Park, Jin Won Lee, INTERSPEECH, 2017.

## 另请参阅

### 函数

`trainingOptions` | `trainNetwork`

## 详细信息

- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 使用 GRU 网络进行性别分类

此示例说明如何使用深度学习对说话者的性别进行分类。该示例使用门控循环单元 (GRU) 网络和 Gammatone 倒频谱系数 (gtcc)、音调、谐波比和几个频谱形状描述符。

### 简介

基于语音信号的性别分类是许多音频系统的重要组成部分，例如自动语音识别、说话者识别和基于内容的多媒体索引。

本示例使用 GRU 网络，这是一种循环神经网络 (RNN)，非常适合研究序列和时序数据。GRU 网络可以学习序列的时间步之间的长期相关性。

此示例用 Gammatone 倒频谱系数 (gtcc (Audio Toolbox))、音调估计 (pitch (Audio Toolbox))、谐波比 (harmonicRatio (Audio Toolbox)) 和几个频谱形状描述符 (“Spectral Descriptors” (Audio Toolbox)) 序列来训练 GRU 网络。

为了加快训练过程，请在具有 GPU 的机器上运行此示例。如果您的机器同时有 GPU 和 Parallel Computing Toolbox™，则 MATLAB® 会自动使用 GPU 进行训练；否则，它使用 CPU。

### 通过预训练网络进行性别分类

在进入详细训练过程之前，您将使用预训练网络对两个测试信号中说话者的性别进行分类。

下载预训练的网络。

```
url = 'http://ssd.mathworks.com/supportfiles/audio/GenderClassification.zip';
downloadNetFolder = tempdir;
netFolder = fullfile(downloadNetFolder,'GenderClassification');

if ~exist(netFolder,'dir')
    unzip(url,downloadNetFolder)
end
```

加载预训练网络以及用于特性归一化的预先计算的向量。

```
matFileName = fullfile(netFolder, 'genderIDNet.mat');
load(matFileName,'genderIDNet','M','S');
```

加载一个男性说话者的测试信号。

```
[audioIn,Fs] = audioread('maleSpeech.flac');
sound(audioIn,Fs)
```

隔离信号中的语音区域。

```
boundaries = detectSpeech(audioIn,Fs);
audioIn = audioIn(boundaries(1):boundaries(2));
```

创建一个 `audioFeatureExtractor` (Audio Toolbox) 以从音频数据中提取特征。您将使用同一对象来提取特征进行训练。

```
extractor = audioFeatureExtractor( ...
    "SampleRate",Fs, ...
    "Window",hamming(round(0.03*Fs),"periodic"), ...
    "OverlapLength",round(0.02*Fs), ...
```

```

...
"gtcc",true, ...
"gtccDelta",true, ...
"gtccDeltaDelta",true, ...

...
"SpectralDescriptorInput","melSpectrum", ...
"spectralCentroid",true, ...
"spectralEntropy",true, ...
"spectralFlux",true, ...
"spectralSlope",true, ...

...
"pitch",true, ...
"harmonicRatio",true);

```

从信号中提取特征并对其进行归一化。

```

features = extract(extractor, audioIn);
features = (features.' - M)./S;

```

对信号进行分类。

```

gender = classify(genderIDNet, features)

gender = categorical
male

```

对另一个女性说话者的信号进行分类。

```

[audioIn, Fs] = audioread('femaleSpeech.flac');
sound(audioIn, Fs)

boundaries = detectSpeech(audioIn, Fs);
audioIn = audioIn(boundaries(1):boundaries(2));

features = extract(extractor, audioIn);
features = (features.' - M)./S;

classify(genderIDNet, features)

ans = categorical
female

```

## 预处理训练音频数据

此示例中使用的 GRU 网络在使用特征向量序列时效果最佳。为了说明如何预处理管道，此示例逐步演示了针对单个音频文件的步骤。

读取包含语音的音频文件的内容。说话者的性别是男性。

```

[audioIn, Fs] = audioread('Counting-16-44p1-mono-15secs.wav');
labels = {'male'};

```

绘制音频信号，然后使用 **sound** 命令收听它。

```

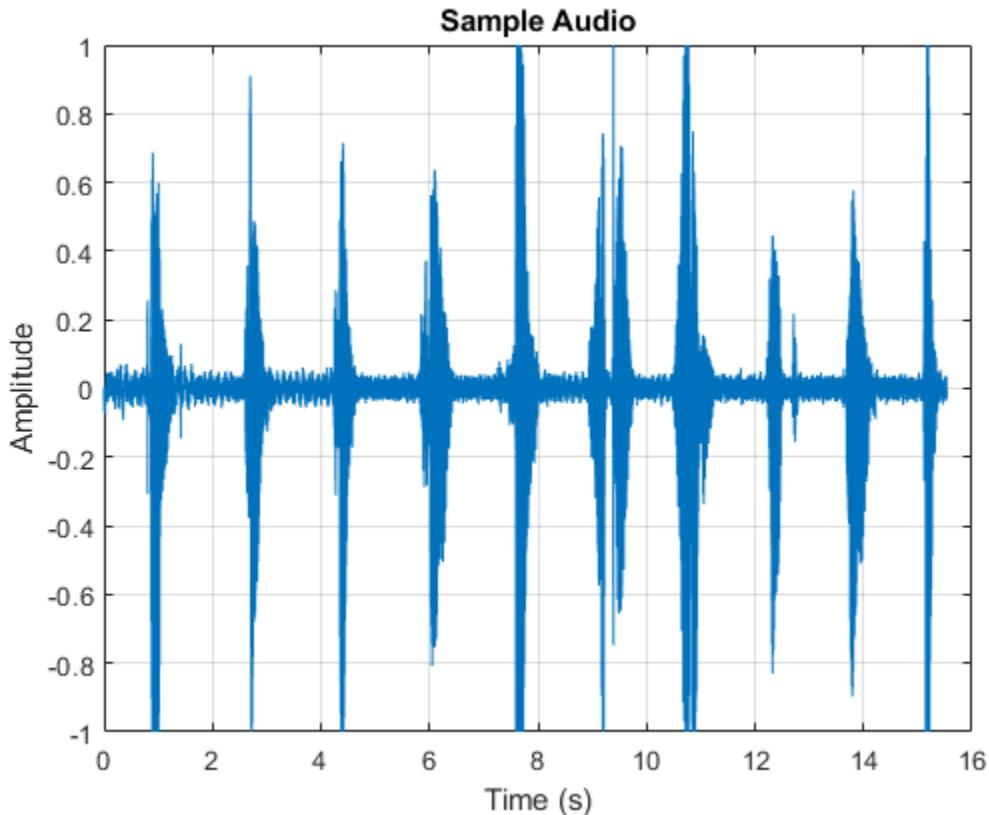
timeVector = (1/Fs) * (0:size(audioIn,1)-1);
figure
plot(timeVector, audioIn)

```

```

ylabel("Amplitude")
xlabel("Time (s)")
title("Sample Audio")
grid on

```



```
sound(audioIn,Fs)
```

语音信号具有静默段，其中不包含与说话者性别相关的有用信息。使用 **detectSpeech** (Audio Toolbox) 定位音频信号中的语音段。

```
speechIndices = detectSpeech(audioIn,Fs);
```

创建一个 **audioFeatureExtractor** (Audio Toolbox) 以从音频数据中提取特征。语音信号在本质上是动态的，并且随着时间而变化。假设语音信号在短时间尺度上是稳定的，并且其处理通常在 20-40 毫秒的时间窗中完成。指定 30 毫秒时间窗，其中包含 20 毫秒重叠。

```

extractor = audioFeatureExtractor( ...
    "SampleRate",Fs, ...
    "Window",hamming(round(0.03*Fs),"periodic"), ...
    "OverlapLength",round(0.02*Fs), ...
    ...
    "gtcc",true, ...
    "gtccDelta",true, ...
    "gtccDeltaDelta",true, ...
    ...
    "SpectralDescriptorInput","melSpectrum", ...
    "spectralCentroid",true, ...
    "spectralEntropy",true, ...

```

```

"spectralFlux",true, ...
"spectralSlope",true, ...
...
"pitch",true, ...
"harmonicRatio",true);

```

从每个音频段中提取特征。`audioFeatureExtractor` 的输出是一个 `numFeatureVectors × numFeatures` 数组。此示例中使用的 `sequenceInputLayer` 需要时间沿第二个维度排列。置换输出数组，使时间沿第二个维度排列。

```

featureVectorsSegment = {};
for ii = 1:size(speechIndices,1)
    featureVectorsSegment{end+1} = ( extract(extractor,audioIn(speechIndices(ii,1):speechIndices(ii,2))) );
end
numSegments = size(featureVectorsSegment)

numSegments = 1×2

```

1 11

```
[numFeatures,numFeatureVectorsSegment1] = size(featureVectorsSegment{1})
```

```
numFeatures = 45
```

```
numFeatureVectorsSegment1 = 124
```

复制标签，使它们与音频段一一对应。

```

labels = repelem(labels,size(speechIndices,1))

labels = 1×11 cell
{'male'} {'male'}

```

当使用 `sequenceInputLayer` 时，通常使用长度一致的序列最合适。将特征向量数组转换为特征向量序列。对每个序列使用 20 个特征向量，其中 5 个特征向量重叠。

```

featureVectorsPerSequence = 20;
featureVectorOverlap = 5;
hopLength = featureVectorsPerSequence - featureVectorOverlap;

idx1 = 1;
featuresTrain = {};
sequencePerSegment = zeros(numel(featureVectorsSegment),1);
for ii = 1:numel(featureVectorsSegment)
    sequencePerSegment(ii) = max(floor((size(featureVectorsSegment{ii},2) - featureVectorsPerSequence)/hopLength));
    idx2 = 1;
    for j = 1:sequencePerSegment(ii)
        featuresTrain{idx1,1} = featureVectorsSegment{ii}(:,idx2:idx2 + featureVectorsPerSequence - 1);
        idx1 = idx1 + 1;
        idx2 = idx2 + hopLength;
    end
end

```

为简明起见，辅助函数 `HelperFeatureVector2Sequence`（第 14-0 页）封装上述处理，并在此示例的其余部分中使用。

复制标签，使它们与训练集一一对应。

```
labels = repelem(labels,sequencePerSegment);
```

预处理管道的结果是由若干个  $\text{NumFeatures} \times \text{FeatureVectorsPerSequence}$  矩阵组成的一个  $\text{NumSequence} \times 1$  元胞数组。labels 是一个  $\text{NumSequence} \times 1$  数组。

```
NumSequence = numel(featuresTrain)
```

```
NumSequence = 27
```

```
[NumFeatures,FeatureVectorsPerSequence] = size(featuresTrain{1})
```

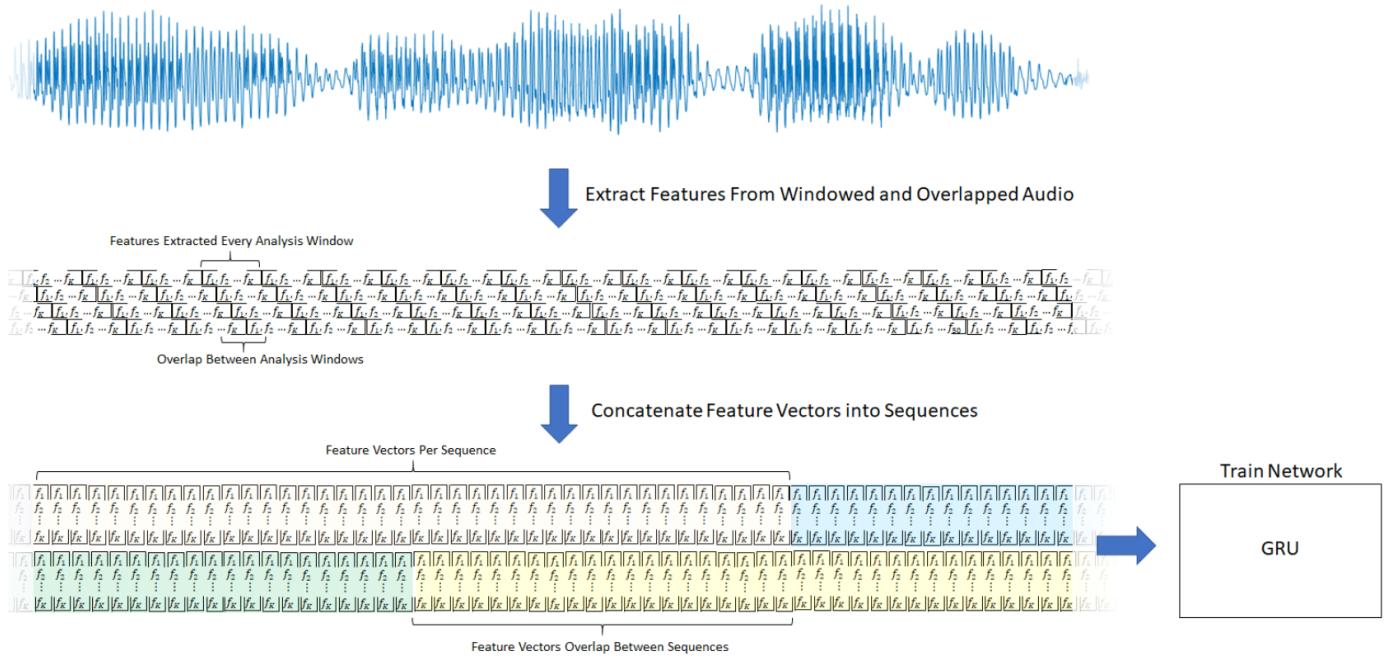
```
NumFeatures = 45
```

```
FeatureVectorsPerSequence = 20
```

```
NumSequence = numel(labels)
```

```
NumSequence = 27
```

下图简要显示每个检测到的语音区域所使用的特征提取。



### 创建训练和测试数据存储

此示例使用 Mozilla 通用语音数据集的一个子集 [1] (第 14-0 页)。该数据集包含测试人员口述短句的 48 kHz 录音。下载该数据集并解压缩下载的文件。将 PathToDatabase 设置为数据的位置。

```
url = 'http://ssd.mathworks.com/supportfiles/audio/commonvoice.zip';
downloadDatasetFolder = tempdir;
dataFolder = fullfile(downloadDatasetFolder,'commonvoice');

if ~exist(dataFolder,'dir')
    disp('Downloading data set (956 MB) ...')
    unzip(url,downloadDatasetFolder)
end
```

```
Downloading data set (956 MB) ...
```

使用 `audioDatastore` 为训练集和验证集创建数据存储。使用 `readtable` 读取与音频文件相关联的元数据。

```
loc = fullfile(dataFolder);
adsTrain = audioDatastore(fullfile(loc,'train'),'IncludeSubfolders',true);
metadataTrain = readtable(fullfile(loc,'train"),"train.tsv","FileType","text");
adsTrain.Labels = metadataTrain.gender;

adsValidation = audioDatastore(fullfile(loc,'validation'),'IncludeSubfolders',true);
metadataValidation = readtable(fullfile(loc,'validation"),"validation.tsv","FileType","text");
adsValidation.Labels = metadataValidation.gender;
```

使用 `countEachLabel` (Audio Toolbox) 检查训练集和验证集的性别分类。

```
countEachLabel(adsTrain)
```

```
ans=2×2 table
  Label    Count
  _____
  female    1000
  male      1000
```

```
countEachLabel(adsValidation)
```

```
ans=2×2 table
  Label    Count
  _____
  female    200
  male      200
```

要使用整个数据集来训练网络并达到尽可能最高的准确度，请将 `reduceDataset` 设置为 `false`。要快速运行此示例，请将 `reduceDataset` 设置为 `true`。

```
reduceDataset = false;
if reduceDataset
    % Reduce the training dataset by a factor of 20
    adsTrain = splitEachLabel(adsTrain,round(numel(adsTrain.Files) / 2 / 20));
    adsValidation = splitEachLabel(adsValidation,20);
end
```

### 创建训练集和验证集

确定数据集中音频文件的采样率，然后更新音频特征提取器的采样率、窗口和重叠长度。

```
[~,adsInfo] = read(adsTrain);
Fs = adsInfo.SampleRate;
extractor.SampleRate = Fs;
extractor.Window = hamming(round(0.03*Fs),"periodic");
extractor.OverlapLength = round(0.02*Fs);
```

为了加快处理速度，请将计算分配给多个工作进程。如果您有 Parallel Computing Toolbox™，该示例将对数据存储进行分区，以便在可用的各工作进程之间以并行方式进行特征提取。确定系统的最佳分区数量。如果您没有 Parallel Computing Toolbox™，该示例将使用单一工作进程。

```
if ~isempty(ver('parallel')) && ~reduceDataset
    pool = gcp;
```

```

numPar = numpartitions(adsTrain,pool);
else
    numPar = 1;
end

```

Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 6).

以循环方式：

- 1 从音频数据存储中读取。
- 2 检测语音区域。
- 3 从语音区域中提取特征向量。

复制标签，使它们与特征向量一一对应。

```

labelsTrain = [];
featureVectors = {};

% Loop over optimal number of partitions
parfor ii = 1:numPar

    % Partition datastore
    subds = partition(adsTrain,numPar,ii);

    % Preallocation
    featureVectorsInSubDS = {};
    segmentsPerFile = zeros(numel(subds.Files),1);

    % Loop over files in partitioned datastore
    for jj = 1:numel(subds.Files)

        % 1. Read in a single audio file
        audioIn = read(subds);

        % 2. Determine the regions of the audio that correspond to speech
        speechIndices = detectSpeech(audioIn,Fs);

        % 3. Extract features from each speech segment
        segmentsPerFile(jj) = size(speechIndices,1);
        features = cell(segmentsPerFile(jj),1);
        for kk = 1:size(speechIndices,1)
            features{kk} = ( extract(extractor,audioIn(speechIndices(kk,1):speechIndices(kk,2))) )';
        end
        featureVectorsInSubDS = [featureVectorsInSubDS;features{:}];

    end
    featureVectors = [featureVectors;featureVectorsInSubDS];

    % Replicate the labels so that they are in one-to-one correspondance
    % with the feature vectors.
    repedLabels = repelem(subds.Labels,segmentsPerFile);
    labelsTrain = [labelsTrain;repedLabels(:)];
end

```

在分类应用中，将所有特征归一化为具有零均值和单位标准差是很好的做法。

计算每个系数的均值和标准差，并使用它们来归一化数据。

```

allFeatures = cat(2,featureVectors{:});
allFeatures(isinf(allFeatures)) = nan;
M = mean(allFeatures,2,'omitnan');
S = std(allFeatures,0,2,'omitnan');
featureVectors = cellfun(@(x)(x-M)./S,featureVectors,'UniformOutput',false);
for ii = 1:numel(featureVectors)
    idx = find(isnan(featureVectors{ii}));
    if ~isempty(idx)
        featureVectors{ii}(idx) = 0;
    end
end

```

将特征向量缓冲到具有 20 个特征向量及 10 个重叠的序列中。如果一个序列的特征向量少于 20 个，则将其丢弃。

```
[featuresTrain,trainSequencePerSegment] = HelperFeatureVector2Sequence(featureVectors,featureVectorsPerSegment);
复制标签，使它们与序列一一对应。
```

```
labelsTrain = repelem(labelsTrain,[trainSequencePerSegment{:}]);
labelsTrain = categorical(labelsTrain);
```

使用与创建训练集相同的步骤创建验证集。

```

labelsValidation = [];
featureVectors = {};
valSegmentsPerFile = [];
parfor ii = 1:numPar
    subds = partition(adsValidation,numPar,ii);
    featureVectorsInSubDS = {};
    valSegmentsPerFileInSubDS = zeros(numel(subds.Files),1);
    for jj = 1:numel(subds.Files)
        audioIn = read(subds);
        speechIndices = detectSpeech(audioIn,Fs);
        numSegments = size(speechIndices,1);
        features = cell(valSegmentsPerFileInSubDS(jj),1);
        for kk = 1:numSegments
            features{kk} = ( extract(extractor,audioIn(speechIndices(kk,1):speechIndices(kk,2))) );
        end
        featureVectorsInSubDS = [featureVectorsInSubDS;features(:)];
        valSegmentsPerFileInSubDS(jj) = numSegments;
    end
    repedLabels = repelem(subds.Labels,valSegmentsPerFileInSubDS);
    labelsValidation = [labelsValidation;repedLabels(:)];
    featureVectors = [featureVectors;featureVectorsInSubDS];
    valSegmentsPerFile = [valSegmentsPerFile;valSegmentsPerFileInSubDS];
end

featureVectors = cellfun(@(x)(x-M)./S,featureVectors,'UniformOutput',false);
for ii = 1:numel(featureVectors)
    idx = find(isnan(featureVectors{ii}));
    if ~isempty(idx)
        featureVectors{ii}(idx) = 0;
    end
end

[featuresValidation,valSequencePerSegment] = HelperFeatureVector2Sequence(featureVectors,featureVectorsPerSegment);
labelsValidation = repelem(labelsValidation,[valSequencePerSegment{:}]);
labelsValidation = categorical(labelsValidation);
```

## 定义 GRU 网络架构

GRU 网络可以学习序列数据的时间步之间的长期相关性。此示例使用 `gruLayer` 对序列进行前向和后向处理。

将输入大小指定为大小为 `NumFeatures` 的序列。指定输出大小为 75 的一个 GRU 层，并输出序列。然后，指定一个输出大小为 75 的 GRU 层，并输出序列的最后一个元素。此命令指示 GRU 层将其输入映射到 75 个特征，然后准备好输出到全连接层。最后，通过包含大小为 2 的全连接层，后跟 `softmax` 层和分类层，来指定两个类。

```
layers = [...  
    sequenceInputLayer(size(featuresTrain{1},1))  
    gruLayer(75,"OutputMode","sequence")  
    gruLayer(75,"OutputMode","last")  
    fullyConnectedLayer(2)  
    softmaxLayer  
    classificationLayer];
```

接下来，指定分类器的训练选项。将 `MaxEpochs` 设置为 4 以便基于训练数据对网络进行 4 轮训练。将 `MiniBatchSize` 设置为 256 以便网络可以一次查看 128 个训练信号。将 `Plots` 指定为 "training-progress" 以生成随着迭代次数增加显示训练进度的图。将 `Verbose` 设置为 `false` 以禁止打印对应于图中所示数据的表输出。将 `Shuffle` 指定为 "every-epoch" 以便在每轮开始时使训练序列变为乱序。将 `LearnRateSchedule` 指定为 "piecewise" 以便每经过一定数量的轮次 (1) 时，按指定的因子 (0.1) 降低学习率。

此示例使用自适应矩估计 (ADAM) 求解器。与默认的具有动量的随机梯度下降 (SGDM) 求解器相比，ADAM 在使用 GRU 之类的循环神经网络 (RNN) 时性能更好。

```
miniBatchSize = 256;  
validationFrequency = floor(numel(labelsTrain)/miniBatchSize);  
options = trainingOptions("adam", ...  
    "MaxEpochs",4, ...  
    "MiniBatchSize",miniBatchSize, ...  
    "Plots","training-progress", ...  
    "Verbose",false, ...  
    "Shuffle","every-epoch", ...  
    "LearnRateSchedule","piecewise", ...  
    "LearnRateDropFactor",0.1, ...  
    "LearnRateDropPeriod",1, ...  
    'ValidationData',{featuresValidation,labelsValidation}, ...  
    'ValidationFrequency',validationFrequency);
```

## 训练 GRU 网络

使用 `trainNetwork` 用指定的训练选项和层架构训练 GRU 网络。由于训练集很大，训练过程可能需要几分钟。

```
net = trainNetwork(featuresTrain,labelsTrain,layers,options);
```

训练进度图的顶部子图表示训练准确度，即基于每个小批量的分类准确度。当训练在成功进行时，此值通常会逐渐增大，直到 100%。底部子图显示训练损失，即基于每个小批量的交叉熵损失。当训练在成功进行时，该值通常会逐渐降低，直到为零。

如果训练未收敛，绘图可能会在各值之间振荡，而不会呈现向上或向下趋势。这种振荡意味着训练准确度没有提高，训练损失没有减少。这种情况可能发生在训练开始时，或者在训练准确度有一些初步提高后。在许多情况下，更改训练选项可以帮助网络实现收敛。减少 `MiniBatchSize` 或减少 `InitialLearnRate` 可能会导致更长的训练时间，但这可能有助于网络更好地学习。

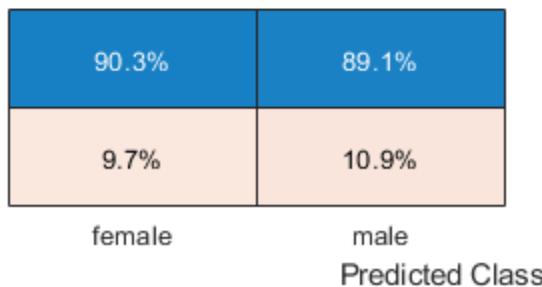
## 可视化训练准确度

计算训练准确度，该准确度表示分类器对于所训练信号的准确度。首先，对训练数据进行分类。

```
prediction = classify(net,featuresTrain);
```

绘制混淆矩阵。使用列汇总和行汇总显示两个类的精确率和召回率。

```
figure
cm = confusionchart(categorical(labelsTrain),prediction,'title','Training Accuracy');
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
```



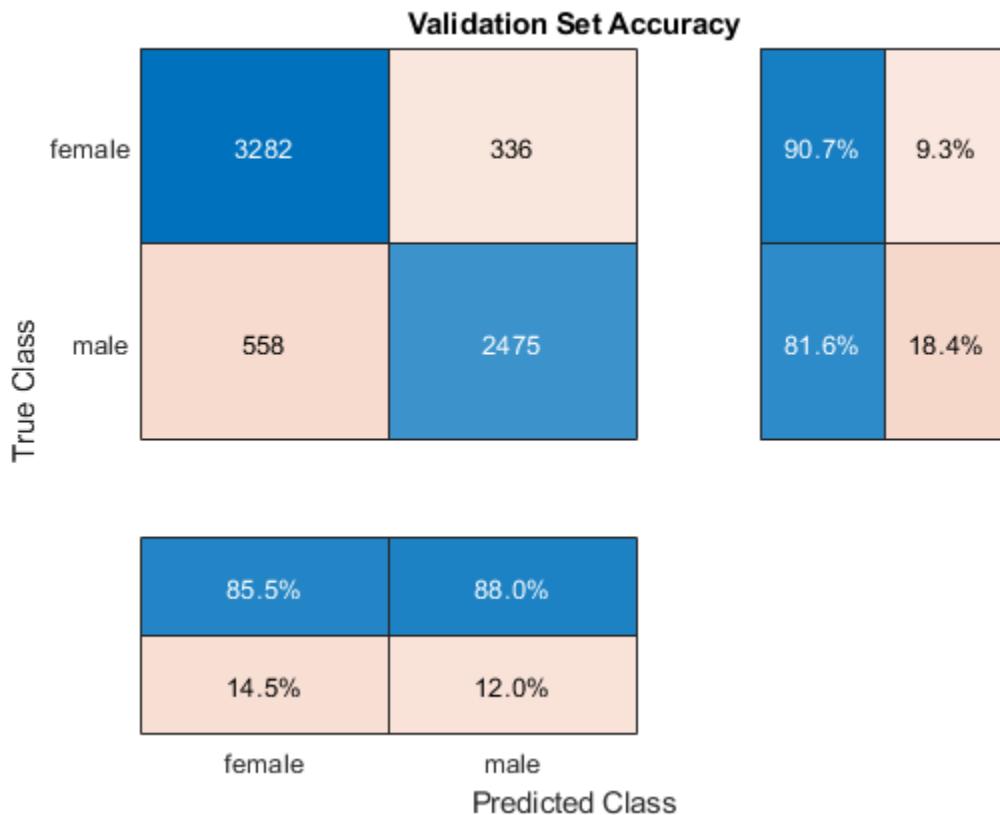
## 可视化验证准确度

计算验证准确度。首先，对训练数据进行分类。

```
[prediction,probabilities] = classify(net,featuresValidation);
```

绘制混淆矩阵。使用列汇总和行汇总显示两个类的精确率和召回率。

```
figure
cm = confusionchart(categorical(labelsValidation),prediction,'title','Validation Set Accuracy');
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
```



该示例基于每个训练语音文件生成多个序列。如果考虑对应于同一个文件的所有序列的输出类，并应用“最大规则”决策从中选择置信度分数最高的语音段所在的类，则可以获得更高的准确度。

确定验证集中每个文件生成的序列数。

```
sequencePerFile = zeros(size(valSegmentsPerFile));
valSequencePerSegmentMat = cell2mat(valSequencePerSegment);
idx = 1;
for ii = 1:numel(valSegmentsPerFile)
    sequencePerFile(ii) = sum(valSequencePerSegmentMat(idx:idx+valSegmentsPerFile(ii)-1));
    idx = idx + valSegmentsPerFile(ii);
end
```

通过分析从同一文件生成的所有序列的输出类，基于每个训练文件预测性别。

```
numFiles = numel(adsValidation.Files);
actualGender = categorical(adsValidation.Labels);
predictedGender = actualGender;
scores = cell(1,numFiles);
counter = 1;
cats = unique(actualGender);
for index = 1:numFiles
    scores{index} = probabilities(counter: counter + sequencePerFile(index) - 1,:);
    m = max(mean(scores{index},1,[],1));
    if m(1) >= m(2)
        predictedGender(index) = cats(1);
    else
        predictedGender(index) = cats(2);
```

```

end
counter = counter + sequencePerFile(index);
end

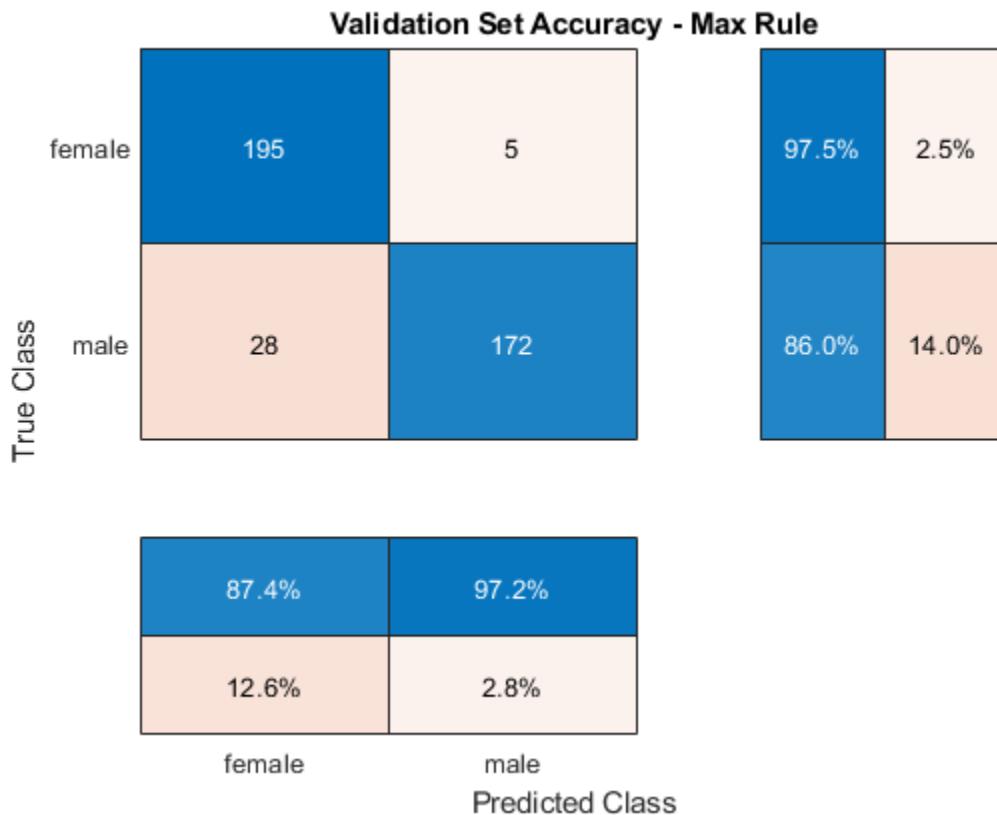
```

可视化基于多数规则预测的混淆矩阵。

```

figure
cm = confusionchart(actualGender,predictedGender,'title','Validation Set Accuracy - Max Rule');
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';

```



## 参考资料

[1] Mozilla 通用语音数据集

## 支持函数

```

function [sequences,sequencePerSegment] = HelperFeatureVector2Sequence(features,featureVectorsPerSequence)
if featureVectorsPerSequence <= featureVectorOverlap
    error('The number of overlapping feature vectors must be less than the number of feature vectors per sequence')
end

hopLength = featureVectorsPerSequence - featureVectorOverlap;
idx1 = 1;
sequences = {};
sequencePerSegment = cell(numel(features),1);
for ii = 1:numel(features)
    sequencePerSegment{ii} = max(floor((size(features{ii},2) - featureVectorsPerSequence)/hopLength) + 1,0);
end

```

```
idx2 = 1;
for j = 1:sequencePerSegment{ii}
    sequences{idx1,1} = features{ii}{:,idx2:idx2 + featureVectorsPerSequence - 1}; %#ok<AGROW>
    idx1 = idx1 + 1;
    idx2 = idx2 + hopLength;
end
end
end
```

### 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [lstmLayer](#)

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “长短期记忆网络” (第 1-38 页)

## 强化学习示例

---



# 预测性维护示例

---



# 导入、导出和自定义

---

- “定义自定义深度学习层” (第 17-2 页)
- “指定自定义权重初始化函数” (第 17-16 页)
- “比较层权重初始化函数” (第 17-22 页)
- “基于预训练的 Keras 层组合网络” (第 17-28 页)
- “定义自定义训练循环、损失函数和网络” (第 17-33 页)

## 定义自定义深度学习层

**提示** 本主题说明如何针对您的问题定义自定义深度学习层。有关 Deep Learning Toolbox 中的内置层的列表，请参阅“深度学习层列表”（第 1-18 页）。

本主题说明深度学习层的架构，以及如何针对您的问题定义自定义层。

类型	说明
层	<p>定义一个自定义深度学习层，并指定可选的可学习参数。</p> <p>有关如何使用可学习参数定义自定义层的示例，请参阅“Define Custom Deep Learning Layer with Learnable Parameters”。有关如何定义具有多个输入的自定义层的示例，请参阅“Define Custom Deep Learning Layer with Multiple Inputs”。</p>
分类输出层	<p>定义一个自定义分类输出层并指定损失函数。</p> <p>有关如何定义自定义分类输出层和指定损失函数的示例，请参阅“Define Custom Classification Output Layer”。</p>
回归输出层	<p>定义一个自定义回归输出层并指定损失函数。</p> <p>有关如何定义一个自定义回归输出层和指定损失函数的示例，请参阅“Define Custom Regression Output Layer”。</p>

## 层模板

您可以使用以下模板来定义新层。

### 中间层模板

此模板提供具有可学习参数的中间层的大致结构。如果该层没有可学习参数，则可以省略 **properties (learnable)** 部分。有关如何定义具有可学习参数的层的示例，请参阅“Define Custom Deep Learning Layer with Learnable Parameters”。

```
classdef myLayer < nnet.layer.Layer % & nnet.layer.Formattable (Optional)

    properties
        % (Optional) Layer properties.

        % Layer properties go here.
    end

    properties (Learnable)
        % (Optional) Layer learnable parameters.

        % Layer learnable parameters go here.
    end

    methods
        function layer = myLayer()
            % (Optional) Create a myLayer.
            % This function must have the same name as the class.
        end
    end
```

```

% Layer constructor function goes here.
end

function [Z1, ..., Zm] = predict(layer, X1, ..., Xn)
    % Forward input data through the layer at prediction time and
    % output the result.
    %
    % Inputs:
    %     layer      - Layer to forward propagate through
    %     X1, ..., Xn - Input data
    % Outputs:
    %     Z1, ..., Zm - Outputs of layer forward function

    % Layer forward function for prediction goes here.
end

function [Z1, ..., Zm, memory] = forward(layer, X1, ..., Xn)
    % (Optional) Forward input data through the layer at training
    % time and output the result and a memory value.
    %
    % Inputs:
    %     layer      - Layer to forward propagate through
    %     X1, ..., Xn - Input data
    % Outputs:
    %     Z1, ..., Zm - Outputs of layer forward function
    %     memory      - Memory value for custom backward propagation

    % Layer forward function for training goes here.
end

function [dLdX1, ..., dLdXn, dLdW1, ..., dLdWk] = ...
    backward(layer, X1, ..., Xn, Z1, ..., Zm, dLdZ1, ..., dLdZm, memory)
    % (Optional) Backward propagate the derivative of the loss
    % function through the layer.
    %
    % Inputs:
    %     layer      - Layer to backward propagate through
    %     X1, ..., Xn      - Input data
    %     Z1, ..., Zm      - Outputs of layer forward function
    %     dLdZ1, ..., dLdZm - Gradients propagated from the next layers
    %     memory      - Memory value from forward function
    % Outputs:
    %     dLdX1, ..., dLdXn - Derivatives of the loss with respect to the
    %                         inputs
    %     dLdW1, ..., dLdWk - Derivatives of the loss with respect to each
    %                         learnable parameter

    % Layer backward function goes here.
end
end
end

```

## 分类输出层模板

此模板提供具有损失函数的分类输出层的大致结构。有关如何定义分类输出层和指定损失函数的示例，请参阅“Define Custom Classification Output Layer”。

```

classdef myClassificationLayer < nnet.layer.ClassificationLayer

properties
    % (Optional) Layer properties.

    % Layer properties go here.
end

methods
    function layer = myClassificationLayer()
        % (Optional) Create a myClassificationLayer.

        % Layer constructor function goes here.
    end

```

```

end

function loss = forwardLoss(layer, Y, T)
    % Return the loss between the predictions Y and the training
    % targets T.
    %
    % Inputs:
    %     layer - Output layer
    %     Y   - Predictions made by network
    %     T   - Training targets
    %
    % Output:
    %     loss - Loss between Y and T

    % Layer forward loss function goes here.
end

function dLdY = backwardLoss(layer, Y, T)
    % (Optional) Backward propagate the derivative of the loss
    % function.
    %
    % Inputs:
    %     layer - Output layer
    %     Y   - Predictions made by network
    %     T   - Training targets
    %
    % Output:
    %     dLdY - Derivative of the loss with respect to the
    %            predictions Y

    % Layer backward loss function goes here.
end
end
end

```

## 回归输出层模板

此模板提供具有损失函数的回归输出层的大致结构。有关如何定义回归输出层和指定损失函数的示例，请参阅“Define Custom Regression Output Layer”。

```

classdef myRegressionLayer < nnet.layer.RegressionLayer

properties
    % (Optional) Layer properties.

    % Layer properties go here.
end

methods
    function layer = myRegressionLayer()
        % (Optional) Create a myRegressionLayer.

        % Layer constructor function goes here.
    end

    function loss = forwardLoss(layer, Y, T)
        % Return the loss between the predictions Y and the training
        % targets T.
        %
        % Inputs:
        %     layer - Output layer
        %     Y   - Predictions made by network
        %     T   - Training targets
        %
        % Output:
        %     loss - Loss between Y and T

        % Layer forward loss function goes here.
    end

    function dLdY = backwardLoss(layer, Y, T)
        % (Optional) Backward propagate the derivative of the loss
        % function.
        %
        % Inputs:

```

```
% layer - Output layer
% Y - Predictions made by network
% T - Training targets
%
% Output:
% dLdY - Derivative of the loss with respect to the
% predictions Y
%
% Layer backward loss function goes here.
end
end
```

## 中间层架构

在训练过程中，软件在网络中以迭代方式执行前向传导和后向传导。

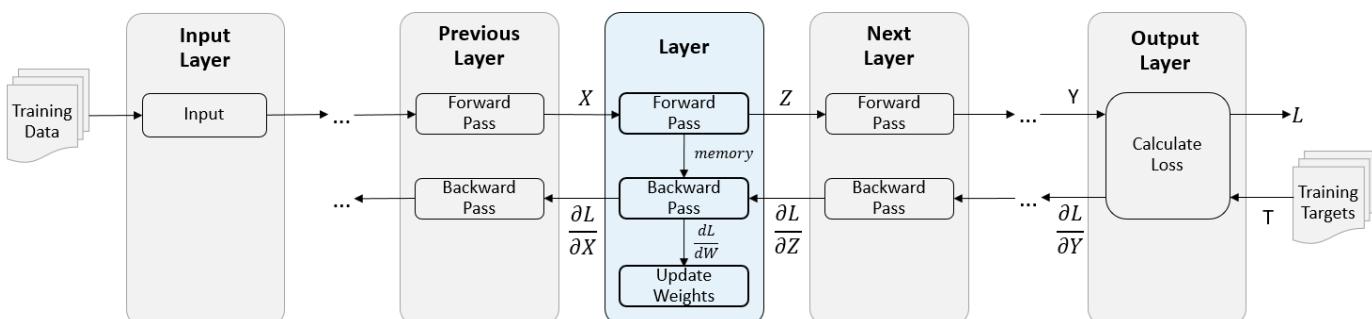
在网络中执行前向传导时，每个层获取前面各层的输出，应用一个函数，然后将结果输出（前向传播）到后面各层。

层可以有多个输入或输出。例如，层可以从多个前面的层中获取  $X_1, \dots, X_n$ ，并将输出  $Z_1, \dots, Z_m$  前向传播到后面各层。

在执行网络的前向传导结束时，输出层计算预测值  $Y$  和真实目标  $T$  之间的损失  $L$ 。

在网络的后向传导过程中，每个层都获取损失关于该层的输出的导数，计算损失  $L$  关于输入的导数，然后后向传播结果。如果层具有可学习参数，则该层还会计算层权重（可学习参数）的导数。该层使用权重的导数来更新可学习参数。

下图说明通过深度神经网络的数据流，重点显示通过具有单个输入  $X$ 、单个输出  $Z$  和可学习参数  $W$  的层的数据流。



## 中间层属性

在类定义的 **properties** 部分声明层属性。

默认情况下，自定义中间层具有以下属性。

属性	说明
Name	层名称，指定为字符向量或字符串标量。要将一个层包括在层次图中，您必须指定非空的唯一层名称。如果您使用该层训练串行网络并且 Name 设置为 " "，则软件会在训练时自动为该层指定名称。

属性	说明
Description	层的单行描述，指定为字符向量或字符串标量。当层显示在 Layer 数组中时，会出现此描述。如果没有指定层描述，则软件将显示层类名。
Type	层的类型，指定为字符向量或字符串标量。当层显示在 Layer 数组中时，会显示 Type 的值。如果没有指定层类型，则软件将显示层类名。
NumInputs	层的输入的数目，指定为正整数。如果您未指定此值，软件会自动将 NumInputs 设置为 InputNames 中的名称的数目。默认值为 1。
InputNames	层的输入名称，指定为字符向量元胞数组。如果您未指定此值并且 NumInputs 大于 1，则软件会自动将 InputNames 设置为 {'in1',..., 'inN'}，其中 N 等于 NumInputs。默认值为 {'in'}。
NumOutputs	层的输出的数目，指定为正整数。如果您未指定此值，软件会自动将 NumOutputs 设置为 OutputNames 中的名称的数目。默认值为 1。
OutputNames	层的输出名称，指定为字符向量元胞数组。如果您未指定此值并且 NumOutputs 大于 1，则软件会自动将 OutputNames 设置为 {'out1',..., 'outM'}，其中 M 等于 NumOutputs。默认值为 {'out'}。

如果层没有其他属性，则您可以省略 properties 部分。

**提示** 如果要创建一个具有多个输入的层，则必须在层构造函数中设置 NumInputs 或 InputNames 属性。如果要创建一个具有多个输出的层，则必须在层构造函数中设置 NumOutputs 或 OutputNames 属性。有关示例，请参阅“Define Custom Deep Learning Layer with Multiple Inputs”。

### 可学习参数

在类定义的 properties (Learnable) 部分声明层可学习参数。您可以将数值数组或 dlnetwork 对象指定为可学习参数。如果层没有可学习参数，则可以省略 properties (Learnable) 部分。

您也可以指定可学习参数的学习率因子和 L2 因子。默认情况下，每个可学习参数的学习率因子和 L2 因子都设置为 1。

对于内置层和自定义层，可以使用以下函数设置和获取学习率因子和 L2 正则化因子。

函数	说明
setLearnRateFactor	设置可学习参数的学习率因子。
setL2Factor	设置可学习参数的 L2 正则化因子。
getLearnRateFactor	获取可学习参数的学习率因子。
getL2Factor	获取可学习参数的 L2 正则化因子。

要指定可学习参数的学习率因子和 L2 因子，请分别使用语法 `layer = setLearnRateFactor(layer,'MyParameterName',value)` 和 `layer = setL2Factor(layer,parameterName,value)`。

要获得可学习参数的学习率因子和 L2 因子的值，请分别使用语法  
`getLearnRateFactor(layer,'MyParameterName')` 和 `getL2Factor(layer,parameterName)`。

例如，以下语法将名为 'Alpha' 的可学习参数的学习率因子设置为 0.1。

```
layer = setLearnRateFactor(layer,'Alpha',0.1);
```

### 前向函数

一些层在训练期间的行为和其在预测期间的行为是不同的。例如，丢弃层仅在训练期间执行丢弃，在预测期间不起作用。层使用 `predict` 或 `forward` 函数来执行前向传导。如果前向传导在预测时间进行，则该层使用 `predict` 函数。如果前向传导在训练时进行，则该层使用 `forward` 函数。如果不需要对预测时间和训练时间使用两个不同函数，则可以省略 `forward` 函数。在本例中，层在训练时使用 `predict`。

如果您定义函数 `forward` 和自定义 `backward` 函数，则必须为参数 `memory` 指定值，您可以在后向传播期间使用该值。

`predict` 的语法是 `[Z1,...,Zm] = predict(layer,X1,...,Xn)`，其中 `X1,...,Xn` 是 `n` 个层输入，`Z1,...,Zm` 是 `m` 个层输出。`n` 和 `m` 的值必须对应于该层的 `NumInputs` 和 `NumOutputs` 属性。

**提示** 如果 `predict` 的输入的数目可能变化，则使用 `varargin` 代替 `X1,...,Xn`。在本例中，`varargin` 是由输入组成的元胞数组，其中 `varargin{i}` 对应于 `Xi`。如果输出的数目可能变化，则使用 `varargout` 代替 `Z1,...,Zm`。在本例中，`varargout` 是由输出组成的元胞数组，其中 `varargout{j}` 对应于 `Zj`。

**提示** 如果自定义层具有一个作为可学习参数的 `dlnetwork` 对象，则在自定义层的 `predict` 函数中，应对 `dlnetwork` 使用 `predict` 函数。使用 `dlnetwork` 对象的 `predict` 函数可确保软件使用正确的层运算进行预测。

`forward` 的语法是 `[Z1,...,Zm,mem] = forward(layer,X1,...,Xn)`，其中 `X1,...,Xn` 是 `n` 个层输入，`Z1,...,Zm` 是 `m` 个层输出，`mem` 是层的内存。

**提示** 如果 `forward` 的输入的数目可能变化，则使用 `varargin` 代替 `X1,...,Xn`。在本例中，`varargin` 是由输入组成的元胞数组，其中 `varargin{i}` 对应于 `Xi`。如果输出的数目可能变化，则使用 `varargout` 代替 `Z1,...,Zm`。在本例中，`varargout` 是由输出组成的元胞数组，其中 `varargout{j}` 对应于 `Zj` (`j = 1, ..., NumOutputs`)，`varargout{NumOutputs + 1}` 对应于 `mem`。

**提示** 如果自定义层具有一个作为可学习参数的 `dlnetwork` 对象，则在自定义层的 `forward` 函数中，应对 `dlnetwork` 对象使用 `forward` 函数。使用 `dlnetwork` 对象的 `forward` 函数可确保软件使用正确的层运算进行训练。

输入的维度取决于数据的类型和所连接层的输出：

层输入	输入大小	观测值维度
二维图像	$h \times w \times c \times N$ ，其中 $h$ 、 $w$ 和 $c$ 分别对应于图像的高度、宽度和通道数， $N$ 是观测值数目。	4

层输入	输入大小	观测值维度
三维图像	$h \times w \times d \times c \times N$ , 其中 $h$ 、 $w$ 、 $d$ 和 $c$ 分别对应于三维图像的高度、宽度、深度和通道数, $N$ 是观测值数目。	5
向量序列	$c \times N \times S$ , 其中 $c$ 是序列的特征数, $N$ 是观测值数目, $S$ 是序列长度。	2
二维图像序列	$h \times w \times c \times N \times S$ , 其中 $h$ 、 $w$ 和 $c$ 分别对应于图像的高度、宽度和通道数, $N$ 是观测值数目, $S$ 是序列长度。	4
三维图像序列	$h \times w \times d \times c \times N \times S$ , 其中 $h$ 、 $w$ 、 $d$ 和 $c$ 分别对应于三维图像的高度、宽度、深度和通道数量, $N$ 是观测值数目, $S$ 是序列长度。	5

对于输出序列的层，层可以输出任意长度的序列或输出无时间维度的数据。请注意，当使用 `trainNetwork` 函数训练输出序列的网络时，输入和输出序列的长度必须匹配。

## 后向函数

层后向函数计算损失关于输入数据的导数，然后将结果输出（后向传播）到前一层。如果层具有可学习参数（例如，层权重），则 `backward` 还计算可学习参数的导数。当使用 `trainNetwork` 函数时，层会在后向传导过程中使用这些导数自动更新可学习参数。

定义后向函数是可选的。如果未指定后向函数，并且层前向函数支持 `dlarray` 对象，则软件会使用自动微分自动确定后向函数。有关支持 `dlarray` 对象的函数列表，请参阅“List of Functions with dlarray Support”。当您需要执行以下操作时，请定义一个自定义后向函数：

- 使用特定算法来计算导数。
- 在前向函数中使用不支持 `dlarray` 对象的操作。

具有可学习 `dlnetwork` 对象的自定义层不支持自定义后向函数。

要定义一个自定义后向函数，请创建名为 `backward` 的函数。

`backward` 的语法是 `[dLdX1,...,dLdXn,dLdW1,...,dLdWk] = backward(layer,X1,...,Xn,Z1,...,Zm,dLdZ1,...,dLdZm,memory)`，其中：

- $X_1, \dots, X_n$  是  $n$  个层输入
- $Z_1, \dots, Z_m$  是层前向函数的  $m$  个输出
- $dLdZ_1, \dots, dLdZ_m$  是从后一层反向传播的梯度
- 如果定义了 `forward`，则 `memory` 是 `forward` 的内存输出，否则 `memory` 是 `[]`。

对于输出， $dLdX_1, \dots, dLdX_n$  是损失关于层输入的导数， $dLdW_1, \dots, dLdW_k$  是损失关于  $k$  个可学习参数的导数。要通过防止在前向和后向传导之间保存未使用的变量来减少内存使用量，请用 `~` 替换对应的输入参数。

---

**提示** 如果 `backward` 的输入数目可能变化，则使用 `varargin` 代替 `layer` 后的输入参数。在本例中，`varargin` 是由输入组成的元胞数组，其中 `varargin{i}` 对应于  $X_i$  ( $i = 1, \dots, \text{NumInputs}$ )，

`varargin{NumInputs+j}` 和 `varargin{NumInputs+NumOutputs+j}` 分别对应于  $Z_j$  和  $dLdZ_j$  ( $j=1, \dots, \text{NumOutputs}$ )，`varargin{end}` 对应于 `memory`。

如果输出的数目可能变化，则使用 `varargout` 代替输出参数。在本例中，`varargout` 是由输出组成的元胞数组，其中 `varargout{i}` 对应于  $dLdX_i$  ( $i = 1, \dots, \text{NumInputs}$ )，`varargout{NumInputs+t}` 对应于  $dLdW_t$  ( $t = 1, \dots, k$ )，其中  $k$  是可学习参数的数目。

$X_1, \dots, X_n$  和  $Z_1, \dots, Z_m$  的值与前向函数中的值相同。 $dLdZ_1, \dots, dLdZ_m$  的维度分别与  $Z_1, \dots, Z_m$  的维度相同。

$dLdX_1, \dots, dLdX_n$  的维度和数据类型分别与  $X_1, \dots, X_n$  的维度和数据类型相同。 $dLdW_1, \dots, dLdW_k$  的维度和数据类型分别与  $W_1, \dots, W_k$  的维度和数据类型相同。

要计算损失的导数，您可以使用链式法则：

$$\frac{\partial L}{\partial X^{(i)}} = \sum_j \frac{\partial L}{\partial Z_j} \frac{\partial Z_j}{\partial X^{(i)}}$$

$$\frac{\partial L}{\partial W_i} = \sum_j \frac{\partial L}{\partial Z_j} \frac{\partial Z_j}{\partial W_i}$$

当使用 `trainNetwork` 函数时，层会在后向传导过程中使用导数  $dLdW_1, \dots, dLdW_k$  自动更新可学习参数。

有关如何定义自定义后向函数的示例，请参阅“Specify Custom Layer Backward Function”。

## GPU 兼容性

如果层前向函数完全支持 `dlarray` 对象，则该层与 GPU 兼容。否则，为了与 GPU 兼容，层函数必须支持 `gpuArray` 类型的输入并返回其输出。

许多 MATLAB 内置函数支持 `gpuArray` 和 `dlarray` 输入参数。有关支持 `dlarray` 对象的函数列表，请参阅“List of Functions with dlarray Support”。有关在 GPU 上执行的函数的列表，请参阅“Run MATLAB Functions on a GPU”（Parallel Computing Toolbox）。要使用 GPU 进行深度学习，您还必须拥有支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。有关在 MATLAB 中使用 GPU 的详细信息，请参阅“GPU Computing in MATLAB”（Parallel Computing Toolbox）。

## 代码生成兼容性

要创建支持代码生成的自定义层，需要满足以下要求：

- 层必须在层定义中指定 pragma 指令 `%#codegen`。
- `predict` 的输入必须：
  - 维度一致。每个输入必须具有相同的维数。
  - 批量大小一致。每个输入的批量大小必须相同。
- `predict` 的输出必须在维度和批量大小上与层输入一致。
- 非标量属性的类型必须为单精度、双精度或字符数组。
- 标量属性的类型必须为数值、逻辑值或字符串。

代码生成仅支持以二维图像作为输入的中间层。

有关如何创建支持代码生成的自定义层的示例，请参阅“Define Custom Deep Learning Layer for Code Generation”。

## 网络合成

要创建一个本身定义了层次图的自定义层，您可以将 `dlnetwork` 对象指定为可学习参数。这种方法称为网络合成。您可以使用网络合成来实现以下目的：

- 创建一个自定义层，该层表示一个由可学习层组成的模块，例如，一个残差模块。
- 创建一个具有控制流的网络，例如，网络的一部分可以根据输入数据而动态变化。
- 创建一个具有循环的网络，例如，网络的某些部分将输出反馈到自身。

## 检查层的有效性

如果您创建自定义深度学习层，则可以使用 `checkLayer` 函数来检查该层是否有效。该函数检查层的有效性、GPU 兼容性、正确定义的梯度和代码生成兼容性。要检查层是否有效，请运行以下命令：

```
checkLayer(layer,validInputSize,'ObservationDimension',dim)
```

其中，`layer` 是层的实例，`validInputSize` 是用于指定层的有效输入大小的向量或元胞数组，`dim` 指定层输入数据中观测值的维度。对于较大的输入大小，梯度检查的运行时间较长。为了加快测试速度，请指定较小的有效输入大小。

有关详细信息，请参阅“Check Custom Layer Validity”。

### 使用 `checkLayer` 检查层的有效性

检查自定义层 `preluLayer` 的层有效性。

定义自定义 PReLU 层。要创建此层，请将文件 `preluLayer.m` 保存在当前文件夹中。

创建一个层实例，并使用 `checkLayer` 检查其有效性。将有效输入大小指定为层的典型输入的单个观测值的大小。该层需要四维数组输入，其中前三个维度对应于前一个层输出的高度、宽度和通道数，第四个维度对应于观测值。

指定观测值的输入的典型大小，并将 '`ObservationDimension`' 设置为 4。

```
layer = preluLayer(20,'prelu');
validInputSize = [24 24 20];
checkLayer(layer,validInputSize,'ObservationDimension',4)
```

Skipping GPU tests. No compatible GPU device found.

Skipping code generation compatibility tests. To check validity of the layer for code generation, specify the 'CheckCodegen' argument.

Running nnet.checklayer.TestLayerWithoutBackward

.....

Done nnet.checklayer.TestLayerWithoutBackward

---

#### Test Summary:

13 Passed, 0 Failed, 0 Incomplete, 9 Skipped.

Time elapsed: 0.18046 seconds.

在此处，该函数不检测该层的任何问题。

## 在网络中包含层

您可以像使用 Deep Learning Toolbox 中的任何其他层一样使用自定义层。

定义自定义 PReLU 层。要创建此层，请将文件 `preluLayer.m` 保存在当前文件夹中。

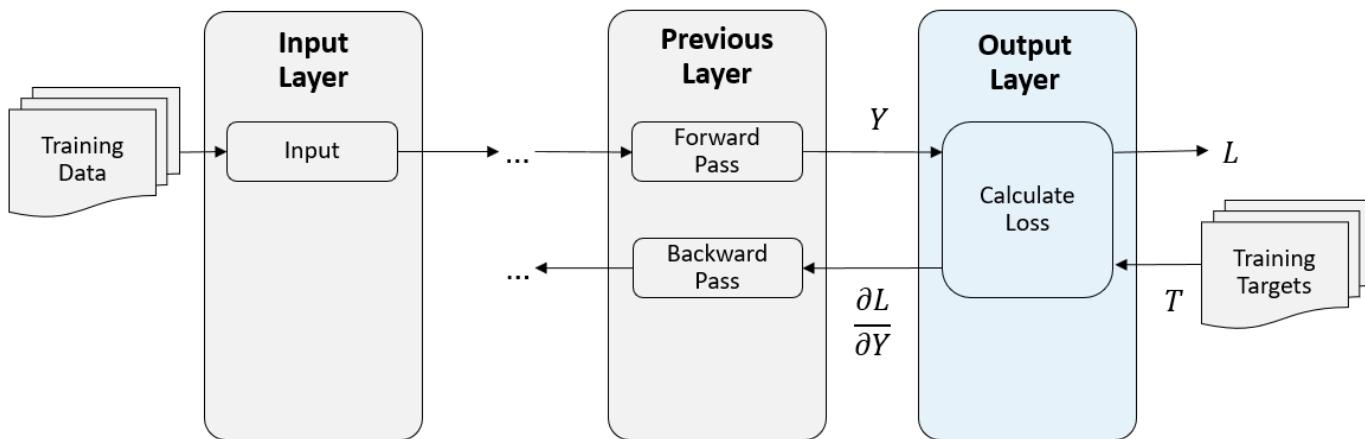
创建一个包含自定义层 `preluLayer` 的层数组。

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    batchNormalizationLayer
    preluLayer(20,'prelu')
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];
```

## 输出层架构

在训练时的前向传导结束时，输出层获取前一个层的预测（输出） $y$ ，并计算这些预测和训练目标之间的损失  $L$ 。输出层计算损失  $L$  关于预测  $y$  的导数，并将结果输出（后向传播）到前一个层。

下图说明通过卷积神经网络和输出层的数据流。



### 输出层属性

在类定义的 `properties` 部分声明层属性。

默认情况下，自定义输出层具有以下属性：

- **Name** - 层名称，指定为字符向量或字符串标量。要将一个层包括在层次图中，您必须指定非空的唯一层名称。如果您使用该层训练串行网络并且 `Name` 设置为 "，则软件会在训练时自动为该层指定名称。
- **Description** - 层的单行描述，指定为字符向量或字符串标量。当层显示在 `Layer` 数组中时，会出现此描述。如果未指定层描述，则软件将显示 "Classification Output" 或 "Regression Output"。
- **Type** - 层的类型，指定为字符向量或字符串标量。当层显示在 `Layer` 数组中时，会显示 `Type` 的值。如果没有指定层类型，则软件将显示层类名。

自定义分类层还具有以下属性：

- **Classes** - 输出层的类，指定为分类向量、字符串数组、字符向量元胞数组或 'auto'。如果 **Classes** 是 'auto'，则软件会在训练时自动设置类。如果您指定字符串数组或字符向量元胞数组 **str**，则软件会将输出层的类设置为 **categorical(str,str)**。

自定义回归层还具有以下属性：

- **ResponseNames** - 响应的名称，指定为字符向量元胞数组或字符串数组。在训练时，软件根据训练数据自动设置响应名称。默认值为 {}。

如果层没有其他属性，则您可以省略 **properties** 部分。

### 损失函数

输出层使用前向损失函数计算预测和目标之间的损失 **L**，并使用后向损失函数计算损失关于预测的导数。

**forwardLoss** 的语法是 **loss = forwardLoss(layer, Y, T)**。输入 **Y** 对应于网络做出的预测。这些预测是前一个层的输出。输入 **T** 对应于训练目标。根据指定的损失函数，输出 **loss** 是 **Y** 和 **T** 之间的损失。输出 **loss** 必须为标量。

如果层前向损失函数支持 **dlarray** 对象，则软件会自动确定后向损失函数。有关支持 **dlarray** 对象的函数列表，请参阅“[List of Functions with dlarray Support](#)”。或者，要定义自定义后向损失函数，请创建一个名为 **backwardLoss** 的函数。有关如何定义自定义后向损失函数的示例，请参阅“[Specify Custom Output Layer Backward Loss Function](#)”。

**backwardLoss** 的语法是 **dLdY = backwardLoss(layer, Y, T)**。输入 **Y** 包含网络做出的预测，**T** 包含训练目标。输出 **dLdY** 是损失关于预测 **Y** 的导数。输出 **dLdY** 的大小必须与层输入 **Y** 的大小相同。

对于分类问题，**T** 的维度取决于问题的类型。

分类任务	输入大小	观测值维度
二维图像分类	$1 \times 1 \times K \times N$ ，其中 $K$ 是类数， $N$ 是观测值数目。	4
三维图像分类	$1 \times 1 \times 1 \times K \times N$ ，其中 $K$ 是类数， $N$ 是观测值数目。	5
“序列到标签”分类	$K \times N$ ，其中 $K$ 是类数， $N$ 是观测值数目。	2
“序列到序列”分类	$K \times N \times S$ ，其中 $K$ 是类数， $N$ 是观测值数目， $S$ 是序列长度。	2

**Y** 的大小取决于前一个层的输出。为了确保 **Y** 与 **T** 的大小相同，您必须在输出层之前包含一个输出正确大小的层。例如，为了确保 **Y** 是  $K$  个类的预测分数的四维数组，您可以包含一个大小为  $K$  的全连接层，后跟一个 **softmax** 层，然后是输出层。

对于回归问题，**T** 的维度还取决于问题的类型。

回归任务	输入大小	观测值维度
二维图像回归	$1 \times 1 \times R \times N$ ，其中 $R$ 是响应数， $N$ 是观测值数目。	4

回归任务	输入大小	观测值维度
二维图像到图像回归	$h \times w \times c \times N$ , 其中 $h$ 、 $w$ 和 $c$ 分别是输出的高度、宽度和通道数, $N$ 是观测值数目。	4
三维图像回归	$1 \times 1 \times 1 \times R \times N$ , 其中 $R$ 是响应数, $N$ 是观测值数目。	5
三维图像到图像回归	$h \times w \times d \times c \times N$ , 其中 $h$ 、 $w$ 、 $d$ 和 $c$ 分别是输出的高度、宽度、深度和通道数, $N$ 是观测值数目。	5
“序列到单个” 回归	$R \times N$ , 其中 $R$ 是响应数, $N$ 是观测值数目。	2
“序列到序列” 回归	$R \times N \times S$ , 其中 $R$ 是响应数, $N$ 是观测值数目, $S$ 是序列长度。	2

例如, 如果网络定义具有一个响应的图像回归网络, 并且小批量大小为 50, 则  $T$  是大小为  $1 \times 1 \times 1 \times 50$  的四维数组。

$Y$  的大小取决于前一个层的输出。为了确保  $Y$  与  $T$  的大小相同, 您必须在输出层之前包含一个输出正确大小的层。例如, 对于具有  $R$  个响应的图像回归, 为了确保  $Y$  是具有正确大小的四维数组, 您可以在输出层之前包含一个大小为  $R$  的全连接层。

**forwardLoss** 和 **backwardLoss** 函数具有以下输出参数。

函数	输出参数	说明
<b>forwardLoss</b>	<b>loss</b>	预测 $Y$ 和真实目标 $T$ 之间的计算损失。
<b>backwardLoss</b>	<b>dLdY</b>	损失关于预测 $Y$ 的导数。

**backwardLoss** 必须输出  $dLdY$ , 其大小为前一个层预期的大小, 并且  $dLdY$  与  $Y$  大小相同。

## GPU 兼容性

如果层前向函数完全支持 **dlarray** 对象, 则该层与 GPU 兼容。否则, 为了与 GPU 兼容, 层函数必须支持 **gpuArray** 类型的输入并返回其输出。

许多 MATLAB 内置函数支持 **gpuArray** 和 **dlarray** 输入参数。有关支持 **dlarray** 对象的函数列表, 请参阅 “List of Functions with dlarray Support”。有关在 GPU 上执行的函数的列表, 请参阅 “Run MATLAB Functions on a GPU” (Parallel Computing Toolbox)。要使用 GPU 进行深度学习, 您还必须拥有支持的 GPU 设备。有关受支持设备的信息, 请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。有关在 MATLAB 中使用 GPU 的详细信息, 请参阅 “GPU Computing in MATLAB” (Parallel Computing Toolbox)。

## 在网络中包含自定义回归输出层

您可以像使用 Deep Learning Toolbox 中的任何其他输出层一样使用自定义输出层。本节说明如何使用自定义输出层创建和训练网络以用于回归任务。

该示例构造一个卷积神经网络架构, 训练网络, 并使用经过训练的网络预测手写数字的旋转角度。这些预测对于光学字符识别很有用。

定义一个自定义均值绝对误差回归层。要创建此层，请将文件 `maeRegressionLayer.m` 保存在当前文件夹中。

加载示例训练数据。

```
[XTrain,~,YTrain] = digitTrain4DArrayData;
```

创建一个层数组，并包含自定义回归输出层 `maeRegressionLayer`。

```
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(5,20)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(1)
    maeRegressionLayer('mae')]
```

```
layers =
6x1 Layer array with layers:
```

```
1 " Image Input      28x28x1 images with 'zerocenter' normalization
2 " Convolution      20 5x5 convolutions with stride [1 1] and padding [0 0 0 0]
3 " Batch Normalization  Batch normalization
4 " ReLU              ReLU
5 " Fully Connected   1 fully connected layer
6 'mae' Regression Output  Mean absolute error
```

设置训练选项并训练网络。

```
options = trainingOptions('sgdm','Verbose',false);
net = trainNetwork(XTrain,YTrain,layers,options);
```

通过计算预测旋转角度和实际旋转角度之间的预测误差来评估网络性能。

```
[XTest,~,YTest] = digitTest4DArrayData;
YPred = predict(net,XTest);
predictionError = YTest - YPred;
```

计算在实际角度的可接受误差界限内的预测值的数量。将阈值设置为 10 度，并计算此阈值范围内的预测百分比。

```
thr = 10;
numCorrect = sum(abs(predictionError) < thr);
numTestImages = size(XTest,4);
accuracy = numCorrect/numTestImages
```

```
accuracy = 0.7524
```

## 另请参阅

`setLearnRateFactor` | `checkLayer` | `setL2Factor` | `getLearnRateFactor` | `getL2Factor` | `assembleNetwork`

## 详细信息

- “Define Custom Deep Learning Layer with Learnable Parameters”

- “Define Custom Deep Learning Layer with Multiple Inputs”
- “Define Nested Deep Learning Layer”
- “Define Custom Deep Learning Layer for Code Generation”
- “Define Custom Classification Output Layer”
- “Define Custom Regression Output Layer”
- “Specify Custom Layer Backward Function”
- “Specify Custom Output Layer Backward Loss Function”
- “Check Custom Layer Validity”
- “深度学习层列表” (第 1-18 页)
- “Deep Learning Tips and Tricks”

## 指定自定义权重初始化函数

此示例说明如何为后跟泄漏 ReLU 层的卷积层创建自定义 He 权重初始化函数。

这个 He 初始化函数从均值为零、方差为  $\sigma^2 = \frac{2}{(1 + a^2)n}$  的正态分布进行采样，其中  $a$  是卷积层后的泄漏 ReLU 层的缩放因子， $n = \text{FilterSize}(1) * \text{FilterSize}(2) * \text{NumChannels}$ 。

对于可学习层，当选项 'WeightsInitializer'、'InputWeightsInitializer' 或 'RecurrentWeightsInitializer' 设置为 'he' 时，软件使用  $a=0$ 。要将  $a$  设置为不同值，请创建一个自定义函数以用作权重初始化函数。

### 加载数据

将数字样本数据加载为图像数据存储。imageDatastore 函数根据文件夹名称自动对图像加标签。

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos',...
    'nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

将数据划分为训练数据集和验证数据集，以使训练集中的每个类别包含 750 个图像，并且验证集包含对应每个标签的其余图像。splitEachLabel 将数据存储拆分为两个新的数据存储以用于训练和验证。

```
numTrainFiles = 750;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

### 定义网络架构

定义卷积神经网络架构：

- 大小为 [28 28 1] 的图像输入层，输入图像的大小
- 三个二维卷积层，滤波器大小为 3 并分别具有 8、16 和 32 个滤波器
- 每个卷积层后有一个泄漏 ReLU 层
- 大小为 10 的全连接层，类的数量
- Softmax 层
- 分类层

对于每个卷积层，将权重初始化函数设置为 leakyHe 函数。在本示例末尾列出的 leakyHe 函数接受输入 sz（层权重的大小），并返回后跟泄漏 ReLU 层的卷积层的 He 初始化函数给出的权重数组。

```
inputSize = [28 28 1];
numClasses = 10;

layers = [
    imageInputLayer(inputSize)
    convolution2dLayer(3,8,'WeightsInitializer',@leakyHe)
    leakyReluLayer
    convolution2dLayer(3,16,'WeightsInitializer',@leakyHe)
    leakyReluLayer
    convolution2dLayer(3,32,'WeightsInitializer',@leakyHe)
    leakyReluLayer
    fullyConnectedLayer(numClasses)
```

```
softmaxLayer
classificationLayer];
```

## 训练网络

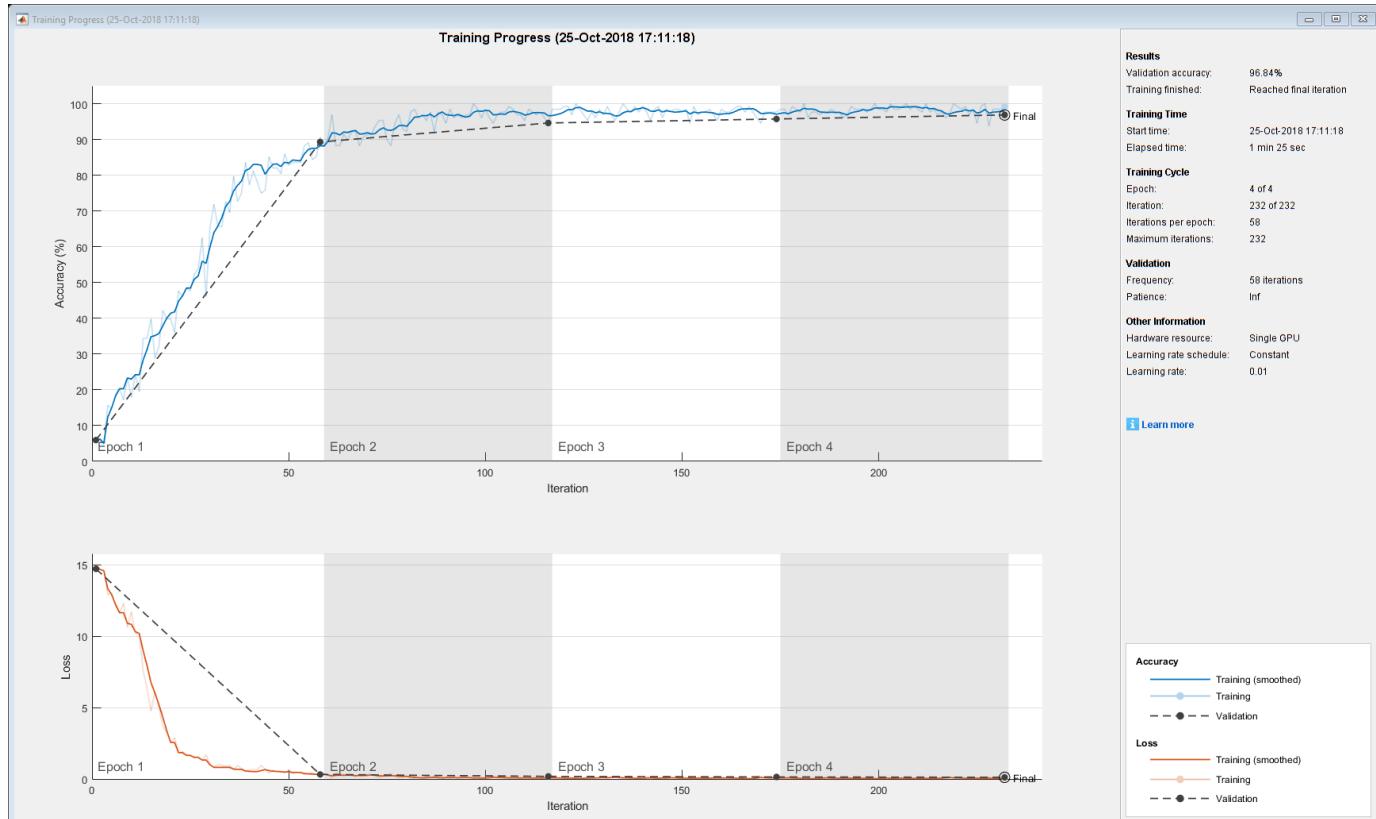
指定训练选项并训练网络。进行四轮训练。要防止梯度爆炸，请将梯度阈值设置为 2。每轮训练后对网络进行一次验证。查看训练进度图。

默认情况下，`trainNetwork` 使用 GPU（如果有），否则使用 CPU。在 GPU 上训练需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。您还可以使用 `trainingOptions` 的‘ExecutionEnvironment’名称-值对组参数指定执行环境。

```
maxEpochs = 4;
miniBatchSize = 128;
numObservations = numel(imdsTrain.Files);
numIterationsPerEpoch = floor(numObservations / miniBatchSize);

options = trainingOptions('sgdm', ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',2, ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',numIterationsPerEpoch, ...
    'Verbose',false, ...
    'Plots','training-progress');
```

```
[netDefault,infoDefault] = trainNetwork(imdsTrain,layers,options);
```



## 测试网络

对验证数据进行分类，并计算分类准确度。

```
YPred = classify(netDefault,imdsValidation);
YValidation = imdsValidation.Labels;
accuracy = mean(YPred == YValidation)

accuracy = 0.9684
```

## 指定附加选项

leakyHe 函数接受可选输入参数 scale。要将额外的变量输入到自定义权重初始化函数中，请将该函数指定为接受单个输入 sz 的匿名函数。为此，请用 @(sz) leakyHe(sz,scale) 替换 @leakyHe 的实例。此处，匿名函数只接受单个输入参数 sz，并使用指定的 scale 输入参数调用 leakyHe 函数。

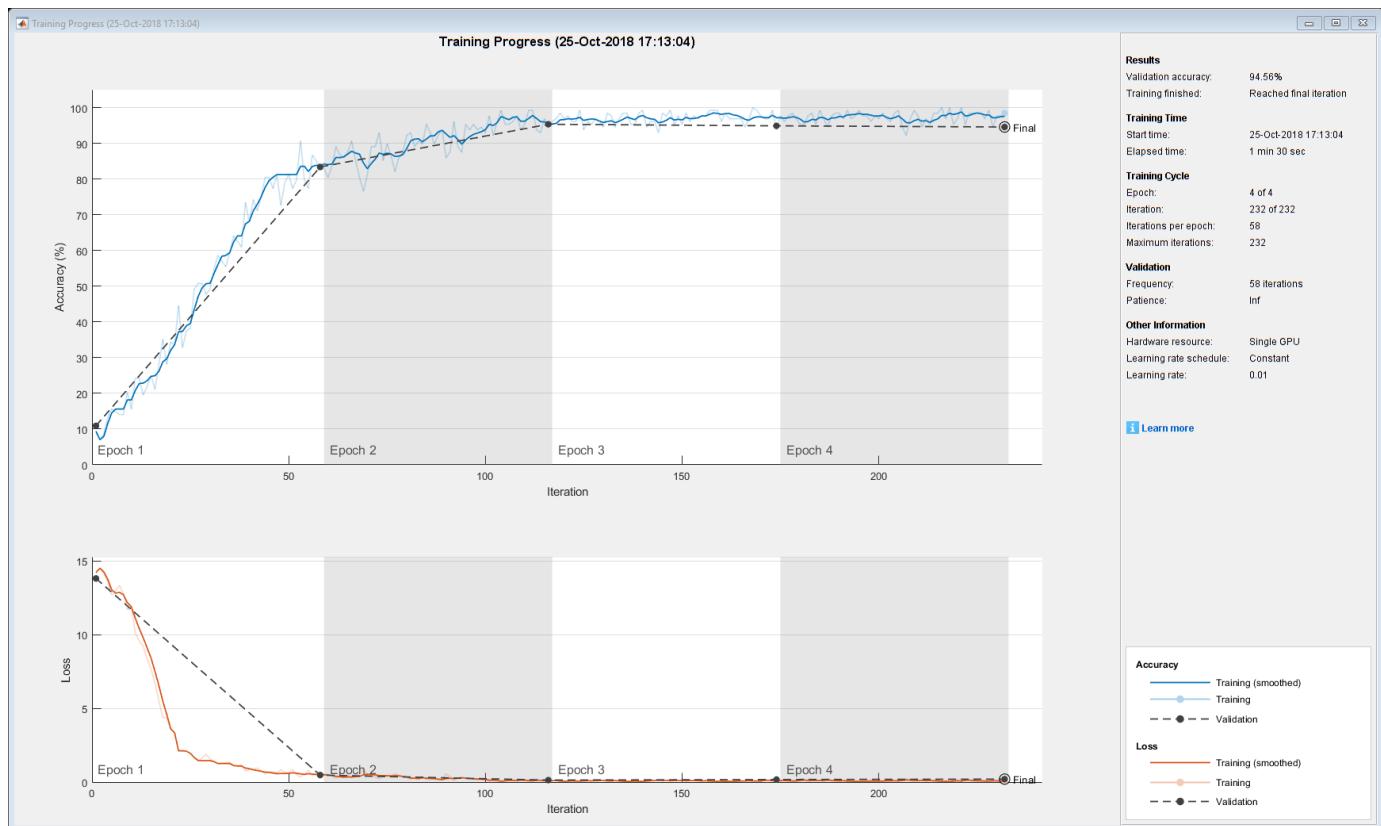
对网络进行以下更改，然后沿用前面的方法训练网络：

- 对于泄漏 ReLU 层，指定 0.01 的尺度乘数。
- 使用 leakyHe 函数初始化卷积层的权重，并指定尺度乘数。

```
scale = 0.01;
```

```
layers = [
    imageInputLayer(inputSize)
    convolution2dLayer(3,8,'WeightsInitializer',@(sz) leakyHe(sz,scale))
    leakyReluLayer(scale)
    convolution2dLayer(3,16,'WeightsInitializer',@(sz) leakyHe(sz,scale))
    leakyReluLayer(scale)
    convolution2dLayer(3,32,'WeightsInitializer',@(sz) leakyHe(sz,scale))
    leakyReluLayer(scale)
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

```
[netCustom,infoCustom] = trainNetwork(imdsTrain,layers,options);
```



对验证数据进行分类，并计算分类准确度。

```
YPred = classify(netCustom,imdsValidation);
YValidation = imdsValidation.Labels;
accuracy = mean(YPred == YValidation)
```

accuracy = 0.9456

## 比较结果

从 `trainNetwork` 函数的信息结构体输出中提取验证准确度。

```
validationAccuracy =
    infoDefault.ValidationAccuracy;
    infoCustom.ValidationAccuracy];
```

对于未计算验证准确度的迭代，验证准确度向量包含 NaN。删除 NaN 值。

```
idx = all(isnan(validationAccuracy));
validationAccuracy(:,idx) = [];
```

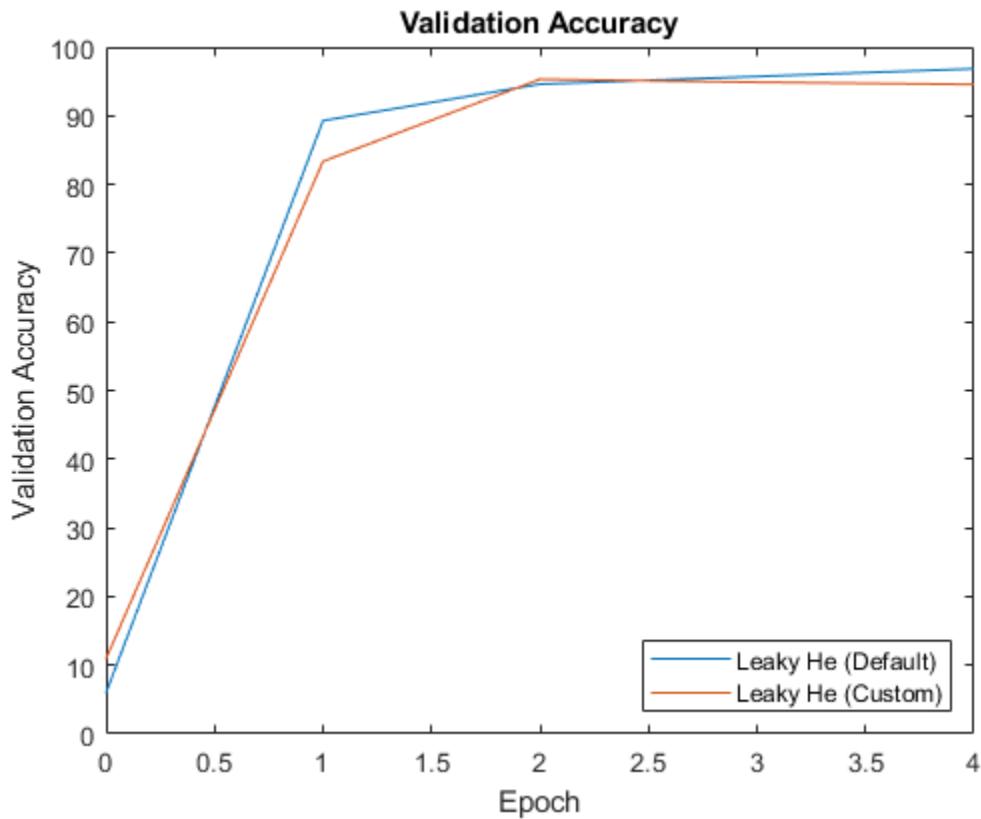
对于每个网络，绘制轮次编号对验证准确度的图。

```
figure
epochs = 0:maxEpochs;
plot(epochs,validationAccuracy)
title("Validation Accuracy")
xlabel("Epoch")
```

```

ylabel("Validation Accuracy")
legend(["Leaky He (Default)" "Leaky He (Custom)"],'Location','southeast')

```



### 自定义权重初始化函数

leakyHe 函数接受输入 sz (层权重的大小) , 并返回后跟泄漏 ReLU 层的卷积层的 He 初始化函数给出的权重数组。该函数还接受可选输入参数 scale, 该参数指定泄漏 ReLU 层的尺度乘数。

```

function weights = leakyHe(sz,scale)

% If not specified, then use default scale = 0.1
if nargin < 2
    scale = 0.1;
end

filterSize = [sz(1) sz(2)];
numChannels = sz(3);
numIn = filterSize(1) * filterSize(2) * numChannels;

varWeights = 2 / ((1 + scale^2) * numIn);
weights = randn(sz) * sqrt(varWeights);

end

```

## 参考书目

- 1 He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun."Delving deep into rectifiers:Surpassing human-level performance on imagenet classification."In Proceedings of the IEEE international conference on computer vision, pp. 1026-1034. 2015.

## 另请参阅

[trainNetwork](#) | [trainingOptions](#)

## 相关示例

- “比较层权重初始化函数” (第 17-22 页)
- “深度学习层列表” (第 1-18 页)
- “Deep Learning Tips and Tricks”
- “在 MATLAB 中进行深度学习” (第 1-2 页)

## 比较层权重初始化函数

此示例说明如何使用不同权重初始化函数来训练深度学习网络。

在训练深度学习网络时，层权重和偏置的初始化会对网络训练的效果产生重大影响。初始化函数的选择对没有批量归一化层的网络的影响更大。

根据层的类型，您可以使用 '**WeightsInitializer**'、'**InputWeightsInitializer**'、'**RecurrentWeightsInitializer**' 和 '**BiasInitializer**' 选项更改权重和偏置初始化。

此示例说明在训练 LSTM 网络时使用以下三种不同权重初始化函数的效果：

- 1 **Glorot 初始化函数** - 用 Glorot 初始化函数初始化输入权重。[1]
- 2 **He 初始化函数** - 用 He 初始化函数初始化输入权重。[2]
- 3 **窄正态初始化函数** - 通过从零均值和标准差为 0.01 的正态分布中独立采样来初始化输入权重。

### 加载数据

加载日语元音数据集。**XTrain** 是包含 270 个不同长度序列的元胞数组，特征维数为 12。**Y** 是标签 1、2、...、9 的分类向量。**XTrain** 中的条目是具有 12 行（每个特征一行）和不同列数（每个时间步一列）的矩阵。

```
[XTrain,YTrain] = japaneseVowelsTrainData;
[XValidation,YValidation] = japaneseVowelsTestData;
```

### 指定网络架构

指定网络架构。对于每个初始化函数，使用相同的网络架构。

将输入大小指定为 12（输入数据的特征数量）。指定具有 100 个隐含单元的 LSTM 层，并输出序列的最后一个元素。最后，通过包含大小为 9 的全连接层，后跟 softmax 层和分类层，来指定九个类。

```
numFeatures = 12;
numHiddenUnits = 100;
numClasses = 9;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer]

layers =
5x1 Layer array with layers:

1 " Sequence Input      Sequence input with 12 dimensions
2 " LSTM                 LSTM with 100 hidden units
3 " Fully Connected     9 fully connected layer
4 " Softmax              softmax
5 " Classification Output crossentropyex
```

### 训练选项

指定训练选项。对于每个初始化函数，使用相同的训练选项来训练网络。

```

maxEpochs = 30;
miniBatchSize = 27;
numObservations = numel(XTrain);
numIterationsPerEpoch = floor(numObservations / miniBatchSize);

options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',2, ...
    'ValidationData',{XValidation,YValidation}, ...
    'ValidationFrequency',numIterationsPerEpoch, ...
    'Verbose',false, ...
    'Plots','training-progress');

```

### Glorot 初始化函数

指定在示例前面部分列出的网络架构，并将 LSTM 层的输入权重初始化函数和全连接层的权重初始化函数设置为 'glorot'。

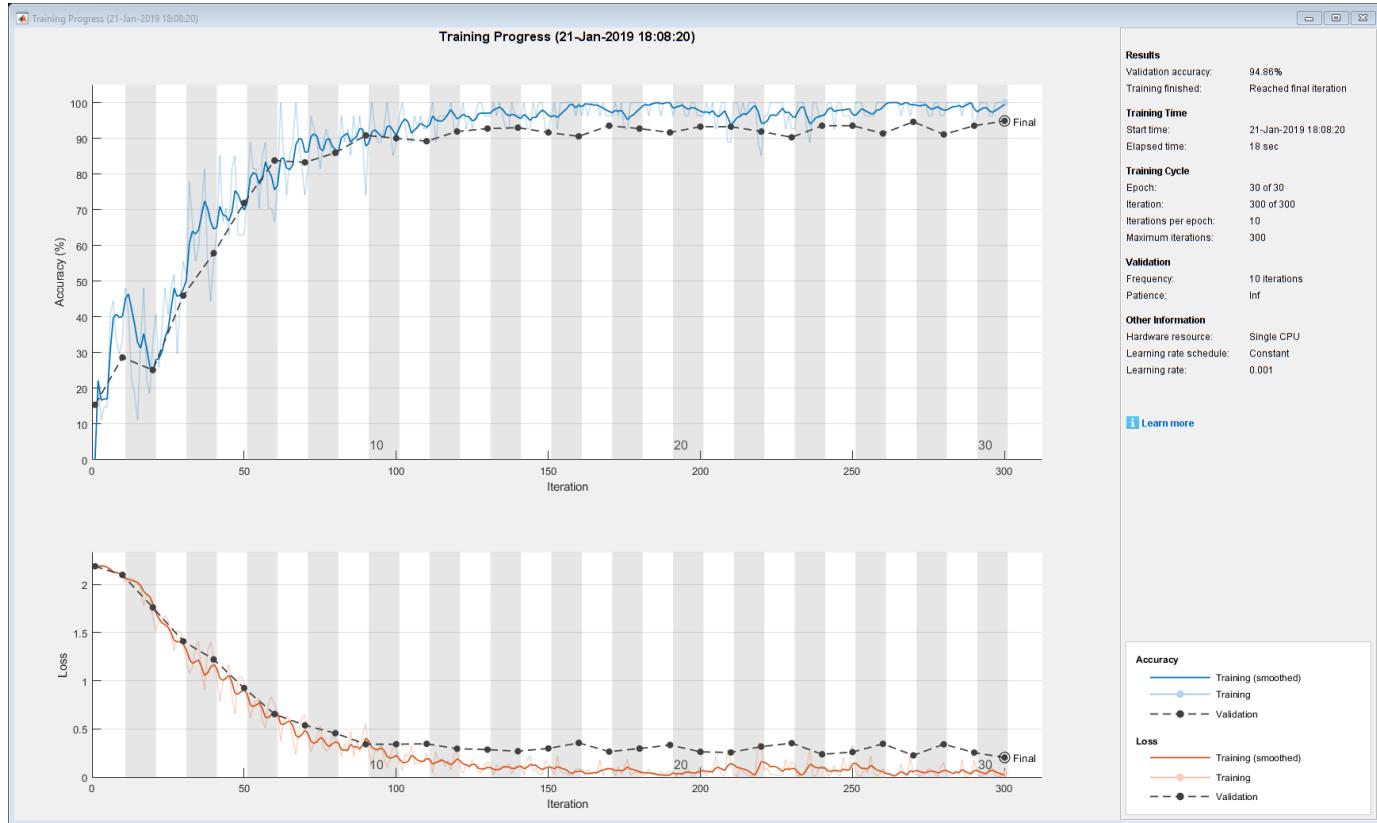
```

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last','InputWeightsInitializer','glorot')
    fullyConnectedLayer(numClasses,'WeightsInitializer','glorot')
    softmaxLayer
    classificationLayer];

```

使用具有 Glorot 权重初始化函数的层来训练网络。

```
[netGlorot,infoGlorot] = trainNetwork(XTrain,YTrain,layers,options);
```

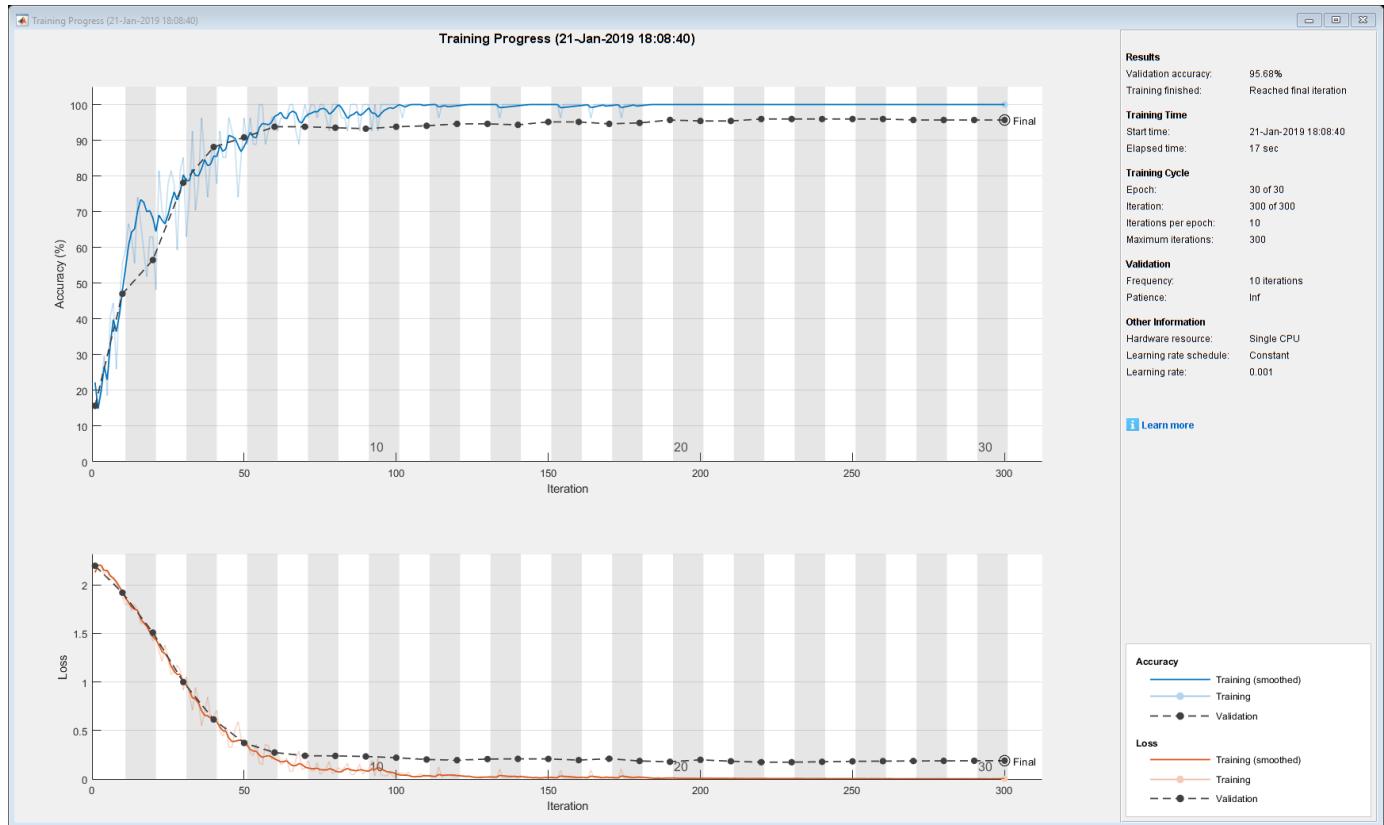


## He 初始化函数

```
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits, 'OutputMode', 'last', 'InputWeightsInitializer', 'he')
    fullyConnectedLayer(numClasses, 'WeightsInitializer', 'he')
    softmaxLayer
    classificationLayer];
```

使用具有 He 权重初始化函数的层来训练网络。

```
[netHe,infoHe] = trainNetwork(XTrain,YTrain,layers,options);
```



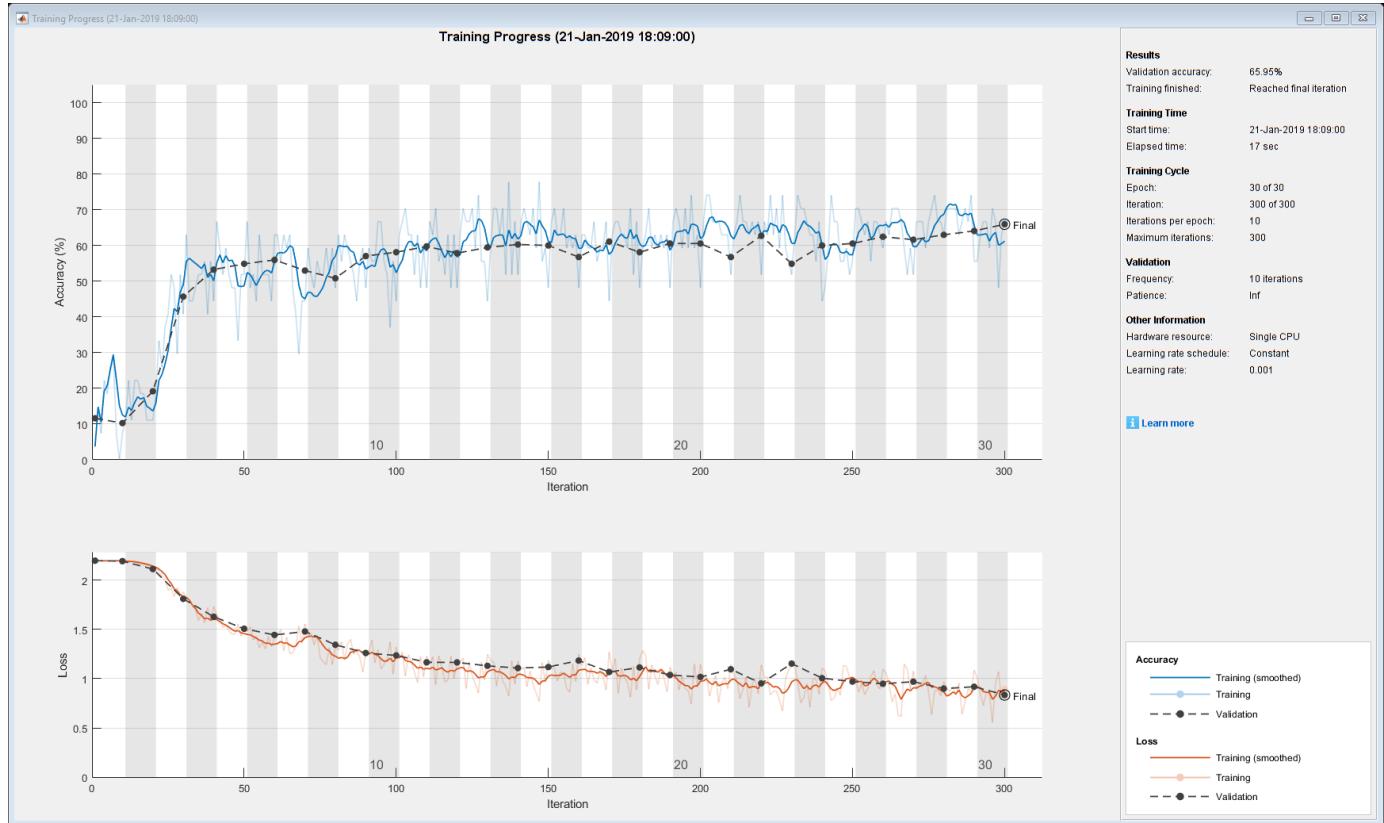
## 窄正态初始化函数

指定在示例前面部分列出的网络架构，并将 LSTM 层的输入权重初始化函数和全连接层的权重初始化函数设置为 'narrow-normal'。

```
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last','InputWeightsInitializer','narrow-normal')
    fullyConnectedLayer(numClasses,'WeightsInitializer','narrow-normal')
    softmaxLayer
    classificationLayer];
```

使用具有窄正态权重初始化函数的层来训练网络。

```
[netNarrowNormal,infoNarrowNormal] = trainNetwork(XTrain,YTrain,layers,options);
```



## 绘制结果

从 `trainNetwork` 函数的信息结构体输出中提取验证准确度。

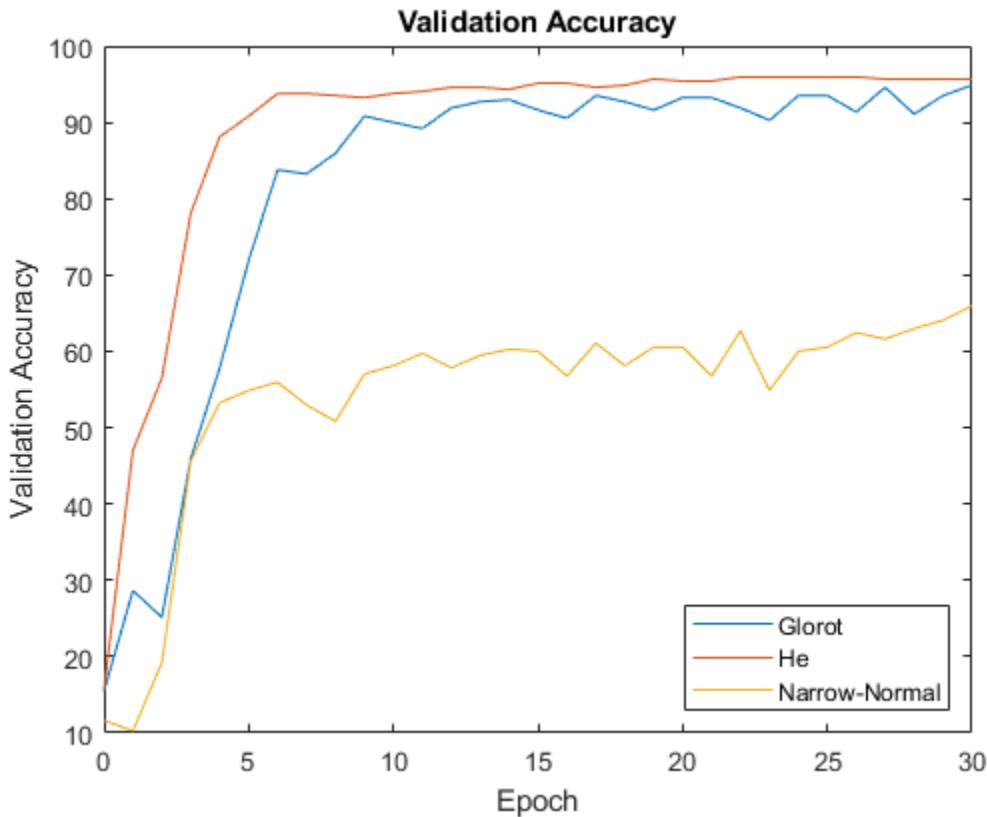
```
validationAccuracy = [
    infoGlorot.ValidationAccuracy;
    infoHe.ValidationAccuracy;
    infoNarrowNormal.ValidationAccuracy];
```

对于未计算验证准确度的迭代，验证准确度向量包含 NaN。删除 NaN 值。

```
idx = all(isnan(validationAccuracy));
validationAccuracy(:,idx) = [];
```

对于每个初始化函数，绘制轮次编号对验证准确度的图。

```
figure
epochs = 0:maxEpochs;
plot(epochs,validationAccuracy)
title("Validation Accuracy")
xlabel("Epoch")
ylabel("Validation Accuracy")
legend(["Glorot" "He" "Narrow-Normal"],'Location','southeast')
```



该图显示了不同初始化函数的整体效果，以及每个初始化函数的训练收敛速度。

## 参考书目

- 1 Glorot, Xavier, and Yoshua Bengio."Understanding the difficulty of training deep feedforward neural networks."In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249-256. 2010.
- 2 He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun."Delving deep into rectifiers:Surpassing human-level performance on imagenet classification."In Proceedings of the IEEE international conference on computer vision, pp. 1026-1034. 2015.

## 另请参阅

[trainNetwork](#) | [trainingOptions](#)

## 相关示例

- “指定自定义权重初始化函数”（第 17-16 页）
- “深度学习层列表”（第 1-18 页）
- “Deep Learning Tips and Tricks”
- “在 MATLAB 中进行深度学习”（第 1-2 页）

## 基于预训练的 Keras 层组合网络

此示例说明如何从预训练的 Keras 网络中导入层、用自定义层替换不支持的层，以及将各层组合成可以进行预测的网络。

### 导入 Keras 网络

从 Keras 网络模型中导入层。'digitsDAGnetwithnoise.h5' 中的网络可对数字图像进行分类。

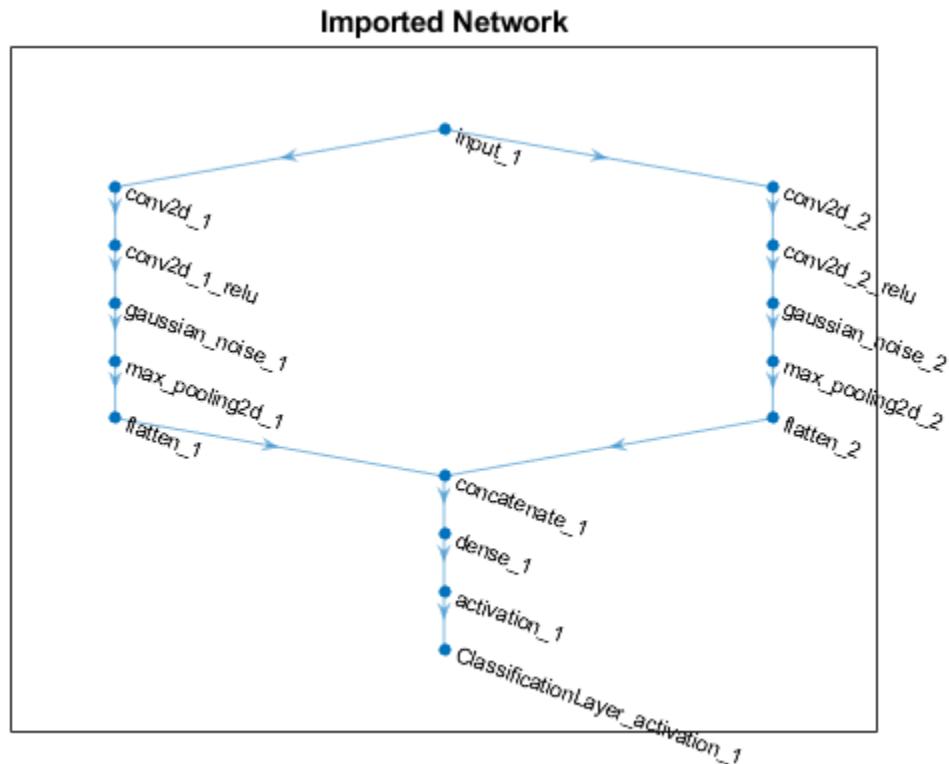
```
filename = 'digitsDAGnetwithnoise.h5';
lgraph = importKerasLayers(filename,'ImportWeights',true);
```

Warning: Unable to import some Keras layers, because they are not supported by the Deep Learning Toolbox. T...

Keras 网络包含一些 Deep Learning Toolbox 不支持的层。importKerasLayers 函数会显示警告，并用占位符层替换不支持的层。

使用 plot 绘制层次图。

```
figure
plot(lgraph)
title("Imported Network")
```



### 替换占位层

要替换占位层，请首先标识要替换的层的名称。使用 findPlaceholderLayers 查找占位层。

```
placeholderLayers = findPlaceholderLayers(lgraph)
```

```
placeholderLayers =  
    2x1 PlaceholderLayer array with layers:  
  
    1 'gaussian_noise_1'  PLACEHOLDER LAYER  Placeholder for 'GaussianNoise' Keras layer  
    2 'gaussian_noise_2'  PLACEHOLDER LAYER  Placeholder for 'GaussianNoise' Keras layer
```

显示这些层的 Keras 配置。

#### placeholderLayers.KerasConfiguration

```
ans = struct with fields:  
  trainable: 1  
    name: 'gaussian_noise_1'  
    stddev: 1.5000  
  
ans = struct with fields:  
  trainable: 1  
    name: 'gaussian_noise_2'  
    stddev: 0.7000
```

定义自定义高斯噪声层。要创建此层，请将文件 **gaussianNoiseLayer.m** 保存在当前文件夹中。然后，创建两个高斯噪声层，它们的配置与导入的 Keras 层相同。

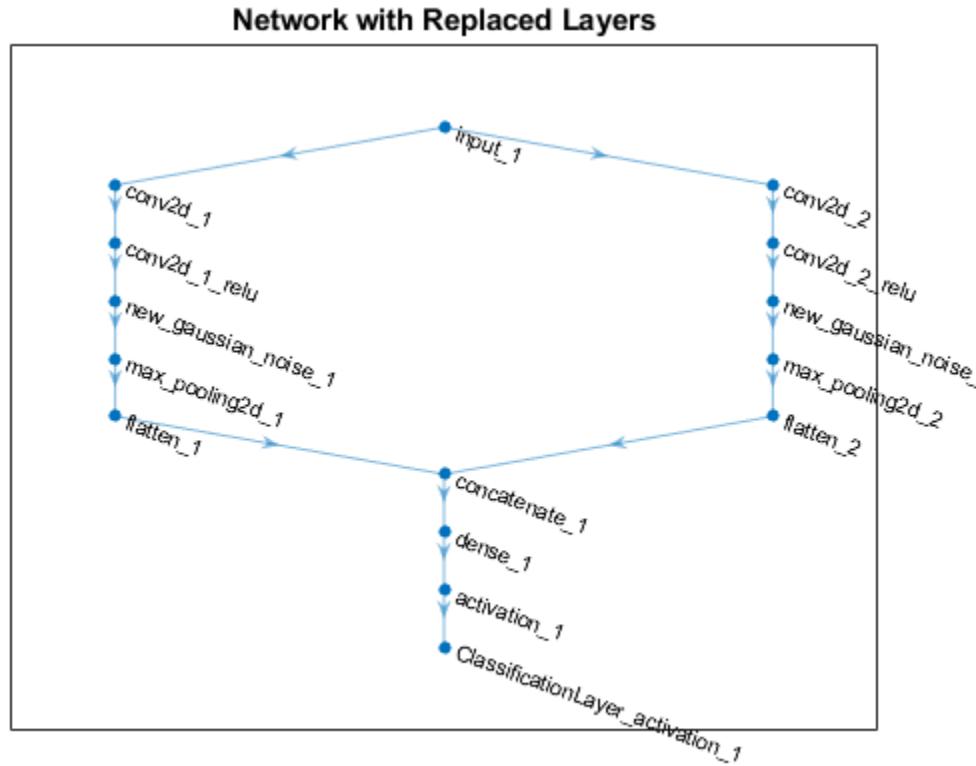
```
gnLayer1 = gaussianNoiseLayer(1.5,'new_gaussian_noise_1');  
gnLayer2 = gaussianNoiseLayer(0.7,'new_gaussian_noise_2');
```

使用 **replaceLayer** 将占位层替换为自定义层。

```
lgraph = replaceLayer(lgraph,'gaussian_noise_1',gnLayer1);  
lgraph = replaceLayer(lgraph,'gaussian_noise_2',gnLayer2);
```

使用 **plot** 绘制更新后的层次图。

```
figure  
plot(lgraph)  
title("Network with Replaced Layers")
```



### 指定类名称

如果导入的分类层不包含类，则必须在进行预测之前指定这些类。如果不指定类，软件会自动将类设置为 1、2、...、N，其中 N 是类的数量。

通过查看层次图的 `Layers` 属性，查找分类层的索引。

`lgraph.Layers`

```

ans =
15x1 Layer array with layers:

 1 'input_1'           Image Input      28x28x1 images
 2 'conv2d_1'          Convolution    20 7x7x1 convolutions with stride [1 1] and padding 'same'
 3 'conv2d_1_relu'     ReLU          ReLU
 4 'conv2d_2'          Convolution    20 3x3x1 convolutions with stride [1 1] and padding 'same'
 5 'conv2d_2_relu'     ReLU          ReLU
 6 'new_gaussian_noise_1' Gaussian Noise  Gaussian noise with standard deviation 1.5
 7 'new_gaussian_noise_2' Gaussian Noise  Gaussian noise with standard deviation 0.7
 8 'max_pooling2d_1'   Max Pooling    2x2 max pooling with stride [2 2] and padding 'same'
 9 'max_pooling2d_2'   Max Pooling    2x2 max pooling with stride [2 2] and padding 'same'
10 'flatten_1'          Keras Flatten  Flatten activations into 1-D assuming C-style (row-major) order
11 'flatten_2'          Keras Flatten  Flatten activations into 1-D assuming C-style (row-major) order
12 'concatenate_1'      Depth concatenation Depth concatenation of 2 inputs
13 'dense_1'            Fully Connected 10 fully connected layer
14 'activation_1'       Softmax        softmax
15 'ClassificationLayer_activation_1' Classification Output crossentropyex
  
```

分类层的名称为 'ClassificationLayer\_activation\_1'。查看分类层并检查 **Classes** 属性。

```
cLayer = lgraph.Layers(end)

cLayer =
ClassificationOutputLayer with properties:

    Name: 'ClassificationLayer_activation_1'
    Classes: 'auto'
    ClassWeights: 'none'
    OutputSize: 'auto'

    Hyperparameters
    LossFunction: 'crossentropyex'
```

由于层的 **Classes** 属性为 'auto'，因此您必须手动指定类。将类设置为 0、1、...、9，然后将导入的分类层替换为新层。

```
cLayer.Classes = string(0:9)

cLayer =
ClassificationOutputLayer with properties:

    Name: 'ClassificationLayer_activation_1'
    Classes: [0 1 2 3 4 5 6 7 8 9]
    ClassWeights: 'none'
    OutputSize: 10

    Hyperparameters
    LossFunction: 'crossentropyex'
```

```
lgraph = replaceLayer(lgraph,'ClassificationLayer_activation_1',cLayer);
```

## 组合网络

使用 **assembleNetwork** 组合层次图。此函数将返回一个可以用于预测的 **DAGNetwork** 对象。

```
net = assembleNetwork(lgraph)

net =
DAGNetwork with properties:

    Layers: [15x1 nnet.cnn.layer.Layer]
    Connections: [15x2 table]
    InputNames: {'input_1'}
    OutputNames: {'ClassificationLayer_activation_1'}
```

## 另请参阅

[importKerasNetwork](#) | [assembleNetwork](#) | [replaceLayer](#) | [importKerasLayers](#) | [trainNetwork](#) | [layerGraph](#) | [DAGNetwork](#) | [findPlaceholderLayers](#)

## 相关示例

- “在 MATLAB 中进行深度学习”（第 1-2 页）

- “预训练的深度神经网络” (第 1-8 页)
- “定义自定义深度学习层” (第 17-2 页)

# 定义自定义训练循环、损失函数和网络

## 本节内容

[“为自定义训练循环定义深度学习网络”（第 17-33 页）](#)

[“指定损失函数”（第 17-35 页）](#)

[“使用自动微分更新可学习参数”（第 17-36 页）](#)

对于大多数深度学习任务，您可以使用预训练网络，并使其适应您自己的数据。有关说明如何使用迁移学习来重新训练卷积神经网络以对新图像集进行分类的示例，请参阅“[训练深度学习网络以对新图像进行分类](#)”（第 3-6 页）。您也可以通过对 `layerGraph` 对象使用 `trainNetwork` 和 `trainingOptions` 函数来从头创建和训练网络。

如果 `trainingOptions` 函数没有提供您的任务所需的训练选项，则您可以使用自动微分创建自定义训练循环。要了解详细信息，请参阅“[为自定义训练循环定义深度学习网络](#)”（第 17-33 页）。

如果 Deep Learning Toolbox 没有提供任务所需的层（包括指定损失函数的输出层），则您可以创建自定义层。要了解详细信息，请参阅“[定义自定义深度学习层](#)”（第 17-2 页）。对于无法使用输出层指定的损失函数，您可以在自定义训练循环中指定损失。要了解详细信息，请参阅“[指定损失函数](#)”（第 17-35 页）。对于无法使用层次图创建的网络，可以将自定义网络定义为函数。要了解详细信息，请参阅“[将网络定义为模型函数](#)”（第 17-33 页）。

有关对哪项任务使用哪种训练方法的详细信息，请参阅“[Train Deep Learning Model in MATLAB](#)”。

## 为自定义训练循环定义深度学习网络

### 将网络定义为 `dlnetwork` 对象

对于大多数任务，您可以使用 `trainingOptions` 和 `trainNetwork` 函数来控制训练算法的细节。如果 `trainingOptions` 函数没有为您的任务提供所需的选项（例如，自定义学习率调度），则您可以使用 `dlnetwork` 对象定义您自己的自定义训练循环。`dlnetwork` 对象允许您使用自动微分来训练指定为层次图的网络。

对于指定为层次图的网络，您可以直接使用 `dlnetwork` 函数从层次图创建 `dlnetwork` 对象。

```
dlnet = dlnetwork(lgraph);
```

有关 `dlnetwork` 对象支持的层的列表，请参阅 `dlnetwork` 页的“[Supported Layers](#)”部分。有关如何使用自定义学习率调度来训练网络的示例，请参阅“[Train Network Using Custom Training Loop](#)”。

### 将网络定义为模型函数

对于无法使用层次图创建的架构（例如，需要共享权重的 Siamese 网络），您可以将模型定义为 `[dIY1,...,dIYM] = model(parameters,dIX1,...,dIXN)` 形式的函数，其中 `parameters` 包含网络参数，`dIX1,...,dIXN` 对应于 N 个模型输入的输入数据，`dIY1,...,dIYM` 对应于 M 个模型输出。要训练定义为函数的深度学习模型，请使用自定义训练循环。有关示例，请参阅“[Train Network Using Model Function](#)”。

将深度学习模型定义为函数时，必须手动初始化层权重。有关详细信息，请参阅“[Initialize Learnable Parameters for Model Function](#)”。

如果您将自定义网络定义为函数，则模型函数必须支持自动微分。您可以使用以下深度学习运算。此处列出的函数只是一部分。有关支持 `dlarray` 输入的函数的完整列表，请参阅“[List of Functions with dlarray Support](#)”。

函数	说明
<b>avgpool</b>	平均池化运算通过将输入划分为若干池化区域并计算每个区域的平均值来执行下采样。
<b>batchnorm</b>	批量归一化运算独立地对每个通道的所有观测值的输入数据进行归一化。为了加快卷积神经网络的训练并降低对网络初始化的敏感度，请在卷积和非线性运算之间使用批量归一化，如 <code>relu</code> 。
<b>crossentropy</b>	交叉熵运算计算单标签和多标签分类任务的网络预测和目标值之间的交叉熵损失。
<b>crosschannelnorm</b>	跨通道归一化运算使用不同通道中的局部响应来归一化每次激活。跨通道归一化通常遵循 <code>relu</code> 运算。跨通道归一化也称为局部响应归一化。
<b>ctc</b>	CTC 运算计算未对齐序列之间的连接时序分类 (CTC) 损失。
<b>dlconv</b>	卷积运算对输入数据应用滑动滤波器。使用 <code>dlconv</code> 函数进行深度学习卷积、分组卷积和按通道可分离卷积。
<b>dltranspconv</b>	转置卷积运算对特征图进行上采样。
<b>embed</b>	嵌入运算将数值索引转换为数值向量，其中索引对应于离散数据。使用嵌入将离散数据（如分类值或单词）映射到数值向量。
<b>fullyconnect</b>	全连接运算将输入乘以权重矩阵，然后添加偏置向量。
<b>groupnorm</b>	组归一化运算独立地对每个观测的通道分组子集的输入数据进行归一化。为了加快卷积神经网络的训练并降低对网络初始化的敏感度，请在卷积和非线性运算之间使用组归一化，如 <code>relu</code> 。
<b>gru</b>	门控循环单元 (GRU) 运算允许网络学习时序和序列数据中时间步之间的相关性。
<b>huber</b>	Huber 运算计算回归任务的网络预测和目标值之间的 Huber 损失。当 'TransitionPoint' 选项为 1 时，这也称为平滑 $L_1$ 损失。
<b>instancenorm</b>	实例归一化运算独立地对每个观测的每个通道的输入数据进行归一化。为了改进训练卷积神经网络的收敛性并降低对网络超参数的敏感度，请在卷积和非线性运算之间使用实例归一化，如 <code>relu</code> 。
<b>layernorm</b>	层归一化运算独立地对每个观测的所有通道的输入数据进行归一化。为了加快循环和多层次感知神经网络的训练并降低对网络初始化的敏感度，请在可学习运算后使用层归一化，例如 LSTM 和全连接运算。
<b>leakyrelu</b>	泄漏修正线性单元 (ReLU) 激活运算执行非线性阈值运算，其中任何小于零的输入值都会乘以一个固定缩放因子。

函数	说明
<b>lstm</b>	长短期记忆 (LSTM) 运算允许网络学习时序和序列数据中时间步之间的长期相关性。
<b>maxpool</b>	最大池化运算通过将输入划分为若干池化区域并计算每个区域的最大值来执行下采样。
<b>maxunpool</b>	最大去池化运算通过上采样和填充零对最大池化运算的输出进行去池化。
<b>mse</b>	半均方误差运算计算回归任务的网络预测和目标值之间的半均方误差损失。
<b>onehotdecode</b>	one-hot 解码运算将概率向量（例如分类网络的输出）解码为分类标签。 输入 A 可以是 <code>dlarray</code> 。如果 A 已格式化，函数将忽略数据格式。
<b>relu</b>	修正线性单元 (ReLU) 激活运算执行非线性阈值运算，其中任何小于零的输入值都设置为零。
<b>sigmoid</b>	<code>sigmoid</code> 激活运算将 <code>sigmoid</code> 函数应用于输入数据。
<b>softmax</b>	<code>softmax</code> 激活运算将 <code>softmax</code> 函数应用于输入数据的通道维度。

## 指定损失函数

使用自定义训练循环时，必须在模型梯度函数中计算损失。计算梯度时使用损失值来更新网络权重。要计算损失，您可以使用以下函数。

函数	说明
<b>softmax</b>	<code>softmax</code> 激活运算将 <code>softmax</code> 函数应用于输入数据的通道维度。
<b>sigmoid</b>	<code>sigmoid</code> 激活运算将 <code>sigmoid</code> 函数应用于输入数据。
<b>crossentropy</b>	交叉熵运算计算单标签和多标签分类任务的网络预测和目标值之间的交叉熵损失。
<b>huber</b>	Huber 运算计算回归任务的网络预测和目标值之间的 Huber 损失。当 'TransitionPoint' 选项为 1 时，这也称为平滑 L <sub>1</sub> 损失。
<b>mse</b>	半均方误差运算计算回归任务的网络预测和目标值之间的半均方误差损失。
<b>ctc</b>	CTC 运算计算未对齐序列之间的连接时序分类 (CTC) 损失。

您也可以通过创建 `loss = myLoss(Y,T)` 形式的函数来使用自定义损失函数，其中 Y 和 T 分别对应于网络预测和目标，`loss` 是返回的损失。

有关如何训练可通过自定义损失函数来生成图像的生成对抗网络 (GAN) 的示例，请参阅“训练生成对抗网络 (GAN)”（第 3-57 页）。

## 使用自动微分更新可学习参数

当用自定义训练循环训练深度学习模型时，软件可最小化关于可学习参数的损失。为了最小化损失，软件使用损失关于可学习参数的梯度。要使用自动微分计算这些梯度，必须定义模型梯度函数。

### 定义模型梯度函数

对于指定为 `dlnetwork` 对象的模型，创建 `gradients = modelGradients(dlnet,dIX,T)` 形式的函数，其中 `dlnet` 是网络，`dIX` 是网络输入，`T` 包含目标，`gradients` 包含返回的梯度。您也可以向梯度函数传递额外的参数（例如，当损失函数需要额外的信息时），或返回额外的参数（例如，用于绘制训练进度的指标）。

对于指定为函数的模型，创建 `gradients = modelGradients(parameters,dIX,T)` 形式的函数，其中 `parameters` 包含可学习参数，`dIX` 是模型输入，`T` 包含目标，`gradients` 包含返回的梯度。您也可以向梯度函数传递额外的参数（例如，当损失函数需要额外的信息时），或返回额外的参数（例如，用于绘制训练进度的指标）。

要了解有关为自定义训练循环定义模型梯度函数的详细信息，请参阅“Define Model Gradients Function for Custom Training Loop”。

### 更新可学习参数

要使用自动微分计算模型梯度，请使用 `dlfeval` 函数，该函数可以对启用了自动微分的函数进行计算。对于 `dlfeval` 的第一个输入，传递指定为函数句柄的模型梯度函数。对于以下输入，传递模型梯度函数所需的变量。对于 `dlfeval` 函数的输出，指定与模型梯度函数相同的输出。

要使用梯度更新可学习参数，您可以使用以下函数。

函数	说明
<code>adamupdate</code>	使用自适应矩估计 (Adam) 更新参数
<code>rmspropupdate</code>	使用均方根传播 (RMSProp) 更新参数
<code>sgdmupdate</code>	使用带动量的随机梯度下降 (SGDM) 更新参数
<code>dlupdate</code>	使用自定义函数更新参数

### 另请参阅

`dlarray` | `dlgradient` | `dlfeval` | `dlnetwork`

### 详细信息

- “训练生成对抗网络 (GAN)”（第 3-57 页）
- “Train Network Using Custom Training Loop”
- “Specify Training Options in Custom Training Loop”
- “Define Model Gradients Function for Custom Training Loop”
- “Update Batch Normalization Statistics in Custom Training Loop”
- “Update Batch Normalization Statistics Using Model Function”
- “Make Predictions Using `dlnetwork` Object”
- “Make Predictions Using Model Function”
- “Train Network Using Model Function”

- “Initialize Learnable Parameters for Model Function”
- “Train Deep Learning Model in MATLAB”
- “[定义自定义深度学习层](#)”（第 17-2 页）
- “List of Functions with dlarray Support”
- “Automatic Differentiation Background”
- “Use Automatic Differentiation In Deep Learning Toolbox”



# 深度学习数据预处理

---

- “预处理图像以进行深度学习”（第 18-2 页）
- “为图像到图像的回归准备数据存储”（第 18-6 页）
- “使用无法放入内存的序列数据训练网络”（第 18-14 页）
- “使用序列数据的自定义小批量数据存储来训练网络”（第 18-18 页）
- “使用深度学习对无法放入内存的文本数据进行分类”（第 18-21 页）
- “使用自定义小批量数据存储对无法放入内存的文本数据进行分类”（第 18-27 页）

## 预处理图像以进行深度学习

要训练网络并基于新数据进行预测，您的图像必须与网络的输入大小相匹配。如果您需要调整图像的大小以匹配网络，则您可以将数据重新缩放或裁剪到所需的大小。

通过对数据应用随机增强，您实际上是增加了训练数据量。增强还使您能够训练网络，使其在图像数据失真时仍保持稳定。例如，您可以对输入图像添加随机旋转，使网络不会因输入图像中存在旋转而改变。**augmentedImage datastore** 提供了一种方便的方法，可以将一组有限的增强应用于二维图像来解决分类问题。

对于更高级的预处理操作，为了对图像进行预处理以解决回归问题，或者为了对三维体图像进行预处理，您可以从内置数据存储开始。您还可以使用 **transform** 和 **combine** 函数，根据自己的管道对图像进行预处理。

### 使用重新缩放和裁剪调整图像大小

您可以将图像数据存储为数值数组、**ImageDatastore** 对象或表。**ImageDatastore** 使您能够从太大而无法放入内存的图像集合中分批导入数据。您可以使用增强的图像数据存储或经过调整大小的四维数组进行训练、预测和分类。您只能将经过调整大小的三维数组用于预测和分类。

有两种方法可以调整图像数据的大小以匹配网络的输入大小。

- 重新缩放，即将图像的高度和宽度乘以缩放因子。如果缩放因子在垂直和水平方向不相同，则重新缩放会更改像素的空间范围和纵横比。
- 裁剪，即提取图像的子区域并保留每个像素的空间范围。您可以从图像的中心或随机位置裁剪图像。

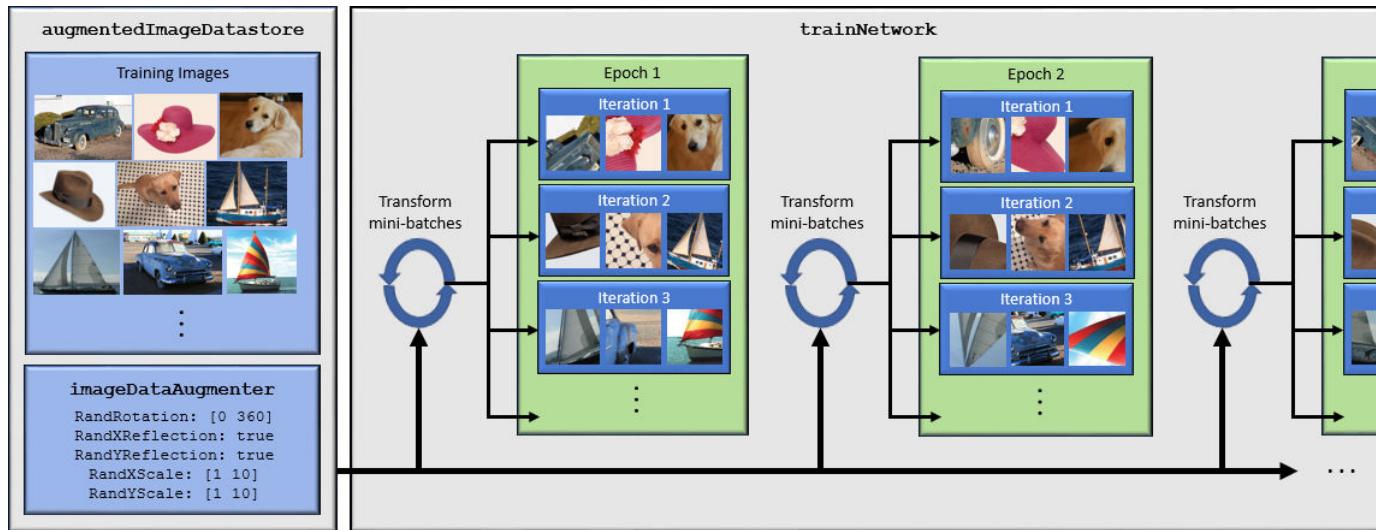
用于调整大小的选项	数据格式	用于调整大小的函数	示例代码
重新缩放	<ul style="list-style-type: none"> <li>表示单色或多光谱图像的三维数组</li> <li>表示灰度图像堆叠的三维数组</li> <li>表示图像堆叠的四维数组</li> </ul>	<b>imresize</b>	<b>im = imresize(I,outputSize);</b> <b>outputSize</b> 指定经过重新缩放的图像的维度。
	<ul style="list-style-type: none"> <li>表示图像堆叠的四维数组</li> <li><b>ImageDatastore</b></li> <li>表</li> </ul>	<b>augmentedImage datastore</b>	<b>auimds = augmentedImage datastore(outputSize);</b> <b>outputSize</b> 指定经过重新缩放的图像的维度。
裁剪	<ul style="list-style-type: none"> <li>表示单色或多光谱图像的三维数组</li> </ul>	<b>imcrop</b>	<b>im = imcrop(I,rect);</b> <b>rect</b> 指定二维裁剪窗口的大小和位置。
	<ul style="list-style-type: none"> <li>表示灰度图像堆叠的三维数组</li> <li>表示颜色或多光谱图像堆叠的四维数组</li> </ul>	<b>imcrop3</b>	<b>im = imcrop3(I,cuboid);</b> <b>cuboid</b> 指定三维裁剪窗口的大小和位置。

用于调整大小的选项	数据格式	用于调整大小的函数	示例代码
	<ul style="list-style-type: none"> <li>表示图像堆叠的四维数组</li> <li><b>ImageDatastore</b></li> <li>表</li> </ul>	<b>augmentedImage datastore</b>	<pre>auimds = augmentedImageDatastore(output) 将 m 指定为 'centercrop' 表示从输入图像的中心进行裁剪。 将 m 指定为 'randcrop' 表示从输入图像的随机位置进行裁剪。</pre>

## 用随机几何变换增强图像进行训练

对于图像分类问题，您可以使用 **augmentedImageDatastore** 通过调整大小、旋转、翻转、剪切和平移变换的随机组合来增强图像。

下图显示 **trainNetwork** 如何使用增强的图像数据存储来变换每轮的训练数据。当您使用数据增强时，在每轮训练中使用每个图像的一个随机增强版本。有关工作流的示例，请参阅“Train Network with Augmented Images”。



- 1 指定训练图像。
- 2 通过创建  **imageDataAugmenter**，配置图像变换选项，如旋转角度范围以及是否随机应用翻转。

**提示** 要预览应用于示例图像的变换，请使用  **augment** 函数。

- 3 创建一个  **augmentedImageDatastore**。指定训练图像、输出图像的大小以及  **imageDataAugmenter**。输出图像的大小必须与网络的  **imageInputLayer** 的大小兼容。
- 4 训练网络，将增强的图像数据存储指定为  **trainNetwork** 的数据源。对于训练的每次迭代，增强的图像数据存储对训练数据的小批量中的图像应用随机组合的变换。

当您使用增强的图像数据存储作为训练图像的数据源时，数据存储会随机扰动每轮的训练数据，因此每轮使用的数据集略有不同。每轮的训练图像的实际数量不变。经过变换的图像不会存储在内存中。

## 使用内置数据存储执行附加图像处理运算

一些数据存储在读取一批数据时会执行特定且有限的图像预处理操作。下表中列出了这些特定于应用的数据存储。您可以将这些数据存储用作使用 Deep Learning Toolbox 的深度学习应用情形的训练、验证和测试数据集的数据源。所有这些数据存储都以 `trainNetwork` 支持的格式返回数据。

数据存储	说明
<code>augmentedImageDatastore</code>	应用随机仿射几何变换，包括调整大小、旋转、翻转、剪切和平移，来训练深度神经网络。有关示例，请参阅“Transfer Learning Using Pretrained Network”。
<code>pixelLabelImageDatastore</code>	将相同的仿射几何变换应用于图像和对应的真实值标签，用于训练语义分割网络（需要 Computer Vision Toolbox™）。有关示例，请参阅“使用深度学习进行语义分割”（第 8-11 页）。
<code>randomPatchExtractionDatastore</code>	从图像或像素标签图像中提取多对随机补片（需要 Image Processing Toolbox™）。您可以选择对各对补片应用相同的随机仿射几何变换。有关示例，请参阅“使用深度学习执行单图像超分辨率”（第 9-8 页）。
<code>denoisingImageDatastore</code>	应用随机生成的高斯噪声来训练去噪网络（需要 Image Processing Toolbox）。

## 通过组合和变换来应用自定义图像处理管道

要执行比特定于应用的数据存储提供的操作更常见和更复杂的图像预处理操作，您可以使用 `transform` 和 `combine` 函数。有关详细信息，请参阅“Datastores for Deep Learning”。

### 用图像数据变换数据存储

`transform` 函数根据您定义的变换函数来变换基础数据存储读取的数据，从而创建一个经过更改的数据存储形式，称为基础数据存储。

自定义变换函数必须接受基础数据存储的 `read` 函数返回的格式的数据。对于 `ImageDatastore` 中的图像数据，格式取决于 `ReadSize` 属性。

- 当 `ReadSize` 为 1 时，变换函数必须接受整数数组。数组的大小与 `ImageDatastore` 中图像的类型一致。例如，灰度图像的维度为  $m \times n$ ，真彩色图像的维度为  $m \times n \times 3$ ，具有  $c$  个通道的多光谱图像的维度为  $m \times n \times c$ 。
- 当 `ReadSize` 大于 1 时，变换函数必须接受由图像数据组成的元胞数组。每个元素对应于批量中的一个图像。

`transform` 函数必须返回与网络的输入大小相匹配的数据。`transform` 函数不支持一对多观测值映射。

**提示** 当基础 `ImageDatastore` 读取一批 JPG 或 PNG 图像文件时，`transform` 函数支持预取。对于这些图像类型，请不要使用 `ImageDatastore` 的 `readFcn` 参数来应用图像预处理，因为此选项通常会显著减慢速度。如果您使用自定义读取函数，则 `ImageDatastore` 不会预取。

### 将数据存储与图像数据结合使用

`combine` 函数将从多个数据存储读取的数据串联起来，并保持数据存储之间的奇偶校验。

- 将数据串联为一个包含两列的表或一个包含两列的元胞数组，用于训练具有单一输入的网络，如图像到图像回归网络。

- 将数据串联为一个包含  $(\text{numInputs}+1)$  个列的元胞数组，用于训练具有多个输入的网络。

## 另请参阅

`trainNetwork` | `imresize` | `transform` | `combine` | `ImageDatastore`

## 相关示例

- "Train Network with Augmented Images"
- "训练深度学习网络以对新图像进行分类" (第 3-6 页)
- "Create and Explore Datastore for Image Classification"
- "为图像到图像的回归准备数据存储" (第 18-6 页)

## 详细信息

- "Datastores for Deep Learning"
- "Preprocess Volumes for Deep Learning"
- "在 MATLAB 中进行深度学习" (第 1-2 页)

## 为图像到图像的回归准备数据存储

此示例说明如何准备数据存储，以便使用 `ImageDatastore` 的 `transform` 和 `combine` 函数来训练图像到图像的回归网络。

此示例说明如何使用适于训练去噪网络的管道预处理数据。然后，此示例使用预处理后的含噪数据来训练简单的卷积自编码器网络，以去除图像噪声。

### 使用预处理管道准备数据

此示例使用椒盐噪声模型，其中一部分输入图像像素设置为 0 或 1（分别表示黑色和白色）。含噪图像充当网络输入。原始图像充当预期的网络响应。网络学习检测和消除椒盐噪声。

将数字数据集中的原始图像加载为 `ImageDatastore`。该数据存储包含 10,000 个数字 0 至 9 的合成图像。这些图像是通过对使用不同字体创建的数字图像应用随机变换生成的。每个数字图像为  $28 \times 28$  像素。该数据存储包含的每个类别都有相同数量的图像。

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet',...
    'nnndemos','nndatasets','DigitDataset');
imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

指定较大的读取大小，以最小化文件 I/O 的成本。

```
imds.ReadSize = 500;
```

设置全局随机数生成器的种子，以帮助实现结果的可再现性。

```
rng(0)
```

在训练前，使用 `shuffle` 函数打乱数字数据。

```
imds = shuffle(imds);
```

使用 `splitEachLabel` 函数将 `imds` 划分为三个图像数据存储，分别包含用于训练、验证和测试的原始图像。

```
[imdsTrain,imdsVal,imdsTest] = splitEachLabel(imds,0.95,0.025);
```

使用 `transform` 函数创建每个输入图像的含噪版本，它们将用作网络输入。`transform` 函数从基础数据存储中读取数据，并使用在辅助函数 `addNoise`（在本示例末尾定义）中定义的操作来处理数据。`transform` 函数的输出是一个 `TransformedDatastore`。

```
dsTrainNoisy = transform(imdsTrain,@addNoise);
dsValNoisy = transform(imdsVal,@addNoise);
dsTestNoisy = transform(imdsTest,@addNoise);
```

使用 `combine` 函数将含噪图像和原始图像合并到一个数据存储中，该数据存储将数据馈送到 `trainNetwork`。正如 `trainNetwork` 所预期的那样，该合并后的数据存储将数据批次读入一个两列元胞数组。`combine` 函数的输出是一个 `CombinedDatastore`。

```
dsTrain = combine(dsTrainNoisy,imdsTrain);
dsVal = combine(dsValNoisy,imdsVal);
dsTest = combine(dsTestNoisy,imdsTest);
```

使用 `transform` 函数执行对输入和响应数据存储通用的其他预处理操作。`commonPreprocessing` 辅助函数（在本例末尾定义）将输入和响应图像的大小调整为  $32 \times 32$  像素以匹配网络的输入大小，并将每个图像中的数据归一化到范围  $[0, 1]$ 。

```
dsTrain = transform(dsTrain,@commonPreprocessing);
dsVal = transform(dsVal,@commonPreprocessing);
dsTest = transform(dsTest,@commonPreprocessing);
```

最后，使用 `transform` 函数将随机化的增强添加到训练集中。`augmentImages` 辅助函数（在本示例末尾定义）对数据应用随机化的 90 度旋转。相同的旋转应用于网络输入和对应的预期响应。

```
dsTrain = transform(dsTrain,@augmentImages);
```

增强可减少过拟合，让经过训练的网络能够更好地处理经过旋转的数据。验证数据集或测试数据集不需要随机化的增强。

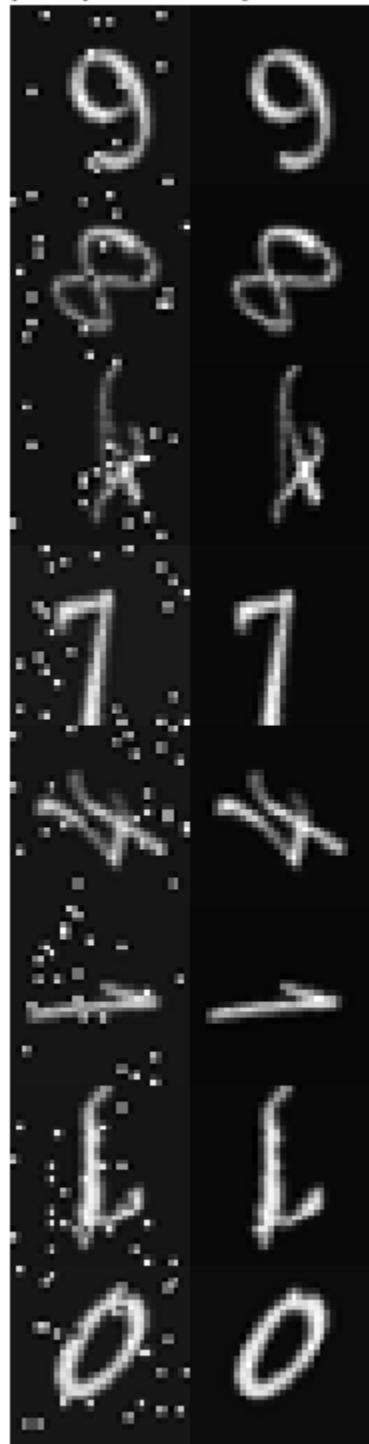
### 预览经过预处理的数据

由于准备训练数据需要几个预处理操作，请在训练前预览经过预处理的数据以确认它看起来是正确的。使用 `preview` 函数预览数据。

使用 `montage` (Image Processing Toolbox) 函数可视化成对的含噪和原始图像的示例。训练数据看起来是正确的。盐和胡椒噪声显示在左列的输入图像中。除了增加噪声外，输入图像和响应图像是相同的。以相同的方式同时对输入图像和响应图像随机旋转 90 度。

```
exampleData = preview(dsTrain);
inputs = exampleData(:,1);
responses = exampleData(:,2);
minibatch = cat(2,inputs,responses);
montage(minibatch,'Size',[8 2])
title('Inputs (Left) and Responses (Right)')
```

Inputs (Left) and Responses (Right)



## 定义卷积自编码器网络

卷积自编码器是一种常见的图像去噪架构。卷积自编码器由两个阶段组成：编码器和解码器。编码器将原始输入图像压缩成一个潜在表示，其宽度和高度较小，但比原始输入图像更深，因为每个空间位置有更多特征映射。压缩的潜在表示在恢复原始图像中的高频特征时会损失一些空间分辨率，但它学会了在进行原始图像编码时不引入噪声伪影。解码器重复对编码信号进行上采样，以将其移回其原始宽度、高度和通道数。由于编码器可去除噪声，经过解码的最终图像具有较少的噪声伪影。

此示例使用 Deep Learning Toolbox™ 中的层定义卷积自编码器网络，这些层包括：

- **imageInputLayer** - 图像输入层
- **convolution2dLayer** - 卷积神经网络的卷积层
- **reluLayer** - 修正线性单元层
- **maxPooling2dLayer** - 二维最大池化层
- **transposedConv2dLayer** - 转置卷积层
- **clippedReluLayer** - 裁剪后的修正线性单元层
- **regressionLayer** - 回归输出层

创建图像输入层。为了简化与以 2 为因子的下采样和上采样相关的填充问题，请选择  $32 \times 32$  输入大小，因为 32 可以被 2、4 和 8 整除。

```
imageLayer = imageInputLayer([32,32,1]);
```

创建编码层。编码器中的下采样是通过池大小为 2、步幅为 2 的最大池化实现的。

```
encodingLayers = [ ...
    convolution2dLayer(3,16,'Padding','same'), ...
    reluLayer, ...
    maxPooling2dLayer(2,'Padding','same','Stride',2), ...
    convolution2dLayer(3,8,'Padding','same'), ...
    reluLayer, ...
    maxPooling2dLayer(2,'Padding','same','Stride',2), ...
    convolution2dLayer(3,8,'Padding','same'), ...
    reluLayer, ...
    maxPooling2dLayer(2,'Padding','same','Stride',2)];
```

创建解码层。解码器使用转置卷积层对编码信号进行上采样。使用 **createUpsampleTransposeConvLayer** 辅助函数创建具有正确上采样因子的转置卷积层。该函数在此示例的末尾定义。

网络使用 **clippedReluLayer** 作为最终激活层，以强制输出在  $[0, 1]$  范围内。

```
decodingLayers = [ ...
    createUpsampleTransposeConvLayer(2,8), ...
    reluLayer, ...
    createUpsampleTransposeConvLayer(2,8), ...
    reluLayer, ...
    createUpsampleTransposeConvLayer(2,16), ...
    reluLayer, ...
    convolution2dLayer(3,1,'Padding','same'), ...
    clippedReluLayer(1.0), ...
    regressionLayer];
```

串联图像输入层、编码层和解码层，以形成卷积自编码器网络架构。

```
layers = [imageLayer,encodingLayers,decodingLayers];
```

### 定义训练选项

使用 Adam 优化器训练网络。使用 `trainingOptions` 函数指定超参数设置。进行 100 轮训练。

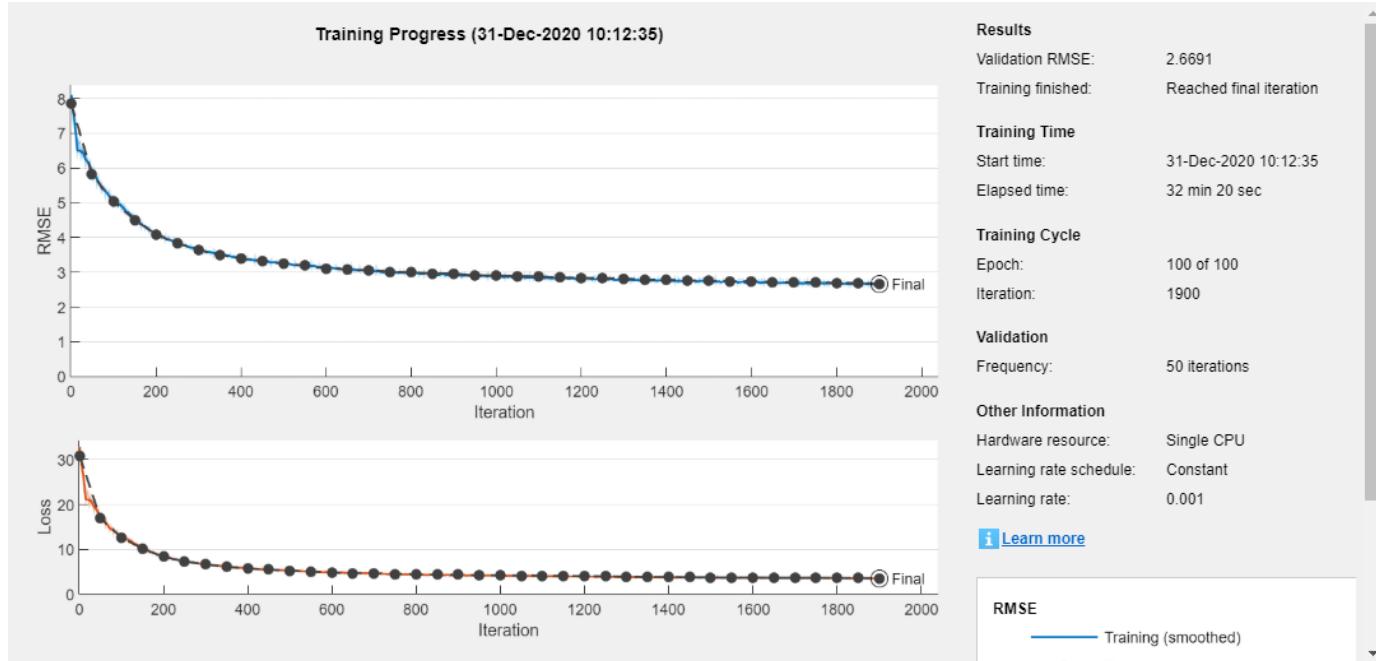
```
options = trainingOptions('adam', ...
    'MaxEpochs',100, ...
    'MiniBatchSize',imds.ReadSize, ...
    'ValidationData',dsVal, ...
    'Plots','training-progress', ...
    'Verbose',false);
```

### 训练网络

现在已配置数据源和训练选项，接下来使用 `trainNetwork` 函数训练卷积自编码器网络。

在 GPU 上（如果有）进行训练。使用 GPU 需要 Parallel Computing Toolbox™ 和支持 CUDA® 的 NVIDIA® GPU。有关详细信息，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。在 NVIDIA Titan XP 上训练需要 25 分钟。

```
net = trainNetwork(dsTrain,layers,options);
```



```
modelDateTime = string(datetime('now','Format','yyyy-MM-dd-HH-mm-ss'));
save(strcat("trainedImageToImageRegressionNet-",modelDateTime,".mat"),'net');
```

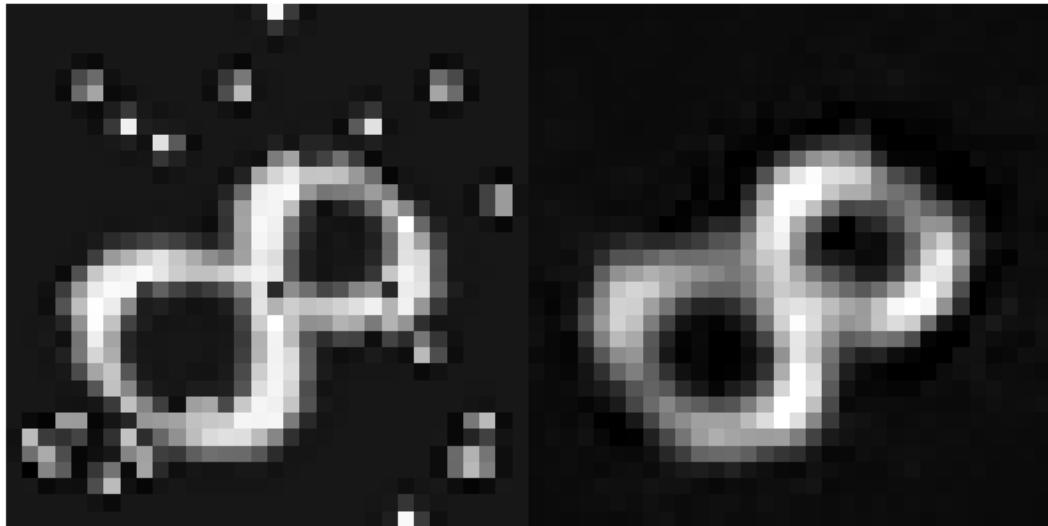
### 评估去噪网络的性能

使用 `predict` 函数从测试集中获取输出图像。

```
ypred = predict(net,dsTest);
```

可视化样本输入图像和来自网络的关联预测输出，以了解去噪效果。与预期相符，网络的输出图像已去除输入图像中的大部分噪声伪影。经过编码和解码过程后，去噪图像稍显模糊。

```
inputImageExamples = preview(dsTest);
montage({inputImageExamples{1},ypred(:,:,:,:)});
```



通过分析峰值信噪比 (PSNR) 评估网络性能。

```
ref = inputImageExamples{1,2};
originalNoisyImage = inputImageExamples{1,1};
psnrNoisy = psnr(originalNoisyImage,ref)

psnrNoisy = single
17.8455

psnrDenoised = psnr(ypred(:,:,:,:),ref)

psnrDenoised = single
21.8439
```

输出图像的 PSNR 高于含噪输入图像，这与预期相符。

### 支持函数

**addNoise** 辅助函数使用 **imnoise** (Image Processing Toolbox) 函数在图像中添加椒盐噪声。  
**addNoise** 函数要求输入数据的格式为图像数据元胞数组，这与 **ImageDatastore** 的 **read** 函数返回的数据格式相匹配。

```
function dataOut = addNoise(data)

    dataOut = data;
    for idx = 1:size(data,1)
        dataOut{idx} = imnoise(data{idx},'salt & pepper');
    end

end
```

**commonPreprocessing** 辅助函数定义训练集、验证集和测试集共有的预处理。该辅助函数执行以下预处理步骤。

- 1 将图像数据转换为数据类型 `single`。
- 2 使用 `imresize` (Image Processing Toolbox) 函数调整图像数据的大小以匹配输入层的大小。
- 3 使用 `rescale` 函数将数据归一化到范围 [0, 1]。

该辅助函数要求输入数据的格式为图像数据的两列元胞数组，这与 `CombinedDatastore` 的 `read` 函数返回的数据格式相匹配。

```
function dataOut = commonPreprocessing(data)

    dataOut = cell(size(data));
    for col = 1:size(data,2)
        for idx = 1:size(data,1)
            temp = single(data{idx,col});
            temp = imresize(temp,[32,32]);
            temp = rescale(temp);
            dataOut{idx,col} = temp;
        end
    end
end
```

**augmentImages** 辅助函数使用 `rot90` 函数在数据中添加随机化的 90 度旋转值。相同的旋转应用于网络输入和对应的预期响应。函数要求输入数据的格式为图像数据的两列元胞数组，这与 `CombinedDatastore` 的 `read` 函数返回的数据格式相匹配。

```
function dataOut = augmentImages(data)

    dataOut = cell(size(data));
    for idx = 1:size(data,1)
        rot90Val = randi(4,1,1)-1;
        dataOut(idx,:) = {rot90(data{idx,1},rot90Val),rot90(data{idx,2},rot90Val)};
    end
end
```

**createUpsampleTransposeConvLayer** 辅助函数定义一个转置卷积层，该层以指定因子对层输入进行上采样。

```
function out = createUpsampleTransposeConvLayer(factor,numFilters)

    filterSize = 2*factor - mod(factor,2);
    cropping = (factor-mod(factor,2))/2;
    numChannels = 1;

    out = transposedConv2dLayer(filterSize,numFilters, ...
        'NumChannels',numChannels,'Stride',factor,'Cropping',cropping);
end
```

## 另请参阅

`trainNetwork` | `trainingOptions` | `transform` | `combine` | `imageDatastore`

## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)

### 详细信息

- “Datastores for Deep Learning”

## 使用无法放入内存的序列数据训练网络

此示例说明如何通过变换和合并数据存储基于无法放入内存的序列数据来训练深度学习网络。

变换后的数据存储对从基础数据存储读取的数据进行变换或处理。您可以使用变换后的数据存储作为深度学习应用的训练数据集、验证数据集、测试数据集以及预测数据集的数据源。使用变换后的数据存储可读取无法放入内存的数据，或者在读取批量数据时执行特定的预处理操作。当您有若干单独的包含预测变量和标签的数据存储时，您可以将它们合并起来，以便将数据输入到深度学习网络中。

在训练网络时，软件通过填充、截断或拆分输入数据来创建长度相同的小批量序列。对于内存数据，**trainingOptions** 函数提供了填充和截断输入序列的选项，但对于内存外的数据，您必须手动填充和截断序列。

### 加载训练数据

按照 [1] 和 [2] 中的说明加载日语元音数据集。zip 文件 **japaneseVowels.zip** 包含不同长度的序列。这些序列分成两个文件夹 **Train** 和 **Test**，分别包含训练序列和测试序列。在每个文件夹中，序列又分成编号从 1 到 9 的子文件夹。这些子文件夹的名称是标签名称。一个 MAT 文件表示一个序列。每个序列均为一个包含 12 行（每个特征占一行）和不同列数（每个时间步占一列）的矩阵。行数是序列的维度，列数是序列的长度。

解压缩序列数据。

```
filename = "japaneseVowels.zip";
outputFolder = fullfile(tempdir,"japaneseVowels");
unzip(filename,outputFolder);
```

对于训练预测变量，请创建一个文件数据存储，并将读取函数指定为 **load** 函数。**load** 函数将数据从 MAT 文件加载到结构体数组中。要从训练文件夹的子文件夹中读取文件，请将 '**IncludeSubfolders**' 选项设置为 **true**。

```
folderTrain = fullfile(outputFolder,"Train");
fdsPredictorTrain = fileDatastore(folderTrain, ...
    'ReadFcn',@load, ...
    'IncludeSubfolders',true);
```

预览数据存储。返回的结构体包含来自第一个文件的单个序列。

```
preview(fdsPredictorTrain)

ans = struct with fields:
    X: [12×20 double]
```

对于标签，请创建一个文件数据存储，并将读取函数指定为 **readLabel** 函数，该函数在示例末尾定义。**readLabel** 函数从子文件夹名称中提取标签。

```
classNames = string(1:9);
fdsLabelTrain = fileDatastore(folderTrain, ...
    'ReadFcn',@(filename) readLabel(filename,classNames), ...
    'IncludeSubfolders',true);
```

预览数据存储。输出对应于第一个文件的标签。

```
preview(fdsLabelTrain)
```

```
ans = categorical
1
```

### 变换和合并数据存储

要将预测变量的数据存储中的序列数据输入深度学习网络，序列的小批量必须具有相同的长度。使用在数据存储末尾定义的 `padSequence` 函数变换数据存储，该函数填充或截断序列以使其长度为 20。

```
sequenceLength = 20;
tdsTrain = transform(fdsPredictorTrain,@(data) padSequence(data,sequenceLength));
```

预览变换后的数据存储。输出对应于来自第一个文件的填充序列。

```
X = preview(tdsTrain)
```

```
X = 1×1 cell array
{12×20 double}
```

要将来自两个数据存储的预测变量和标签输入一个深度学习网络，请使用 `combine` 函数将其合并。

```
cdsTrain = combine(tdsTrain, fdsLabelTrain);
```

预览合并后的数据存储。数据存储返回一个  $1 \times 2$  元胞数组。第一个元素对应于预测变量。第二个元素对应于标签。

```
preview(cdsTrain)
```

```
ans = 1×2 cell array
{12×20 double} {1}
```

### 定义 LSTM 网络架构

定义 LSTM 网络架构。将输入数据的特征数量指定为输入大小。指定具有 100 个隐含单元的 LSTM 层，并输出序列的最后一个元素。最后，指定一个输出大小等于类数的全连接层，后接一个 softmax 层和一个分类层。

```
numFeatures = 12;
numClasses = numel(classNames);
numHiddenUnits = 100;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits, 'OutputMode', 'last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

指定训练选项。将求解器设置为 '`adam`'，将 '`GradientThreshold`' 设置为 2。将小批量大小设置为 27，并将最大训练轮数设置为 75。数据存储不支持乱序，因此将 '`Shuffle`' 设置为 '`'never'`'。

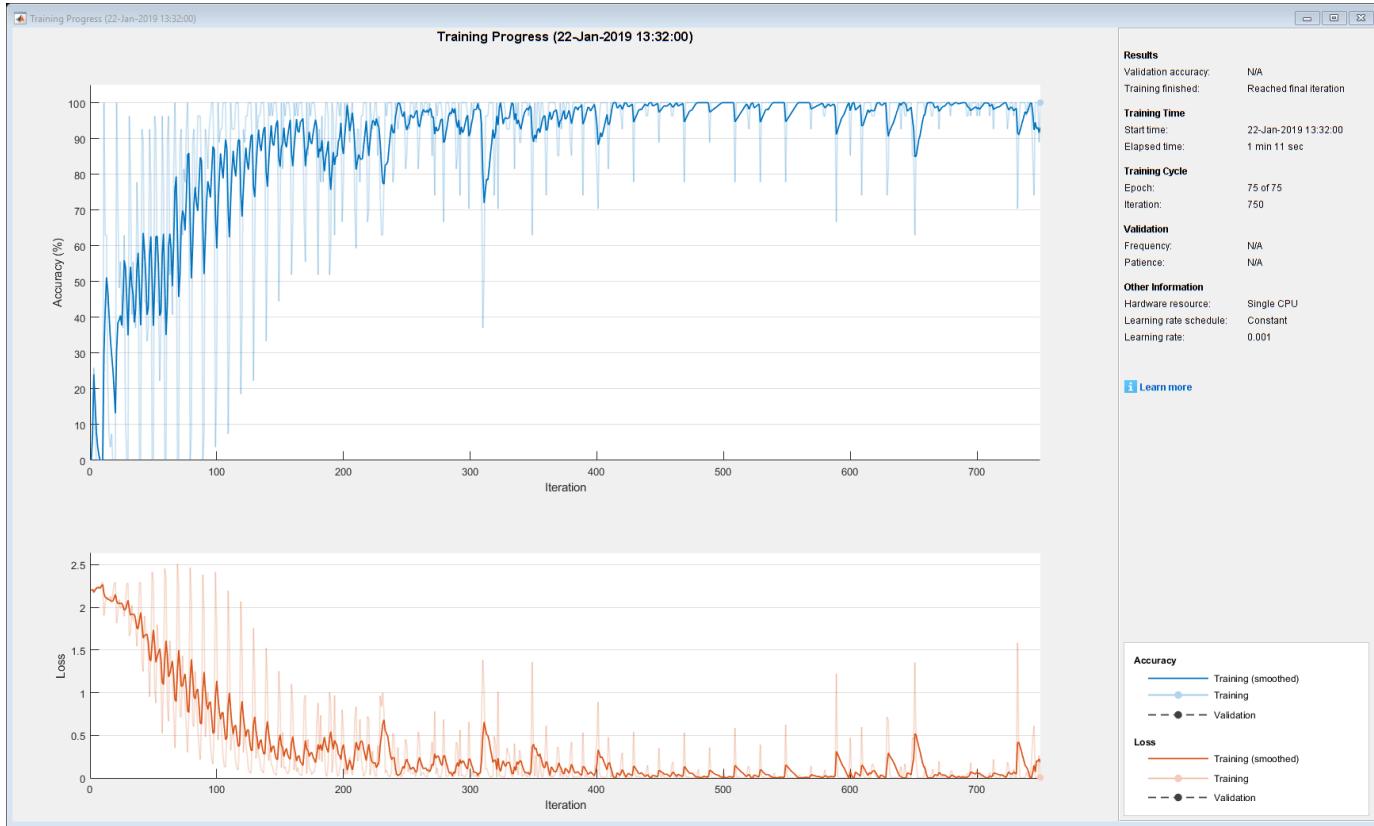
由于小批量数据存储较小且序列较短，因此更适合在 CPU 上训练。将 '`ExecutionEnvironment`' 设置为 '`'cpu'`'。要在 GPU (如果可用) 上进行训练，请将 '`ExecutionEnvironment`' 设置为 '`'auto'`' (默认值)。

```
miniBatchSize = 27;
```

```
options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'MaxEpochs',75, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',2, ...
    'Shuffle','never',...
    'Verbose',0, ...
    'Plots','training-progress');
```

使用指定的训练选项训练 LSTM 网络。

```
net = trainNetwork(cdsTrain,layers,options);
```



## 测试网络

使用与训练数据相同的步骤来创建包含保留测试数据的变换后的数据存储。

```
folderTest = fullfile(outputFolder,"Test");
fdsPredictorTest = fileDatastore(folderTest, ...
    'ReadFcn',@load, ...
    'IncludeSubfolders',true);
tdsTest = transform(fdsPredictorTest,@(data) padSequence(data,sequenceLength));
```

使用经过训练的网络对测试数据进行预测。

```
YPred = classify(net,tdsTest,'MiniBatchSize',miniBatchSize);
```

基于测试数据计算分类准确度。要获取测试集的标签，请使用读取函数 `readLabel` 创建一个文件数据存储，并指定包含子文件夹。通过将 '`UniformRead`' 选项设置为 `true`，指定输出可垂直串联。

```
fdsLabelTest = fileDatastore(folderTest, ...
    'ReadFcn',@(filename) readLabel(filename,classNames), ...
    'IncludeSubfolders',true, ...
    'UniformRead',true);
YTest = readall(fdsLabelTest);

accuracy = mean(YPred == YTest)

accuracy = 0.9351
```

## 函数

`readLabel` 函数根据 `classNames` 中的类别从指定文件名中提取标签。

```
function label = readLabel(filename,classNames)
```

```
filepath = fileparts(filename);
[~,label] = fileparts(filepath);

label = categorical(string(label),classNames);
```

```
end
```

`padSequence` 函数填充或截断 `data.X` 中的序列，使其具有指定的序列长度，并以  $1 \times 1$  元胞形式返回结果。

```
function sequence = padSequence(data,sequenceLength)
```

```
sequence = data.X;
[C,S] = size(sequence);
```

```
if S < sequenceLength
    padding = zeros(C,sequenceLength-S);
    sequence = [sequence padding];
else
    sequence = sequence(:,1:sequenceLength);
end
```

```
sequence = {sequence};
```

```
end
```

## 另请参阅

`lstmLayer` | `trainNetwork` | `trainingOptions` | `sequenceInputLayer` | `combine` | `transform`

## 相关示例

- “使用深度学习进行序列分类”（第 4-2 页）
- “使用深度学习进行时序预测”（第 4-9 页）
- “长短期记忆网络”（第 1-38 页）
- “深度学习层列表”（第 1-18 页）
- “Deep Learning Tips and Tricks”

## 使用序列数据的自定义小批量数据存储来训练网络

此示例说明如何使用自定义小批量数据存储基于无法放入内存的序列数据来训练深度学习网络。

小批量数据存储是支持批量读取数据的数据存储实现。使用小批量数据存储可读取无法放入内存的数据，或者在读取批量数据时执行特定的预处理操作。您可以使用小批量数据存储作为深度学习应用程序的训练数据集、验证数据集、测试数据集以及预测数据集的源。

此示例使用自定义小批量数据存储 `sequenceDatastore.m`。您可以通过自定义数据存储函数来调整此数据存储以适应您的数据。有关说明如何创建您自己的自定义小批量数据存储的示例，请参阅“Develop Custom Mini-Batch Datastore”。

### 加载训练数据

按照 [1] 和 [2] 中的说明加载日语元音数据集。`zip` 文件 `japaneseVowels.zip` 包含不同长度的序列。这些序列分成两个文件夹 `Train` 和 `Test`，分别包含训练序列和测试序列。在每个文件夹中，序列又分成编号从 1 到 9 的子文件夹。这些子文件夹的名称是标签名称。一个 `MAT` 文件表示一个序列。每个序列均为一个包含 12 行（每个特征占一行）和不同列数（每个时间步占一列）的矩阵。行数是序列的维度，列数是序列的长度。

解压缩序列数据。

```
filename = "japaneseVowels.zip";
outputFolder = fullfile(tempdir,"japaneseVowels");
unzip(filename,outputFolder);
```

### 创建自定义小批量数据存储

创建一个自定义小批量数据存储。小批量数据存储 `sequenceDatastore` 从文件夹中读取数据，并从子文件夹名称中获取标签。要使用此数据存储，请先将文件 `sequenceDatastore.m` 保存到路径中。

使用 `sequenceDatastore` 创建包含序列数据的数据存储。

```
folderTrain = fullfile(outputFolder,"Train");
dsTrain = sequenceDatastore(folderTrain)
```

```
dsTrain =
sequenceDatastore with properties:
```

```
    Datastore: [1×1 matlab.io.datastore.FileDatastore]
    Labels: [270×1 categorical]
    NumClasses: 9
    SequenceDimension: 12
    MiniBatchSize: 128
    NumObservations: 270
```

### 定义 LSTM 网络架构

定义 LSTM 网络架构。将输入数据的序列维度指定为输入大小。指定具有 100 个隐含单元的 LSTM 层，并输出序列的最后一个元素。最后，指定一个输出大小等于类数的全连接层，后接一个 `softmax` 层和一个分类层。

```
inputSize = dsTrain.SequenceDimension;
numClasses = dsTrain.NumClasses;
numHiddenUnits = 100;
layers = [
```

```

sequenceInputLayer(inputSize)
lstmLayer(numHiddenUnits,'OutputMode','last')
fullyConnectedLayer(numClasses)
softmaxLayer
classificationLayer];

```

指定训练选项。将 'adam' 指定为求解器，并将 'GradientThreshold' 指定为 1。将小批量大小设置为 27，并将最大训练轮数设置为 75。为确保数据存储创建的小批量的大小是 `trainNetwork` 函数所需的小批量化，还应将数据存储的小批量大小设置为相同的值。

由于小批量数据存储较小且序列较短，因此更适合在 CPU 上训练。将 'ExecutionEnvironment' 设置为 'cpu'。要在 GPU（如果可用）上进行训练，请将 'ExecutionEnvironment' 设置为 'auto'（默认值）。

```

miniBatchSize = 27;
options = trainingOptions('adam', ...
    'ExecutionEnvironment','cpu', ...
    'MaxEpochs',75, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',1, ...
    'Verbose',0, ...
    'Plots','training-progress');
dsTrain.MiniBatchSize = miniBatchSize;

```

使用指定的训练选项训练 LSTM 网络。

```
net = trainNetwork(dsTrain,layers,options);
```



## 测试网络

根据测试数据创建一个序列数据存储。

```
folderTest = fullfile(outputFolder,"Test");
dsTest = sequenceDatastore(folderTest);
```

对测试数据进行分类。指定与训练数据相同的小批量大小。为确保数据存储创建的小批量的大小是 `classify` 函数所需的大小，还应将数据存储的小批量大小设置为相同的值。

```
dsTest.MiniBatchSize = miniBatchSize;
YPred = classify(net,dsTest,'MiniBatchSize',miniBatchSize);
```

计算预测值的分类准确度。

```
YTest = dsTest.Labels;
acc = sum(YPred == YTest)./numel(YTest)
```

```
acc = 0.9432
```

## 参考

- [1] Kudo, M., J. Toyama, and M. Shimbo. "Multidimensional Curve Classification Using Passing-Through Regions." *Pattern Recognition Letters*. Vol. 20, No. 11–13, pp. 1103–1111.
- [2] Kudo, M., J. Toyama, and M. Shimbo. *Japanese Vowels Data Set*. <https://archive.ics.uci.edu/ml/datasets/Japanese+Vowels>

## 另请参阅

[trainNetwork](#) | [trainingOptions](#) | [lstmLayer](#) | [sequenceInputLayer](#)

## 相关示例

- “Develop Custom Mini-Batch Datastore”
- “使用深度学习进行时序预测” (第 4-9 页)
- “使用深度学习进行“序列到序列”分类” (第 4-33 页)
- “使用深度学习进行“序列到序列”回归” (第 4-38 页)
- “长短期记忆网络” (第 1-38 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 使用深度学习对无法放入内存的文本数据进行分类

此示例说明在深度学习网络中，如何使用变换后的数据存储对无法放入内存的文本数据进行分类。

变换后的数据存储对从基础数据存储读取的数据进行变换或处理。您可以使用变换后的数据存储作为深度学习应用的训练数据集、验证数据集、测试数据集以及预测数据集的数据源。使用变换后的数据存储可读取无法放入内存的数据，或者在读取批量数据时执行特定的预处理操作。

在训练网络时，软件通过填充、截断或拆分输入数据来创建长度相同的小批量序列。`trainingOptions` 函数提供了填充和截断输入序列的选项，但是，这些选项不太适合单词向量序列。此外，此函数不支持在自定义数据存储中填充数据。您必须改为手动填充和截断序列。如果对单词向量序列进行左填充和截断，训练效果可能会得到改善。

“Classify Text Data Using Deep Learning” (Text Analytics Toolbox) 示例手动将所有文档截断和填充到相同长度。此过程为非常短的文档添加了大量填充，同时也丢弃了非常长的文档中的大量数据。

取而代之，为了防止添加过多填充或丢弃过多数据，请创建一个变换后的数据存储，以将小批量数据输入到网络中。此示例中创建的转换后的数据存储将小批量文档转换为序列或单词索引，并对每个小批量进行左填充，使其长度等于小批量中最长文档的长度。

## 加载预训练的单词嵌入

数据存储需要使用单词嵌入以将文档转换为向量序列。使用 `fastTextWordEmbedding` 加载预训练的单词嵌入。此函数需要 Text Analytics Toolbox™ Model for fastText English 16 Billion Token Word Embedding 支持包。如果未安装此支持包，则函数会提供下载链接。

```
emb = fastTextWordEmbedding;
```

## 加载数据

根据 `factoryReports.csv` 中的数据创建一个表格文本数据存储。指定仅读取 "Description" 和 "Category" 列中的数据。

```
filenameTrain = "factoryReports.csv";
textName = "Description";
labelName = "Category";
ttdsTrain = tabularTextDatastore(filenameTrain,'SelectedVariableNames',[textName labelName]);
```

查看数据存储的预览。

```
preview(ttdsTrain)
```

`ans=8×2 table`

Description	Category
{'Items are occasionally getting stuck in the scanner spools.'	{'Mechanical Failure'}
{'Loud rattling and banging sounds are coming from assembler pistons.'}	{'Mechanical Failure'}
{'There are cuts to the power when starting the plant.'	{'Electronic Failure'}
{'Fried capacitors in the assembler.'	{'Electronic Failure'}
{'Mixer tripped the fuses.'	{'Electronic Failure'}
{'Burst pipe in the constructing agent is spraying coolant.'	{'Leak'}
{'A fuse is blown in the mixer.'	{'Electronic Failure'}
{'Things continue to tumble off of the belt.'	{'Mechanical Failure'}

{'Items are occasionally getting stuck in the scanner spools.'	{'Mechanical Failure'}
{'Loud rattling and banging sounds are coming from assembler pistons.'}	{'Mechanical Failure'}
{'There are cuts to the power when starting the plant.'	{'Electronic Failure'}
{'Fried capacitors in the assembler.'	{'Electronic Failure'}
{'Mixer tripped the fuses.'	{'Electronic Failure'}
{'Burst pipe in the constructing agent is spraying coolant.'	{'Leak'}
{'A fuse is blown in the mixer.'	{'Electronic Failure'}
{'Things continue to tumble off of the belt.'	{'Mechanical Failure'}

## 变换数据存储

创建一个自定义变换函数，该函数将从数据存储中读取的数据转换为包含预测变量和响应的表。

`transformText` 函数获取从 `tabularTextDatastore` 对象读取的数据，并返回包含预测变量和响应的表。预测变量是由单词嵌入 `emb` 给出的  $C \times S$  单词向量数组，其中  $C$  是嵌入维数， $S$  是序列长度。响应是类的分类标签。

要获取类名，请使用在示例末尾列出的 `readLabels` 函数从训练数据中读取标签，并找出唯一的类名。

```
labels = readLabels(ttdsTrain,labelName);
classNames = unique(labels);
numObservations = numel(labels);
```

由于表格文本数据存储可以在一次读取中读取多行数据，因此您可以在变换函数中处理一个完整的小批量数据。为了确保变换函数处理完整的小批量数据，请将表格文本数据存储的读取大小设置为将用于训练的小批量大小。

```
miniBatchSize = 64;
ttdsTrain.ReadSize = miniBatchSize;
```

要将表格文本数据的输出转换为训练序列，请使用 `transform` 函数变换数据存储。

```
tdsTrain = transform(ttdsTrain,@(data) transformText(data,emb,classNames))
```

```
tdsTrain =
  TransformedDatastore with properties:
```

```
UnderlyingDatastore: [1×1 matlab.io.datastore.TabularTextDatastore]
SupportedOutputFormats: {"txt" "csv" "xlsx" "xls" "parquet" "parq" "png" "jpg" "jpeg" "tif" "tiff" "v
  Transforms: {@(data)transformText(data,emb,classNames)}
IncludeInfo: 0
```

变换后的数据存储的预览。预测变量是  $C \times S$  数组，其中  $S$  是序列长度， $C$  是特征数（嵌入维数）。响应是分类标签。

```
preview(tdsTrain)
```

```
ans=8×2 table
  predictors      responses
  _____
  {300×11 single}  Mechanical Failure
  {300×11 single}  Mechanical Failure
  {300×11 single}  Electronic Failure
  {300×11 single}  Electronic Failure
  {300×11 single}  Electronic Failure
  {300×11 single}  Leak
  {300×11 single}  Electronic Failure
  {300×11 single}  Mechanical Failure
```

predictors	responses
{300×11 single}	Mechanical Failure
{300×11 single}	Mechanical Failure
{300×11 single}	Electronic Failure
{300×11 single}	Electronic Failure
{300×11 single}	Electronic Failure
{300×11 single}	Leak
{300×11 single}	Electronic Failure
{300×11 single}	Mechanical Failure

## 创建和训练 LSTM 网络

定义 LSTM 网络架构。要将序列数据输入到网络中，请包含一个序列输入层并将输入大小设置为嵌入维度。接下来，添加一个具有 180 个隐含单元的 LSTM 层。要将该 LSTM 层用于“序列到标签”分类问题，请将输出模式设置为 `'last'`。最后，添加一个输出大小等于类数的全连接层、一个 softmax 层和一个分类层。

```
numFeatures = emb.Dimension;
numHiddenUnits = 180;
numClasses = numel(classNames);
layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

指定训练选项。指定求解器为 'adam'，梯度阈值为 2。数据存储不支持乱序，因此请将 'Shuffle' 设置为 'never'。每轮训练后对网络进行一次验证。要监控训练进度，请将 'Plots' 选项设置为 'training-progress'。要隐藏详细输出，请将 'Verbose' 设置为 false。

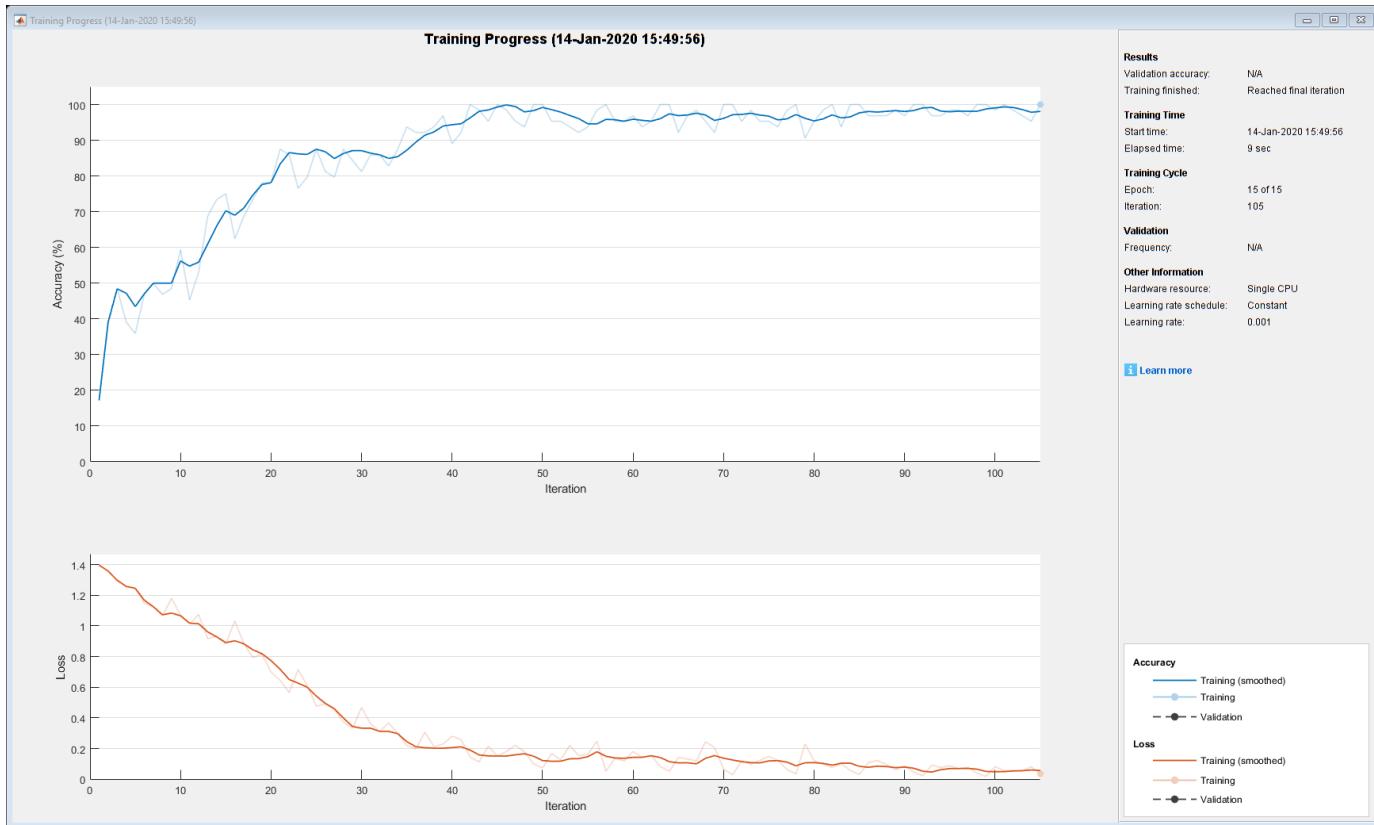
默认情况下，`trainNetwork` 使用 GPU（如果有）。要手动指定执行环境，请使用 `trainingOptions` 的 'ExecutionEnvironment' 名称-值对组参数。在 CPU 上进行训练所需的时间要明显长于在 GPU 上进行训练所需的时间。使用 GPU 进行训练需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关受支持设备的信息，请参阅 “GPU Support by Release” (Parallel Computing Toolbox)。

```
numIterationsPerEpoch = floor(numObservations / miniBatchSize);

options = trainingOptions('adam', ...
    'MaxEpochs',15, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',2, ...
    'Shuffle','never', ...
    'Plots','training-progress', ...
    'Verbose',false);
```

使用 `trainNetwork` 函数训练 LSTM 网络。

```
net = trainNetwork(tdsTrain,layers,options);
```



## 使用新数据进行预测

对三个新报告的事件类型进行分类。创建包含新报告的字符串数组。

```
reportsNew = [
    "Coolant is pooling underneath sorter."
    "Sorter blows fuses at start up."
    "There are some very loud rattling sounds coming from the assembler."];
```

使用与预处理训练文档相同的步骤来预处理文本数据。

```
documentsNew = preprocessText(reportsNew);
```

使用 `doc2sequence` 将文本数据转换为嵌入向量序列。

```
XNew = doc2sequence(emb,documentsNew);
```

使用经过训练的 LSTM 网络对新序列进行分类。

```
labelsNew = classify(net,XNew)
```

```
labelsNew = 3×1 categorical
```

```
Leak
```

```
Electronic Failure
```

```
Mechanical Failure
```

## 变换文本函数

`transformText` 函数获取从 `tabularTextDatastore` 对象读取的数据，并返回包含预测变量和响应的表。预测变量是由单词嵌入 `emb` 给出的  $C \times S$  单词向量数组，其中  $C$  是嵌入维数， $S$  是序列长度。这些响应是 `classNames` 中的类的分类标签。

```
function dataTransformed = transformText(data,emb,classNames)

% Preprocess documents.
textData = data{:,:1};
documents = preprocessText(textData);

% Convert to sequences.
predictors = doc2sequence(emb,documents);

% Read labels.
labels = data{:,2};
responses = categorical(labels,classNames);

% Convert data to table.
dataTransformed = table(predictors,responses);

end
```

## 预处理函数

函数 `preprocessText` 执行以下步骤：

- 1 使用 `tokenizedDocument` 对文本进行分词。
- 2 使用 `lower` 将文本转换为小写。
- 3 使用 `erasePunctuation` 删除标点符号。

```
function documents = preprocessText(textData)

documents = tokenizedDocument(textData);
documents = lower(documents);
documents = erasePunctuation(documents);

end
```

## 读取标签函数

`readLabels` 函数创建 `tabularTextDatastore` 对象 `ttds` 的一个副本，并读取 `labelName` 列中的标签。

```
function labels = readLabels(ttds,labelName)

ttdsNew = copy(ttds);
ttdsNew.SelectedVariableNames = labelName;
tbl = readall(ttdsNew);
labels = tbl.(labelName);

end
```

## 另请参阅

`fastTextWordEmbedding` | `wordEmbeddingLayer` | `doc2sequence` | `tokenizedDocument` | `lstmLayer` | `trainNetwork` | `trainingOptions` | `sequenceInputLayer` | `transform`

## 相关示例

- “使用深度学习进行序列分类” (第 4-2 页)
- “使用深度学习进行时序预测” (第 4-9 页)
- “长短期记忆网络” (第 1-38 页)
- “深度学习层列表” (第 1-18 页)
- “Deep Learning Tips and Tricks”

# 使用自定义小批量数据存储对无法放入内存的文本数据进行分类

此示例说明在深度学习网络中，如何使用自定义小批量数据存储对无法放入内存的文本数据进行分类。

小批量数据存储是支持批量读取数据的数据存储实现。您可以使用小批量数据存储作为深度学习应用程序的训练数据集、验证数据集、测试数据集以及预测数据集的源。使用小批量数据存储可读取无法放入内存的数据，或者在读取批量数据时执行特定的预处理操作。

在训练网络时，软件通过填充、截断或拆分输入数据来创建长度相同的小批量序列。`trainingOptions` 函数提供了填充和截断输入序列的选项，但是，这些选项不太适合单词向量序列。此外，此函数不支持在自定义数据存储中填充数据。您必须改为手动填充和截断序列。如果对单词向量序列进行左填充和截断，训练效果可能会得到改善。

“Classify Text Data Using Deep Learning” (Text Analytics Toolbox) 示例手动将所有文档截断和填充到相同长度。此过程为非常短的文档添加了大量填充，同时也丢弃了非常长的文档中的大量数据。

取而代之，为了防止添加过多填充或丢弃过多数据，请创建一个自定义小批量数据存储，以将小批量数据输入到网络中。自定义小批量数据存储 `textDatastore.m` 将小批量文档转换为序列或单词索引，并对每个小批量进行左填充，使其长度等于小批量中最长文档的长度。对于排序后的数据，此数据存储可以帮助减少添加到数据的填充量，因为文档并非填充到固定长度。同理，数据存储不会丢弃文档中的任何数据。

此示例使用自定义小批量数据存储 `textDatastore.m`。您可以通过自定义函数来调整此数据存储，使之适合您的数据。有关说明如何创建您自己的自定义小批量数据存储的示例，请参阅 “Develop Custom Mini-Batch Datastore”。

## 加载预训练的单词嵌入

数据存储 `textDatastore` 需要使用单词嵌入以将文档转换为向量序列。使用 `fastTextWordEmbedding` 加载预训练的单词嵌入。此函数需要 Text Analytics Toolbox™ Model for fastText English 16 Billion Token Word Embedding 支持包。如果未安装此支持包，则函数会提供下载链接。

```
emb = fastTextWordEmbedding;
```

## 创建文档的小批量数据存储

创建包含训练数据的数据存储。自定义小批量数据存储 `textDatastore` 从 CSV 文件中读取预测变量和标签。对于预测变量，数据存储将文档转换为单词索引序列，对于响应，数据存储返回每个文档的分类标签。

要创建数据存储，请先将自定义小批量数据存储 `textDatastore.m` 保存到路径。有关创建自定义小批量数据存储的详细信息，请参阅 “Develop Custom Mini-Batch Datastore”。

对于训练数据，指定 CSV 文件 "factoryReports.csv"，并且文本和标签分别位于 "Description" 和 "Category" 列中。

```
filenameTrain = "factoryReports.csv";
textName = "Description";
labelName = "Category";
dsTrain = textDatastore(filenameTrain, textName, labelName, emb)

dsTrain =
textDatastore with properties:

    ClassNames: ["Electronic Failure" "Leak" "Mechanical Failure" "Software Failure"]
```

```

Datastore: [1×1 matlab.io.datastore.TransformedDatastore]
EmbeddingDimension: 300
LabelName: "Category"
MiniBatchSize: 128
NumClasses: 4
NumObservations: 480

```

## 创建和训练 LSTM 网络

定义 LSTM 网络架构。要将序列数据输入到网络中，请包含一个序列输入层并将输入大小设置为嵌入维度。接下来，添加一个具有 180 个隐含单元的 LSTM 层。要将该 LSTM 层用于“序列到标签”分类问题，请将输出模式设置为 'last'。最后，添加一个输出大小等于类数的全连接层、一个 softmax 层和一个分类层。

```

numFeatures = dsTrain.EmbeddingDimension;
numHiddenUnits = 180;
numClasses = dsTrain.NumClasses;

layers = [ ...
    sequenceInputLayer(numFeatures)
    lstmLayer(numHiddenUnits,'OutputMode','last')
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];

```

指定训练选项。指定求解器为 'adam'，梯度阈值为 2。数据存储 `textDatastore.m` 不支持乱序，因此请将 'Shuffle' 设置为 'never'。有关说明如何实现支持乱序的数据存储的示例，请参阅“Develop Custom Mini-Batch Datastore”。要监控训练进度，请将 'Plots' 选项设置为 'training-progress'。要隐藏详细输出，请将 'Verbose' 设置为 `false`。

默认情况下，`trainNetwork` 使用 GPU（如果有）。要手动指定执行环境，请使用 `trainingOptions` 的 'ExecutionEnvironment' 名称-值对组参数。在 CPU 上进行训练所需的时间要明显长于在 GPU 上进行训练所需的时间。使用 GPU 进行训练需要 Parallel Computing Toolbox™ 和支持的 GPU 设备。有关支持的设备，请参阅“GPU Support by Release”（Parallel Computing Toolbox）。

```

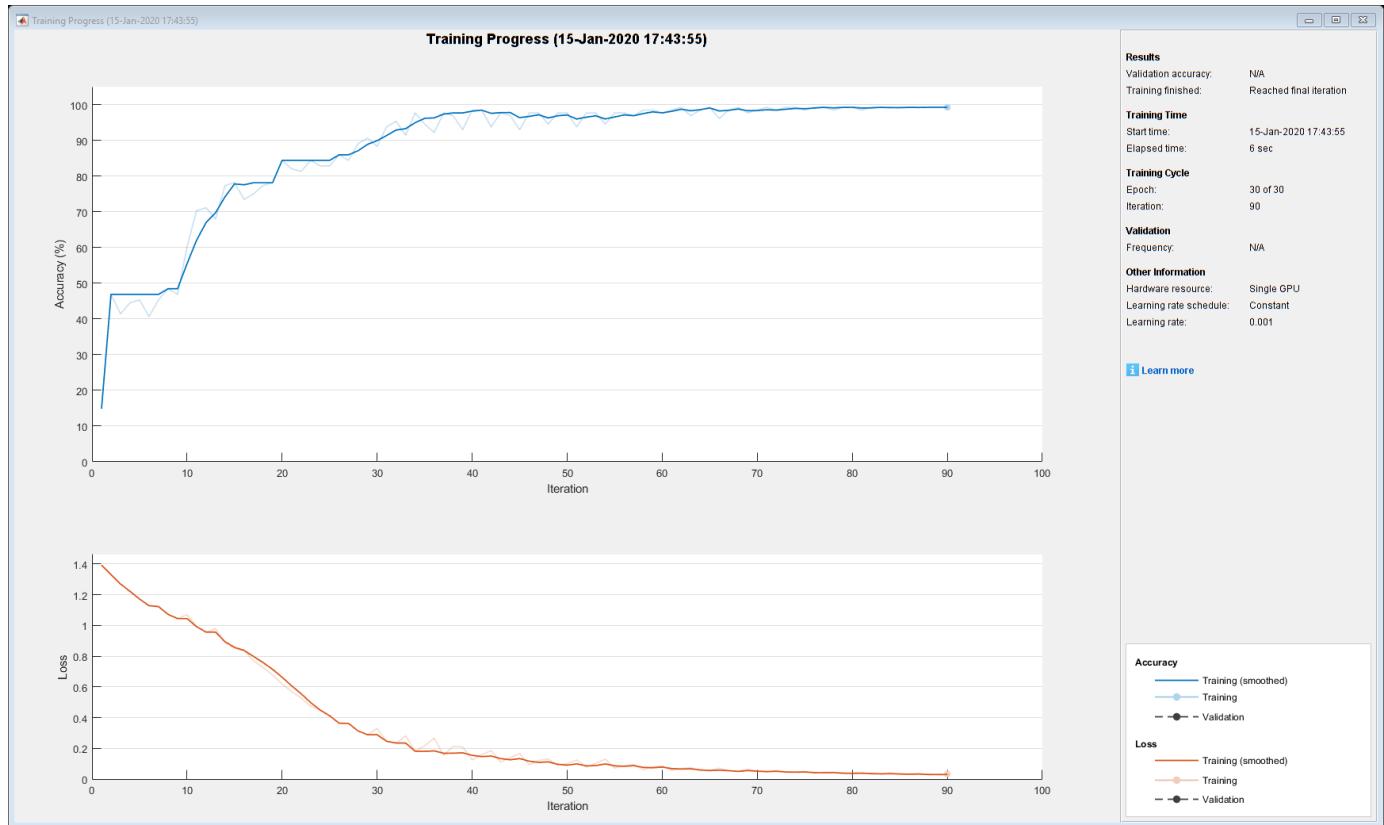
miniBatchSize = 128;
numObservations = dsTrain.NumObservations;
numIterationsPerEpoch = floor(numObservations / miniBatchSize);

options = trainingOptions('adam',...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThreshold',2, ...
    'Shuffle','never', ...
    'Plots','training-progress', ...
    'Verbose',false);

```

使用 `trainNetwork` 函数训练 LSTM 网络。

```
net = trainNetwork(dsTrain,layers,options);
```



## 使用新数据进行预测

对三个新报告的事件类型进行分类。创建包含新报告的字符串数组。

```
reportsNew = [
    "Coolant is pooling underneath sorter."
    "Sorter blows fuses at start up."
    "There are some very loud rattling sounds coming from the assembler."];
```

使用与预处理数据存储 `textDatastore.m` 相同的步骤来预处理文本数据。

```
documents = tokenizedDocument(reportsNew);
documents = lower(documents);
documents = erasePunctuation(documents);
predictors = doc2sequence(emb,documents);
```

使用经过训练的 LSTM 网络对新序列进行分类。

```
labelsNew = classify(net,predictors)
```

```
labelsNew = 3×1 categorical
    Leak
    Electronic Failure
```

Mechanical Failure

## 另请参阅

[wordEmbeddingLayer](#) | [doc2sequence](#) | [tokenizedDocument](#) | [lstmLayer](#) | [trainNetwork](#) | [trainingOptions](#) | [sequenceInputLayer](#) | [wordcloud](#) | [extractHTMLText](#) | [findElement](#) | [htmlTree](#)

## 相关示例

- “使用深度学习生成文本” (第 4-72 页)
- “Create Simple Text Model for Classification” (Text Analytics Toolbox)
- “Analyze Text Data Using Topic Models” (Text Analytics Toolbox)
- “Analyze Text Data Using Multiword Phrases” (Text Analytics Toolbox)
- “Train a Sentiment Classifier” (Text Analytics Toolbox)
- “使用深度学习进行序列分类” (第 4-2 页)
- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 深度学习代码生成

---

- “深度学习网络的代码生成” (第 19-2 页)
- “语义分割网络的代码生成” (第 19-10 页)
- “使用 GPU Coder 优化车道检测” (第 19-14 页)
- “为使用 YOLO v2 的目标检测生成代码” (第 19-24 页)
- “交通标志检测和识别” (第 19-27 页)
- “徽标识别网络” (第 19-35 页)
- “行人检测” (第 19-39 页)
- “去噪深度神经网络的代码生成” (第 19-45 页)
- “训练和部署用于语义分割的全卷积网络” (第 19-49 页)
- “使用 U-Net 的语义分割网络的代码生成” (第 19-60 页)
- “ARM 目标上的深度学习代码生成” (第 19-67 页)
- “通过 ARM 计算使用 codegen 进行深度学习预测” (第 19-71 页)
- “针对不同批量大小在 Intel 目标上进行深度学习代码生成” (第 19-76 页)

## 深度学习网络的代码生成

此示例说明如何为使用深度学习的图像分类应用程序执行代码生成。它使用 `codegen` 命令生成一个 MEX 函数，该函数使用图像分类网络（如 MobileNet-v2、ResNet 和 GoogLeNet）运行预测。

### 第三方前提条件

#### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

#### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅“Third-Party Hardware”（GPU Coder）和“Setting Up the Prerequisite Products”（GPU Coder）。

### 验证 GPU 环境

使用 `coder.checkGpuInstall`（GPU Coder）函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

### mobilenetv2\_predict 入口函数

MobileNet-v2 是一个卷积神经网络，基于来自 ImageNet 数据库的超过一百万个图像进行训练。该网络有 155 层深，可以将图像分类至 1000 个目标类别（例如键盘、鼠标、铅笔和多种动物）。该网络的图像输入大小为  $224 \times 224$ 。使用 `analyzeNetwork` 函数显示深度学习网络架构的交互式可视化。

```
net = mobilenetv2();
analyzeNetwork(net);
```

**mobilenetv2\_predict.m** 入口函数以图像作为输入，并使用预训练的 MobileNet-v2 卷积神经网络对图像运行预测。该函数使用持久性对象 `mynet` 加载串行网络对象，并在后续调用中重用该持久性对象进行预测。

```
type('mobilenetv2_predict.m')
```

```
% Copyright 2017-2019 The MathWorks, Inc.
```

```
function out = mobilenetv2_predict(in)
 %#codegen

 persistent mynet;

 if isempty(mynet)
```

```

mynet = coder.loadDeepLearningNetwork('mobilenetv2','mobilenetv2');
end

% pass in input
out = mynet.predict(in);

```

## 运行 MEX 代码生成

要为 `mobilenetv2_predict` 入口函数生成 CUDA 代码, 请为 MEX 目标创建一个 GPU 代码配置对象, 并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象, 并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。运行 `codegen` 命令, 并指定输入大小为 [224,224,3]。该值对应于 MobileNet-v2 网络的输入层大小。

```

cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg mobilenetv2_predict -args {ones(224,224,3)} -report

```

Code generation successful: To view the report, open('codegen/mex/mobilenetv2\_predict/html/report.mldatx').

## 生成的代码说明

串行网络生成为一个 C++ 类, 其中包含由 155 个层类组成的数组以及用于设置、调用预测和清理网络的函数。

```

class b_mobilenetv2_0
{
    ...
public:
    b_mobilenetv2_0();
    void setup();
    void predict();
    void cleanup();
    ~b_mobilenetv2_0();
};

```

该类的 `setup()` 方法会设置句柄并为网络对象的每层分配内存。`predict()` 方法会对网络中的 155 个层逐层执行预测。

生成的代码文件 `mobilenetv2_predict.cu` 中的入口函数 `mobilenetv2_predict()` 构造 `b_mobilenetv2` 类类型的静态对象, 并对该网络对象调用 `setup` 和 `predict`。

```

static b_mobilenetv2_0 mynet;
static boolean_T mynet_not_empty;

/* Function Definitions */
void mobilenetv2_predict(const real_T in[150528], real32_T out[1000])
{
    if (!mynet_not_empty) {
        DeepLearningNetwork_setup(&mynet);
        mynet_not_empty = true;
    }

    /* pass in input */
    DeepLearningNetwork_predict(&mynet, in, out);
}

```

系统将为具有参数的层（如网络中的全连接层和卷积层）导出相应的二进制文件。例如，文件 `cnn_mobilenetv2_conv*_w` 和 `cnn_mobilenetv2_conv*_b` 对应于网络中卷积层的权重和偏置参数。要查看生成的文件列表，请使用：

```
dir(fullfile(pwd, 'codegen', 'mex', 'mobilenetv2_predict'))
```

### 运行生成的 MEX

加载输入图像。

```
im = imread('peppers.png');  
imshow(im);
```



对输入图像调用 `mobilenetv2_predict_mex`。

```
im = imresize(im, [224,224]);  
predict_scores = mobilenetv2_predict_mex(double(im));
```

获得排名前五的预测分数及其标签。

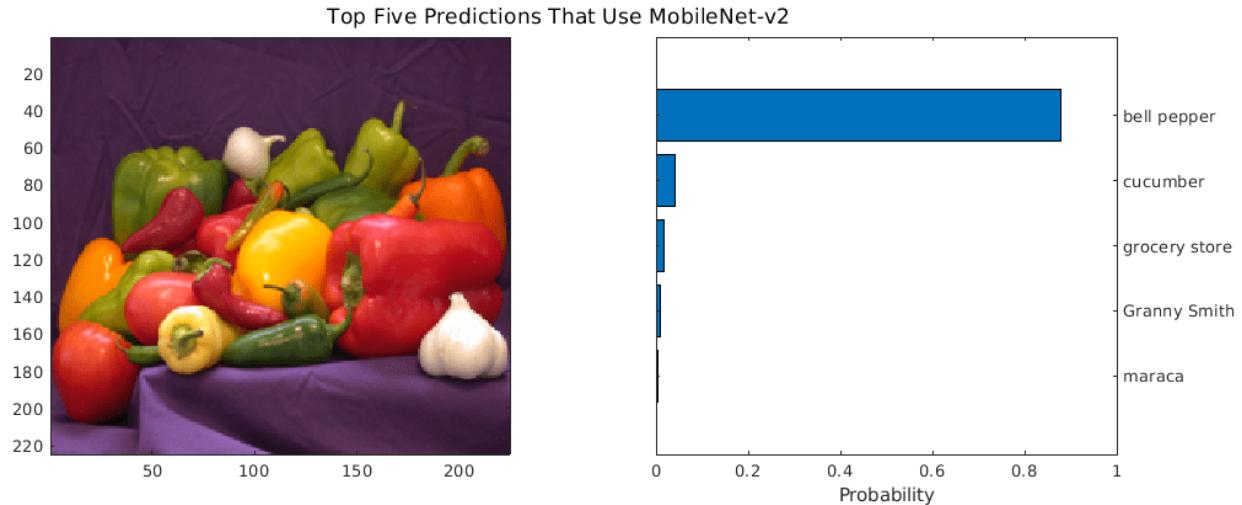
```
[scores,indx] = sort(predict_scores, 'descend');  
classNames = net.Layers(end).ClassNames;  
classNamesTop = classNames(indx(1:5));  
  
h = figure;
```

```

h.Position(3) = 2*h.Position(3);
ax1 = subplot(1,2,1);
ax2 = subplot(1,2,2);

image(ax1,im);
barh(ax2,scores(5:-1:1))
xlabel(ax2,'Probability')
yticklabels(ax2,classNamesTop(5:-1:1))
ax2.YAxisLocation = 'right';
sgtitle('Top Five Predictions That Use MobileNet-v2')

```



## 视频的分类

所包含的辅助函数 `mobilenet_live.m` 从网络摄像头抓取帧，执行预测，并在每个捕获的视频帧上显示分类结果。此示例使用由 MATLAB® Support Package for USB Webcams™ 支持的 `webcam` (MATLAB Support Package for USB Webcams) 函数。您可以通过支持包安装程序下载并安装支持包。

```

type('mobilenet_live.m')

% Copyright 2017-2019 The MathWorks, Inc.

function mobilenet_live

% Connect to a camera
camera = webcam;

% The labels with top 5 prediction scores are
% mapped to corresponding labels
net = mobilenetv2();
classnames = net.Layers(end).ClassNames;

imfull = zeros(224,400,3, 'uint8');

fps = 0;

ax = axes;

while true

```

```
% Take a picture
ipicture = camera.snapshot;

% Resize and cast the picture to single
picture = imresize(ipicture,[224,224]);

% Call MEX function for MobileNet-v2 prediction
tic;
pout = mobilenetv2_predict(single(picture));
newt = toc;

% fps
fps = .9*fps + .1*(1/newt);

% top 5 scores
[top5labels, scores] = getTopFive(pout,classnames);

% display
if isvalid(ax)
    dispResults(ax, imfull, picture, top5labels, scores, fps);
else
    break;
end
end

function dispResults(ax, imfull, picture, top5labels, scores, fps)
for k = 1:3
    imfull(:,177:end,k) = picture(:,:,k);
end

h = imshow(imfull, 'InitialMagnification',200, 'Parent', ax);
scol = 1;
srow = 20;
text(get(h, 'Parent'), scol, srow, sprintf('MobileNet-v2 Demo'), 'color', 'w', 'FontSize', 20);
srow = srow + 20;

text(get(h, 'Parent'), scol, srow, sprintf('Fps = %2.2f', fps), 'color', 'w', 'FontSize', 15);
srow = srow + 20;
for k = 1:5
    t = text(get(h, 'Parent'), scol, srow, top5labels{k}, 'color', 'w', 'FontSize', 15);
    pos = get(t, 'Extent');
    text(get(h, 'Parent'), pos(1)+pos(3)+5, srow, sprintf("%2.2f%%", scores(k)), 'color', 'w', 'FontSize', 15);
    srow = srow + 20;
end

drawnow;
end

function [labels, scores] = getTopFive(predictOut,classnames)
[val,indx] = sort(predictOut, 'descend');
scores = val(1:5)*100;
labels = classnames(indx(1:5));
end
```

清除内存中已加载的静态网络对象。

```
clear mex;
```

## 使用 ResNet-50 网络进行图像分类

您也可以使用 DAG 网络 ResNet-50 进行图像分类。Deep Learning Toolbox 的 ResNet-50 支持包提供了适用于 MATLAB 的预训练 ResNet-50 模型。要下载并安装该支持包，请使用附加功能资源管理器。要了解有关查找和安装附加功能的详细信息，请参阅“[获取和管理附加功能](#)”。

```
net = resnet50;
disp(net)
```

DAGNetwork with properties:

```
Layers: [177×1 nnet.cnn.layer.Layer]
Connections: [192×2 table]
InputNames: {'input_1'}
OutputNames: {'ClassificationLayer_fc1000'}
```

## 运行 MEX 代码生成

要为 `resnet_predict.m` 入口函数生成 CUDA 代码，请为 MEX 目标创建一个 GPU 代码配置对象，并将目标语言设置为 C++。此入口函数调用 `resnet50` 函数来加载网络并对输入图像执行预测。

```
cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg resnet_predict -args {ones(224,224,3)} -report
```

Code generation successful: To view the report, open('codegen/mex/resnet\_predict/html/report.mldatx').

对输入图像调用 `resnet_predict_mex`。

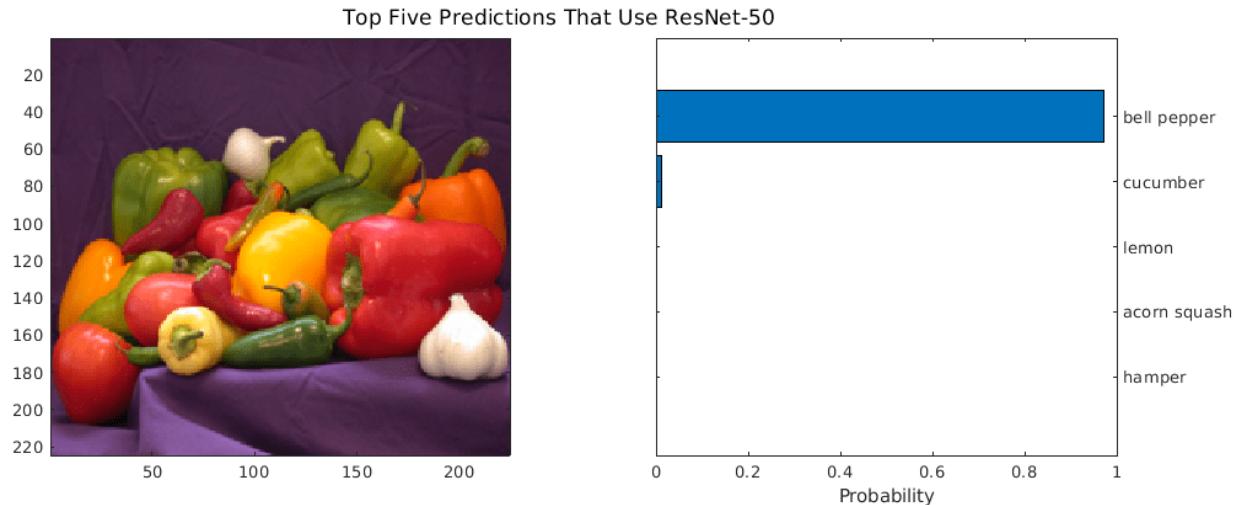
```
predict_scores = resnet_predict_mex(double(im));
```

获得排名前五的预测分数及其标签。

```
[scores,indx] = sort(predict_scores, 'descend');
classNames = net.Layers(end).ClassNames;
classNamesTop = classNames(indx(1:5));

h = figure;
h.Position(3) = 2*h.Position(3);
ax1 = subplot(1,2,1);
ax2 = subplot(1,2,2);

image(ax1,im);
barh(ax2,scores(5:-1:1))
xlabel(ax2,'Probability')
yticklabels(ax2,classNamesTop(5:-1:1))
ax2.YAxisLocation = 'right';
sgtitle("Top Five Predictions That Use ResNet-50")
```



清除内存中已加载的静态网络对象。

```
clear mex;
```

### 使用 GoogLeNet (Inception) 网络进行图像分类

Deep Learning Toolbox 的 GoogLeNet 支持包提供了适用于 MATLAB 的预训练 GoogLeNet 模型。要下载并安装该支持包，请使用附加功能资源管理器。要了解有关查找和安装附加功能的详细信息，请参阅“[获取和管理附加功能](#)”。

```
net = googlenet;
disp(net)
```

DAGNetwork with properties:

```
Layers: [144×1 nnet.cnn.layer.Layer]
Connections: [170×2 table]
InputNames: {'data'}
OutputNames: {'output'}
```

### 运行 MEX 代码生成

为 `googlenet_predict.m` 入口函数生成 CUDA 代码。此入口函数调用 `googlenet` 函数来加载网络并对输入图像执行预测。要从此入口函数生成代码，请为 MEX 目标创建一个 GPU 配置对象。

```
cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg googlenet_predict -args {ones(224,224,3)} -report
```

Code generation successful: To view the report, open('codegen/mex/googlenet\_predict/html/report.mldatx').

对输入图像调用 `googlenet_predict_mex`。

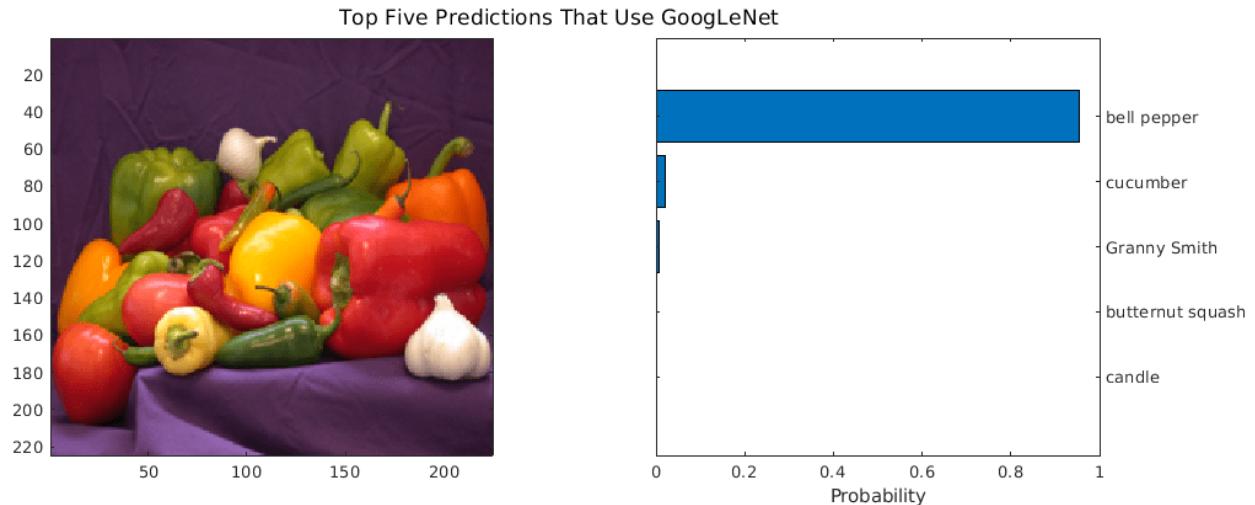
```
im = imresize(im, [224,224]);
predict_scores = googlenet_predict_mex(double(im));
```

获得排名前五的预测分数及其标签。

```
[scores,idx] = sort(predict_scores, 'descend');
classNames = net.Layers(end).ClassNames;
classNamesTop = classNames(idx(1:5));

h = figure;
h.Position(3) = 2*h.Position(3);
ax1 = subplot(1,2,1);
ax2 = subplot(1,2,2);

image(ax1,im);
barh(ax2,scores(5:-1:1))
xlabel(ax2,'Probability')
yticklabels(ax2,classNamesTop(5:-1:1))
ax2.YAxisLocation = 'right';
sgtitle('Top Five Predictions That Use GoogLeNet')
```



清除内存中已加载的静态网络对象。

```
clear mex;
```

## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习”（第 1-2 页）

## 语义分割网络的代码生成

此示例说明如何为使用深度学习的图像分割应用程序生成代码。它使用 `codegen` 命令生成一个基于 SegNet [1] 的 DAG 网络对象执行预测的 MEX 函数。SegNet 是一种用于图像分割的深度学习网络。

### 第三方前提条件

#### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

#### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅“Third-Party Hardware”(GPU Coder)和“Setting Up the Prerequisite Products”(GPU Coder)。

### 验证 GPU 环境

使用 `coder.checkGpuInstall`(GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

### 分割网络

SegNet [1] 是一种卷积神经网络(CNN)，专为语义图像分割而设计。它是针对 CamVid [2] 数据集而训练并导入到 MATLAB® 中进行推断的深度编解码器多类像素级分割网络。SegNet [1] 经过训练，可以分割属于 11 个类的像素，包括天空、建筑物、电线杆、道路、人行道、树、标志符号、围栏、汽车、行人和骑车人。

有关在 MATLAB 中使用 CamVid [2] 数据集训练语义分割网络的信息，请参阅“Semantic Segmentation Using Deep Learning”(Computer Vision Toolbox)。

### segnet\_predict 入口函数

`segnet_predict.m` 入口函数以图像作为输入，并使用保存在 `SegNet.mat` 文件中的深度学习网络对图像执行预测。该函数将 `SegNet.mat` 文件中的网络对象加载到持久变量 `mynet` 中，并在后续的预测调用中重用该持久变量。

```
type('segnet_predict.m')
```

```
function out = segnet_predict(in)
%#codegen
% Copyright 2018-2019 The MathWorks, Inc.

persistent mynet;
```

```

if isempty(mynet)
    mynet = coder.loadDeepLearningNetwork('SegNet.mat');
end

% pass in input
out = predict(mynet,in);

```

### 获取预训练的 SegNet DAG 网络对象

```

net = getSegNet();

Downloading pretrained SegNet (107 MB)...

```

DAG 网络包含 91 个层，包括卷积层、批量归一化层、池化层、去池化层和像素分类输出层。使用 `analyzeNetwork` 函数显示深度学习网络架构的交互式可视化。

```
analyzeNetwork(net);
```

### 运行 MEX 代码生成

要为 `segnet_predict.m` 入口函数生成 CUDA 代码，请为 MEX 目标创建一个 GPU 代码配置对象，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。运行 `codegen` 命令以指定输入大小为 [360,480,3]。该值对应于 SegNet 的输入层大小。

```

cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg segnet_predict -args {ones(360,480,3,'uint8')} -report

```

Code generation successful: To view the report, open('codegen/mex/segnet\_predict/html/report.mldatx').

### 运行生成的 MEX

加载并显示输入图像。对输入图像调用 `segnet_predict_mex`。

```

im = imread('gpcoder_segnet_image.png');
imshow(im);
predict_scores = segnet_predict_mex(im);

```



`predict_scores` 变量是一个三维矩阵，它具有 11 个通道，分别对应于每个类的像素级预测分数。使用最高预测分数计算通道以获得像素级标签。

```
[~,argmax] = max(predict_scores,[],3);
```

在输入图像上叠加分割标签并显示分割区域。

```
classes = [  
    "Sky"  
    "Building"  
    "Pole"  
    "Road"  
    "Pavement"  
    "Tree"  
    "SignSymbol"  
    "Fence"  
    "Car"  
    "Pedestrian"  
    "Bicyclist"  
];  
  
cmap = camvidColorMap();  
SegmentedImage = labeloverlay(im,argmax,'ColorMap',cmap);  
figure  
imshow(SegmentedImage);  
pixelLabelColorbar(cmap,classes);
```



## 参考资料

[1] Badrinarayanan, Vijay, Alex Kendall, and Roberto Cipolla."SegNet:A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." arXiv preprint arXiv:1511.00561, 2015.

[2] Brostow, Gabriel J., Julien Fauqueur, and Roberto Cipolla."Semantic object classes in video:A high-definition ground truth database."Pattern Recognition Letters Vol 30, Issue 2, 2009, pp 88-97.

## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “深度学习在计算机视觉领域的应用”

## 使用 GPU Coder 优化车道检测

此示例说明如何从表示为 `SeriesNetwork` 对象的深度学习网络生成 CUDA® 代码。此示例中的串行网络是一个卷积神经网络，可以从图像中检测并输出车道标记边界。

### 前提条件

- 支持 CUDA 的 NVIDIA® GPU。
- NVIDIA CUDA 工具包和驱动程序。
- NVIDIA cuDNN 库。
- OpenCV 库，用于视频读取和图像显示操作。
- 编译器和库的环境变量。有关支持的编译器和库的版本的信息，请参阅“Third-Party Hardware”(GPU Coder)。有关设置环境变量的信息，请参阅“Setting Up the Prerequisite Products”(GPU Coder)。

### 验证 GPU 环境

使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

### 获得预训练的 SeriesNetwork

```
[laneNet, coeffMeans, coeffStds] = getLaneDetectionNetworkGPU();
```

该网络将图像作为输入并输出两个车道边界，分别对应于自我意识车辆的左右车道。每个车道边界都由抛物线方程  $y = ax^2 + bx + c$  表示，其中  $y$  是横向偏移， $x$  是与车辆的纵向距离。该网络为每个车道输出三个参数  $a$ 、 $b$  和  $c$ 。网络架构类似于 AlexNet，但是最后几层会替换为较小的全连接层和回归输出层。

`laneNet.Layers`

`ans =`

23×1 Layer array with layers:

1 'data'	Image Input	227×227×3 images with 'zerocenter' normalization
2 'conv1'	Convolution	96 11×11×3 convolutions with stride [4 4] and padding [0 0 0 0]
3 'relu1'	ReLU	ReLU
4 'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5 'pool1'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 0 0 0]
6 'conv2'	Convolution	256 5×5×48 convolutions with stride [1 1] and padding [2 2 2 2]
7 'relu2'	ReLU	ReLU
8 'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9 'pool2'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 0 0 0]
10 'conv3'	Convolution	384 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
11 'relu3'	ReLU	ReLU
12 'conv4'	Convolution	384 3×3×192 convolutions with stride [1 1] and padding [1 1 1 1]
13 'relu4'	ReLU	ReLU
14 'conv5'	Convolution	256 3×3×192 convolutions with stride [1 1] and padding [1 1 1 1]
15 'relu5'	ReLU	ReLU

16 'pool5'      Max Pooling 17 'fc6'        Fully Connected 18 'relu6'       ReLU 19 'drop6'       Dropout 20 'fcLane1'     Fully Connected 21 'fcLane1Relu' ReLU 22 'fcLane2'     Fully Connected 23 'output'      Regression Output	3×3 max pooling with stride [2 2] and padding [0 0 0 0] 4096 fully connected layer ReLU 50% dropout 16 fully connected layer ReLU 6 fully connected layer mean-squared-error with 'leftLane_a', 'leftLane_b', and 4 other respo
--	--

## 检查主要入口函数

**type detect\_lane.m**

```

function [laneFound, ltPts, rtPts] = detect_lane(frame, laneCoeffMeans, laneCoeffStds)
% From the networks output, compute left and right lane points in the
% image coordinates. The camera coordinates are described by the caltech
% mono camera model.

%#codegen

% A persistent object mynet is used to load the series network object.
% At the first call to this function, the persistent object is constructed and
% setup. When the function is called subsequent times, the same object is reused
% to call predict on inputs, thus avoiding reconstructing and reloading the
% network object.
persistent lanenet;

if isempty(lanenet)
    lanenet = coder.loadDeepLearningNetwork('laneNet.mat', 'lanenet');
end

lanecoeffsNetworkOutput = lanenet.predict(permute(frame, [2 1 3]));

% Recover original coeffs by reversing the normalization steps

params = lanecoeffsNetworkOutput .* laneCoeffStds + laneCoeffMeans;

isRightLaneFound = abs(params(6)) > 0.5; %c should be more than 0.5 for it to be a right lane
isLeftLaneFound = abs(params(3)) > 0.5;

vehicleXPoints = 3:30; %meters, ahead of the sensor
ltPts = coder.nullcopy(zeros(28,2,'single'));
rtPts = coder.nullcopy(zeros(28,2,'single'));

if isRightLaneFound && isLeftLaneFound
    rtBoundary = params(4:6);
    rt_y = computeBoundaryModel(rtBoundary, vehicleXPoints);
    ltBoundary = params(1:3);
    lt_y = computeBoundaryModel(ltBoundary, vehicleXPoints);

    % Visualize lane boundaries of the ego vehicle
    tform = get_tformToImage;
    % map vehicle to image coordinates
    ltPts = tform.transformPointsInverse([vehicleXPoints', lt_y']); %'
    rtPts = tform.transformPointsInverse([vehicleXPoints', rt_y']); %'
    laneFound = true;
else

```

```

    laneFound = false;
end

end

function yWorld = computeBoundaryModel(model, xWorld)
    yWorld = polyval(model, xWorld);
end

function tform = get_tformToImage
% Compute extrinsics based on camera setup
yaw = 0;
pitch = 14; % pitch of the camera in degrees
roll = 0;

translation = translationVector(yaw, pitch, roll);
rotation = rotationMatrix(yaw, pitch, roll);

% Construct a camera matrix
focalLength = [309.4362, 344.2161];
principalPoint = [318.9034, 257.5352];
Skew = 0;

camMatrix = [rotation; translation] * intrinsicMatrix(focalLength, ...
    Skew, principalPoint);

% Turn camMatrix into 2-D homography
tform2D = [camMatrix(1,:); camMatrix(2,:); camMatrix(4,:)]; % drop Z

tform = projective2d(tform2D);
tform = tform.invert();
end

function translation = translationVector(yaw, pitch, roll)
SensorLocation = [0 0];
Height = 2.1798; % mounting height in meters from the ground
rotationMatrix = (...  

    rotZ(yaw)*... % last rotation  

    rotX(90-pitch)*...  

    rotZ(roll)... % first rotation  

);

% Adjust for the SensorLocation by adding a translation
sl = SensorLocation;

translationInWorldUnits = [sl(2), sl(1), Height];
translation = translationInWorldUnits*rotationMatrix;
end

%-----
% Rotation around X-axis
function R = rotX(a)
a = deg2rad(a);
R = [...
    1 0      0;  

    0 cos(a) -sin(a);  

    0 sin(a)  cos(a)];

```

```

end

%-----
% Rotation around Y-axis
function R = rotY(a)
a = deg2rad(a);
R = [...
    cos(a) 0 sin(a);
    0 1 0;
    -sin(a) 0 cos(a)];
end

%-----
% Rotation around Z-axis
function R = rotZ(a)
a = deg2rad(a);
R = [...
    cos(a) -sin(a) 0;
    sin(a) cos(a) 0;
    0 0 1];
end

%-----
% Given the Yaw, Pitch, and Roll, determine the appropriate Euler
% angles and the sequence in which they are applied to
% align the camera's coordinate system with the vehicle coordinate
% system. The resulting matrix is a Rotation matrix that together
% with the Translation vector defines the extrinsic parameters of the camera.
function rotation = rotationMatrix(yaw, pitch, roll)

rotation = (...%
    rotY(180)*... % last rotation: point Z up
    rotZ(-90)*... % X-Y swap
    rotZ(yaw)*... % point the camera forward
    rotX(90-pitch)*... % "un-pitch"
    rotZ(roll)... % 1st rotation: "un-roll"
);
end

function intrinsicMat = intrinsicMatrix(FocalLength, Skew, PrincipalPoint)
intrinsicMat = ...
    [FocalLength(1) , 0 , 0; ...
    Skew , FocalLength(2) , 0; ...
    PrincipalPoint(1), PrincipalPoint(2), 1];
end

```

### 生成网络代码和后处理代码

网络会计算参数 a、b 和 c，这些参数描述了左右车道边界的抛物线方程。

根据这些参数，计算与车道位置对应的 x 和 y 坐标。这些坐标必须映射到图像坐标。函数 `detect_lane.m` 执行所有这些计算。为 'lib' 目标创建一个 GPU 代码配置对象，从而为该函数生成 CUDA 代码，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。运行 `codegen` 命令。

```

cfg = coder.gpuConfig('lib');
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
cfg.GenerateReport = true;
cfg.TargetLang = 'C++';
codegen -args {ones(227,227,3,'single'),ones(1,6,'double'),ones(1,6,'double')} -config cfg detect_lane

```

Code generation successful: To view the report, open('codegen/lib/detect\_lane/html/report.mldatx').

### 生成的代码说明

串行网络生成为一个 C++ 类，其中包含由 23 个层类组成的数组。

```

class c_lanenet {
public:
    int32_T batchSize; int32_T numLayers; real32_T *inputData; real32_T
    *outputData; MWCNNLayer *layers[23];
public:
    c_lanenet(void); void setup(void); void predict(void); void
    cleanup(void); ~c_lanenet(void);
};

```

该类的 `setup()` 方法会设置句柄并为每个层对象分配内存。`predict()` 方法会针对网络中 23 个层的每个层调用预测。

`cnn_lanenet_conv*_w` 和 `cnn_lanenet_conv*_b` 文件是网络中卷积层的二进制权重和偏置文件。  
`cnn_lanenet_fc*_w` 和 `cnn_lanenet_fc*_b` 文件是网络中全连接层的二进制权重和偏置文件。

```

codegendir = fullfile('codegen', 'lib', 'detect_lane');
dir(codegendir)

```

.	cnn_lanenet0_0_conv4_w.bin
..	cnn_lanenet0_0_conv5_b.bin
.gitignore	cnn_lanenet0_0_conv5_w.bin
DeepLearningNetwork.cu	cnn_lanenet0_0_data_offset.bin
DeepLearningNetwork.h	cnn_lanenet0_0_data_scale.bin
DeepLearningNetwork.o	cnn_lanenet0_0_fc6_b.bin
MWCNNLayerImpl.cu	cnn_lanenet0_0_fc6_w.bin
MWCNNLayerImpl.hpp	cnn_lanenet0_0_fcLane1_b.bin
MWCNNLayerImpl.o	cnn_lanenet0_0_fcLane1_w.bin
MWCudaDimUtility.cu	cnn_lanenet0_0_fcLane2_b.bin
MWCudaDimUtility.hpp	cnn_lanenet0_0_fcLane2_w.bin
MWCustomLayerForCuDNN.cpp	cnn_lanenet0_0_responseNames.txt
MWCustomLayerForCuDNN.hpp	codeInfo.mat
MWCustomLayerForCuDNN.o	codedescriptor.dmr
MWElementwiseAffineLayer.cpp	compileInfo.mat
MWElementwiseAffineLayer.hpp	defines.txt
MWElementwiseAffineLayer.o	detect_lane.a
MWElementwiseAffineLayerImpl.cu	detect_lane.cu
MWElementwiseAffineLayerImpl.hpp	detect_lane.h
MWElementwiseAffineLayerImpl.o	detect_lane.o
MWElementwiseAffineLayerImplKernel.cu	detect_lane_data.cu
MWElementwiseAffineLayerImplKernel.o	detect_lane_data.h
MWFusedConvReLUlayer.cpp	detect_lane_data.o
MWFusedConvReLUlayer.hpp	detect_lane_initialize.cu
MWFusedConvReLUlayer.o	detect_lane_initialize.h
MWFusedConvReLUlayerImpl.cu	detect_lane_initialize.o
MWFusedConvReLUlayerImpl.hpp	detect_lane_ref.rsp

```

MWFusedConvReLUlayerImpl.o      detect_lane_rtw.mk
MWKernelHeaders.hpp            detect_lane_terminate.cu
MWTargetNetworkImpl.cu          detect_lane_terminate.h
MWTargetNetworkImpl.hpp         detect_lane_terminate.o
MWTargetNetworkImpl.o          detect_lane_types.h
buildInfo.mat                  examples
cnn_api.cpp                    gpu_codegen_info.mat
cnn_api.hpp                    html
cnn_api.o                     interface
cnn_lanenet0_0_conv1_b.bin     mean.bin
cnn_lanenet0_0_conv1_w.bin     predict.cu
cnn_lanenet0_0_conv2_b.bin     predict.h
cnn_lanenet0_0_conv2_w.bin     predict.o
cnn_lanenet0_0_conv3_b.bin     rtw_proj.tmw
cnn_lanenet0_0_conv3_w.bin     rtwtypes.h
cnn_lanenet0_0_conv4_b.bin

```

### 生成附加文件以对输出进行后处理

从经过训练的网络中导出均值和标准差，以便在执行过程中使用。

```

codegendir = fullfile(pwd, 'codegen', 'lib','detect_lane');
fid = fopen(fullfile(codegendir,'mean.bin'), 'w');
A = [coeffMeans coeffStds];
fwrite(fid, A, 'double');
fclose(fid);

```

### 主文件

使用主文件编译网络代码。主文件使用 OpenCV VideoCapture 方法从输入视频中读取帧。对每个帧都进行处理和分类，直到不再读取帧为止。在显示每个帧的输出之前，使用 `detect_lane.cu` 中生成的 `detect_lane` 函数对输出进行后处理。

type `main_lanenet.cu`

```

/* Copyright 2016 The MathWorks, Inc. */

#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/core/types.hpp>
#include <opencv2/highgui.hpp>
#include <list>
#include <cmath>
#include "detect_lane.h"

using namespace cv;
void readData(float *input, Mat& orig, Mat & im)
{
    Size size(227,227);
    resize(orig,im,size,0,0,INTER_LINEAR);
    for(int j=0;j<227*227;j++)
    {

```

```
//BGR to RGB
    input[2*227*227+j]=(float)(im.data[j*3+0]);
    input[1*227*227+j]=(float)(im.data[j*3+1]);
    input[0*227*227+j]=(float)(im.data[j*3+2]);
}
}

void addLane(float pts[28][2], Mat & im, int numPts)
{
    std::vector<Point2f> iArray;
    for(int k=0; k<numPts; k++)
    {
        iArray.push_back(Point2f(pts[k][0],pts[k][1]));
    }
    Mat curve(iArray, true);
    curve.convertTo(curve, CV_32S); //adapt type for polylines
    polylines(im, curve, false, CV_RGB(255,255,0), 2, LINE_AA);
}

void writeData(float *outputBuffer, Mat & im, int N, double means[6], double stds[6])
{
    // get lane coordinates
    boolean_T laneFound = 0;
    float ltPts[56];
    float rtPts[56];
    detect_lane(outputBuffer, means, stds, &laneFound, ltPts, rtPts);

    if (!laneFound)
    {
        return;
    }

    float ltPtsM[28][2];
    float rtPtsM[28][2];
    for(int k=0; k<28; k++)
    {
        ltPtsM[k][0] = ltPts[k];
        ltPtsM[k][1] = ltPts[k+28];
        rtPtsM[k][0] = rtPts[k];
        rtPtsM[k][1] = rtPts[k+28];
    }

    addLane(ltPtsM, im, 28);
    addLane(rtPtsM, im, 28);
}

void readMeanAndStds(const char* filename, double means[6], double stds[6])
{
    FILE* pFile = fopen(filename, "rb");
    if (pFile==NULL)
    {
        fputs ("File error",stderr);
        return;
    }

    // obtain file size
    fseek (pFile , 0 , SEEK_END);
```

```

long lSize = ftell(pFile);
rewind(pFile);

double* buffer = (double*)malloc(lSize);

size_t result = fread(buffer,sizeof(double),lSize,pFile);
if (result*sizeof(double) != lSize) {
    fputs ("Reading error",stderr);
    return;
}

for (int k = 0 ; k < 6; k++)
{
    means[k] = buffer[k];
    stds[k] = buffer[k+6];
}
free(buffer);
}

// Main function
int main(int argc, char* argv[])
{
    float *inputBuffer = (float*)calloc(sizeof(float),227*227*3);
    float *outputBuffer = (float*)calloc(sizeof(float),6);

    if ((inputBuffer == NULL) || (outputBuffer == NULL)) {
        printf("ERROR: Input/Output buffers could not be allocated!\n");
        exit(-1);
    }

    // get ground truth mean and std
    double means[6];
    double stds[6];
    readMeanAndStds("mean.bin", means, stds);

    if (argc < 2)
    {
        printf("Pass in input video file name as argument\n");
        return -1;
    }

    VideoCapture cap(argv[1]);
    if (!cap.isOpened())
    {
        printf("Could not open the video capture device.\n");
        return -1;
    }

    cudaEvent_t start, stop;
    float fps = 0;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    Mat orig, im;
    namedWindow("Lane detection demo",WINDOW_NORMAL);
    while(true)
    {
        cudaEventRecord(start);

```

```
cap >> orig;
if (orig.empty()) break;
readData(inputBuffer, orig, im);

writeData(inputBuffer, orig, 6, means, stds);

cudaEventRecord(stop);
cudaEventSynchronize(stop);

char strbuf[50];
float milliseconds = -1.0;
cudaEventElapsedTime(&milliseconds, start, stop);
fps = fps*.9+1000.0/milliseconds*.1;
sprintf (strbuf, "%.2f FPS", fps);
putText(orig, strbuf, Point(200,30), FONT_HERSHEY_DUPLEX, 1, CV_RGB(0,0,0), 2);
imshow("Lane detection demo", orig);
if (waitKey(50)%256 == 27) break; // stop capturing by pressing ESC  */
}

destroyWindow("Lane detection demo");

free(inputBuffer);
free(outputBuffer);

return 0;
}
```

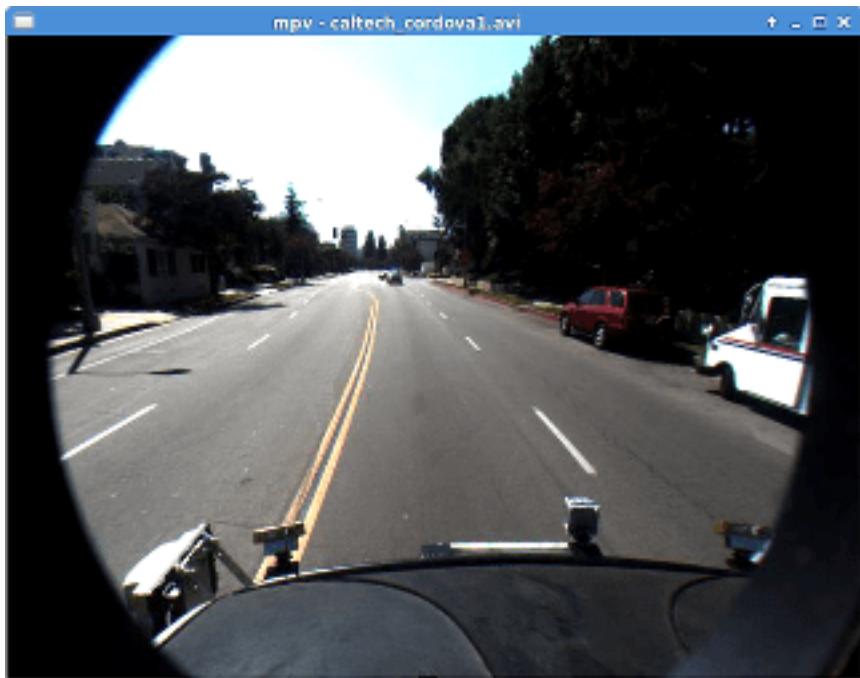
### 下载示例视频

```
if ~exist('./caltech_cordova1.avi', 'file')
url = 'https://www.mathworks.com/supportfiles/gpucoder/media/caltech_cordova1.avi';
websave('caltech_cordova1.avi', url);
end
```

### 编译可执行文件

```
if ispc
setenv('MATLAB_ROOT', matlabroot);
vcvarsall = mex.getCompilerConfigurations('C++').Details.CommandLineShell;
setenv('VCVARSALL', vcvarsall);
system('make_win_lane_detection.bat');
cd(codegendir);
system('lanenet.exe ..\..\..\caltech_cordova1.avi');
else
setenv('MATLAB_ROOT', matlabroot);
system('make -f Makefile_lane_detection.mk');
cd(codegendir);
system('./lanenet ../../caltech_cordova1.avi');
end
```

输入截图



输出截图



## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习”（第 1-2 页）
- “深度学习在计算机视觉领域的应用”

## 为使用 YOLO v2 的目标检测生成代码

此示例说明如何为 you only look once (YOLO) v2 目标检测器生成 CUDA® MEX。YOLO v2 目标检测网络由两个子网络组成。一个特征提取网络，后跟一个检测网络。此示例为 Computer Vision Toolbox™ 的使用 YOLO v2 深度学习进行目标检测示例中训练的网络生成代码。有关详细信息，请参阅“Object Detection Using YOLO v2 Deep Learning”(Computer Vision Toolbox)。您可以修改此示例，以便为从 Computer Vision Toolbox™ 中的导入预训练 ONNX YOLO v2 目标检测器示例导入的网络生成 CUDA® MEX。有关详细信息，请参阅“Import Pretrained ONNX YOLO v2 Object Detector”(Computer Vision Toolbox)。

### 第三方前提条件

#### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

#### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅“Third-Party Hardware”(GPU Coder)和“Setting Up the Prerequisite Products”(GPU Coder)。

### 验证 GPU 环境

使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

### 获取预训练的 DAGNetwork

```
net = getYOLOv2();
```

DAG 网络包含 150 层，包括卷积层、ReLU 层和批量归一化层，以及 YOLO v2 变换层和 YOLO v2 输出层。要以交互可视方式呈现深度学习网络架构，请使用 `analyzeNetwork` 函数。

```
analyzeNetwork(net);
```

### yolov2\_detect 入口函数

`yolov2_detect.m` 入口函数以图像作为输入，并使用保存在 `yolov2ResNet50VehicleExample.mat` 文件中的深度学习网络对图像运行检测器。该函数将 `yolov2ResNet50VehicleExample.mat` 文件中的网络对象加载到持久变量 `yolov2Obj` 中，并在后续的检测调用中重用该持久性对象。

```
type('yolov2_detect.m')
```

```
function outImg = yolov2_detect(in)
```

```
% Copyright 2018-2019 The MathWorks, Inc.

persistent yolov2Obj;

if isempty(yolov2Obj)
    yolov2Obj = coder.loadDeepLearningNetwork('yolov2ResNet50VehicleExample.mat');
end

% pass in input
[bboxes,~,labels] = yolov2Obj.detect(in,'Threshold',0.5);

% convert categorical labels to cell array of character vectors for MATLAB
% execution
if coder.target('MATLAB')
    labels = cellstr(labels);
end

% Annotate detections in the image.
outImg = insertObjectAnnotation(in,'rectangle',bboxes,labels);
```

## 运行 MEX 代码生成

要为 `yolov2_detect.m` 入口函数生成 CUDA 代码，请为 MEX 目标创建一个 GPU 代码配置对象，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。运行 `codegen` 命令以指定输入大小为 [224,224,3]。该值对应于 YOLOv2 的输入层大小。

```
cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg yolov2_detect -args {ones(224,224,3,'uint8')} -report
```

Code generation successful: To view the report, open('codegen/mex/yolov2\_detect/html/report.mldatx').

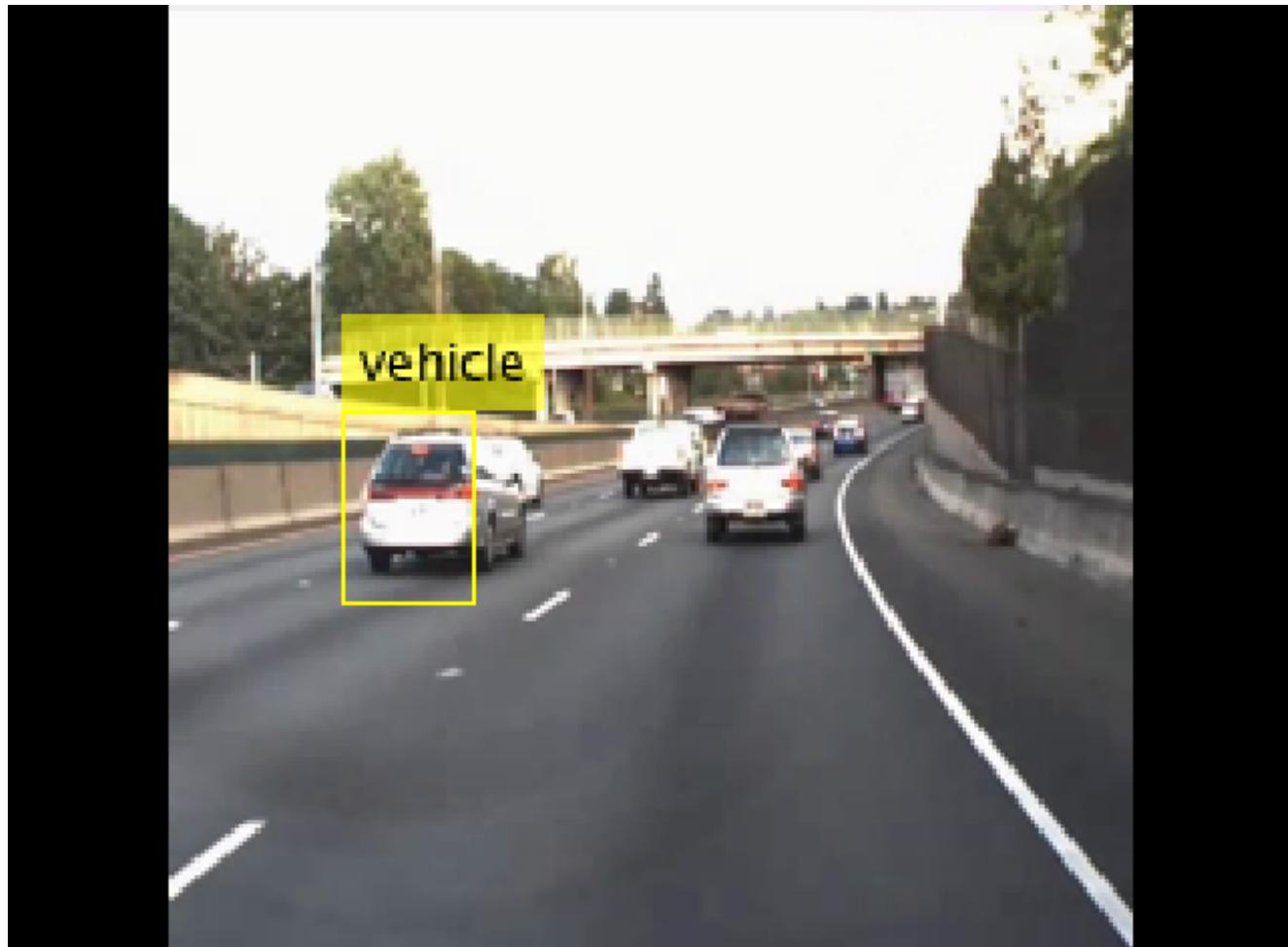
## 运行生成的 MEX

设置视频文件读取器并读取输入视频。创建视频播放器来显示视频和输出检测。

```
videoFile = 'highway_lanechange.mp4';
videoFreader = vision.VideoFileReader(videoFile,'VideoOutputDataType','uint8');
depVideoPlayer = vision.DeployableVideoPlayer('Size','Custom','CustomSize',[640 480]);
```

逐帧读取视频输入，并使用检测器检测视频中的车辆。

```
cont = ~isDone(videoFreader);
while cont
    I = step(videoFreader);
    in = imresize(I,[224,224]);
    out = yolov2_detect_mex(in);
    step(depVideoPlayer, out);
    cont = ~isDone(videoFreader) && isOpen(depVideoPlayer); % Exit the loop if the video player figure window is closed
end
```

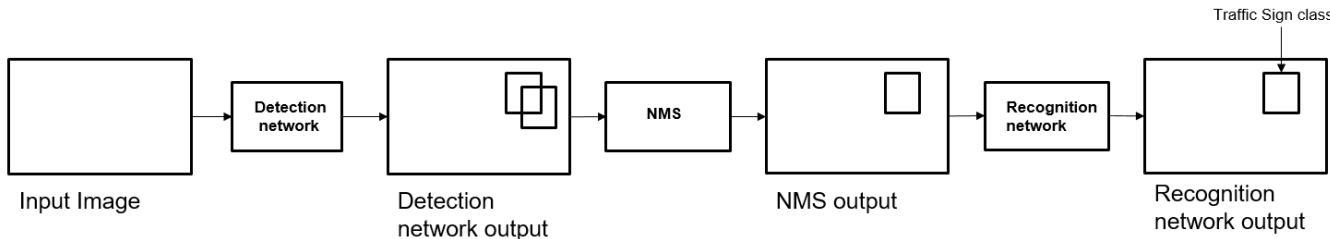


### 参考资料

[1] Redmon, Joseph, and Ali Farhadi."YOLO9000:Better, Faster, Stronger."2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).IEEE, 2017.

# 交通标志检测和识别

此示例说明如何为使用深度学习的交通标志检测和识别应用程序生成 CUDA® MEX 代码。交通标志检测和识别是驾驶辅助系统的重要应用，可辅助并向驾驶员提供有关道路标志的信息。



在此交通标志检测和识别示例中，执行三个步骤 - 检测、非极大值抑制 (NMS) 和识别。首先，该示例使用 You Only Look Once (YOLO) 网络的目标检测网络变体检测输入图像上的交通标志。然后，使用 NMS 算法抑制重叠检测。最后，识别网络对检测到的交通标志进行分类。

## 第三方前提条件

### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅 “Third-Party Hardware” (GPU Coder) 和 “Setting Up the Prerequisite Products” (GPU Coder)。

## 验证 GPU 环境

使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```

envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
  
```

## 检测和识别网络

检测网络在 Darknet 框架中进行训练，并被导入到 MATLAB® 中进行推断。由于交通标志相对于图像而言较小，并且训练数据中每个类的训练样本数量较少，因此在训练检测网络时，所有交通标志都被视为单个类。

检测网络将输入图像划分为一个  $7 \times 7$  网格。每个网格单元都会检测到一个交通标志，前提是该交通标志的中心在该网格单元内。每个单元预测两个边界框以及这两个边界框的置信度分数。置信度分数指示该框是否包含目标。每个单元预测在栅格单元内找到交通标志的概率。最终分数是先前分数的乘积。对此最终得分应用 0.2 的阈值来选择检测值。

使用 MATLAB 基于相同的图像训练识别网络。

`trainRecognitionnet.m` 辅助脚本显示识别网络训练。

## 获得预训练的 SeriesNetwork

下载检测网络和识别网络。

```
getTsdr();
```

检测网络包含 58 个层，包括卷积层、泄漏 ReLU 层和全连接层。

```
load('yolo_tsr.mat');
yolo.Layers
```

```
ans =
```

```
58x1 Layer array with layers:
```

1 'input'	Image Input	448×448×3 images
2 'conv1'	Convolution	64 7×7×3 convolutions with stride [2 2] and padding [3 3 3]
3 'relu1'	Leaky ReLU	Leaky ReLU with scale 0.1
4 'pool1'	Max Pooling	2×2 max pooling with stride [2 2] and padding [0 0 0 0]
5 'conv2'	Convolution	192 3×3×64 convolutions with stride [1 1] and padding [1 1 1 1]
6 'relu2'	Leaky ReLU	Leaky ReLU with scale 0.1
7 'pool2'	Max Pooling	2×2 max pooling with stride [2 2] and padding [0 0 0 0]
8 'conv3'	Convolution	128 1×1×192 convolutions with stride [1 1] and padding [0 0 0 0]
9 'relu3'	Leaky ReLU	Leaky ReLU with scale 0.1
10 'conv4'	Convolution	256 3×3×128 convolutions with stride [1 1] and padding [1 1 1 1]
11 'relu4'	Leaky ReLU	Leaky ReLU with scale 0.1
12 'conv5'	Convolution	256 1×1×256 convolutions with stride [1 1] and padding [0 0 0 0]
13 'relu5'	Leaky ReLU	Leaky ReLU with scale 0.1
14 'conv6'	Convolution	512 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
15 'relu6'	Leaky ReLU	Leaky ReLU with scale 0.1
16 'pool6'	Max Pooling	2×2 max pooling with stride [2 2] and padding [0 0 0 0]
17 'conv7'	Convolution	256 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
18 'relu7'	Leaky ReLU	Leaky ReLU with scale 0.1
19 'conv8'	Convolution	512 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
20 'relu8'	Leaky ReLU	Leaky ReLU with scale 0.1
21 'conv9'	Convolution	256 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
22 'relu9'	Leaky ReLU	Leaky ReLU with scale 0.1
23 'conv10'	Convolution	512 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
24 'relu10'	Leaky ReLU	Leaky ReLU with scale 0.1
25 'conv11'	Convolution	256 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
26 'relu11'	Leaky ReLU	Leaky ReLU with scale 0.1
27 'conv12'	Convolution	512 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]

```

28 'relu12'    Leaky ReLU      Leaky ReLU with scale 0.1
29 'conv13'     Convolution   256 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
30 'relu13'     Leaky ReLU      Leaky ReLU with scale 0.1
31 'conv14'     Convolution   512 3×3×256 convolutions with stride [1 1] and padding [1 1 1 1]
32 'relu14'     Leaky ReLU      Leaky ReLU with scale 0.1
33 'conv15'     Convolution   512 1×1×512 convolutions with stride [1 1] and padding [0 0 0 0]
34 'relu15'     Leaky ReLU      Leaky ReLU with scale 0.1
35 'conv16'     Convolution   1024 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]
36 'relu16'     Leaky ReLU      Leaky ReLU with scale 0.1
37 'pool16'     Max Pooling   2×2 max pooling with stride [2 2] and padding [0 0 0 0]
38 'conv17'     Convolution   512 1×1×1024 convolutions with stride [1 1] and padding [0 0 0 0]
39 'relu17'     Leaky ReLU      Leaky ReLU with scale 0.1
40 'conv18'     Convolution   1024 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]
41 'relu18'     Leaky ReLU      Leaky ReLU with scale 0.1
42 'conv19'     Convolution   512 1×1×1024 convolutions with stride [1 1] and padding [0 0 0 0]
43 'relu19'     Leaky ReLU      Leaky ReLU with scale 0.1
44 'conv20'     Convolution   1024 3×3×512 convolutions with stride [1 1] and padding [1 1 1 1]
45 'relu20'     Leaky ReLU      Leaky ReLU with scale 0.1
46 'conv21'     Convolution   1024 3×3×1024 convolutions with stride [1 1] and padding [1 1 1 1]
47 'relu21'     Leaky ReLU      Leaky ReLU with scale 0.1
48 'conv22'     Convolution   1024 3×3×1024 convolutions with stride [2 2] and padding [1 1 1 1]
49 'relu22'     Leaky ReLU      Leaky ReLU with scale 0.1
50 'conv23'     Convolution   1024 3×3×1024 convolutions with stride [1 1] and padding [1 1 1 1]
51 'relu23'     Leaky ReLU      Leaky ReLU with scale 0.1
52 'conv24'     Convolution   1024 3×3×1024 convolutions with stride [1 1] and padding [1 1 1 1]
53 'relu24'     Leaky ReLU      Leaky ReLU with scale 0.1
54 'fc25'       Fully Connected 4096 fully connected layer
55 'relu25'     Leaky ReLU      Leaky ReLU with scale 0.1
56 'fc26'       Fully Connected 539 fully connected layer
57 'softmax'    Softmax        softmax
58 'classoutput' Classification Output crossentropyex with '1' and 538 other classes

```

识别网络包含 14 个层，包括卷积层、全连接层和分类输出层。

```

load('RecognitionNet.mat');
convnet.Layers

```

ans =

14×1 Layer array with layers:

```

1 'imageinput'  Image Input      48×48×3 images with 'zerocenter' normalization and 'randflplr' augmentation
2 'conv_1'       Convolution    100 7×7×3 convolutions with stride [1 1] and padding [0 0 0 0]
3 'relu_1'       ReLU           ReLU
4 'maxpool_1'   Max Pooling   2×2 max pooling with stride [2 2] and padding [0 0 0 0]
5 'conv_2'       Convolution   150 4×4×100 convolutions with stride [1 1] and padding [0 0 0 0]
6 'relu_2'       ReLU           ReLU
7 'maxpool_2'   Max Pooling   2×2 max pooling with stride [2 2] and padding [0 0 0 0]
8 'conv_3'       Convolution   250 4×4×150 convolutions with stride [1 1] and padding [0 0 0 0]
9 'maxpool_3'   Max Pooling   2×2 max pooling with stride [2 2] and padding [0 0 0 0]
10 'fc_1'        Fully Connected 300 fully connected layer
11 'dropout'     Dropout        90% dropout
12 'fc_2'        Fully Connected 35 fully connected layer
13 'softmax'     Softmax        softmax
14 'classoutput' Classification Output crossentropyex with '0' and 34 other classes

```

**tsdr\_predict 入口函数**

`tsdr_predict.m` 入口函数以图像作为输入，并使用检测网络检测图像中的交通标志。该函数使用 `selectStrongestBbox` 抑制重叠检测 (NMS)，并使用识别网络识别交通标志。该函数将 `yolo_tsrm.mat` 中的网络对象加载到持久变量 `detectionnet` 中，并将 `RecognitionNet.mat` 中的网络对象加载到持久变量 `recognitionnet` 中。该函数在后续调用中将重用这些持久性对象。

```
type('tsdr_predict.m')

function [selectedBbox,idx] = tsdr_predict(img)
%#codegen

% This function detects the traffic signs in the image using Detection Network
% (modified version of Yolo) and recognizes(classifies) using Recognition Network
%
% Inputs :
%
% im      : Input test image
%
% Outputs :
%
% selectedBbox : Detected bounding boxes
% idx       : Corresponding classes

% Copyright 2017-2019 The MathWorks, Inc.

coder.gpu.kernelfun;

% resize the image
img_rz = imresize(img,[448,448]);

% Converting into BGR format
img_rz = img_rz(:,:,3:-1:1);
img_rz = im2single(img_rz);

%% TSD
persistent detectionnet;
if isempty(detectionnet)
    detectionnet = coder.loadDeepLearningNetwork('yolo_tsrm.mat','Detection');
end

predictions = detectionnet.activations(img_rz,56,'OutputAs','channels');

%% Convert predictions to bounding box attributes
classes = 1;
num = 2;
side = 7;
thresh = 0.2;
[h,w,~] = size(img);

boxes = single(zeros(0,4));
probs = single(zeros(0,1));
for i = 0:(side*side)-1
    for n = 0:num-1
        p_index = side*side*classes + i*num + n + 1;
        % Process prediction here
    end
end
```

```

scale = predictions(p_index);
prob = zeros(1,classes+1);
for j = 0:classes
    class_index = i*classes + 1;
    tempProb = scale*predictions(class_index+j);
    if tempProb > thresh

        row = floor(i / side);
        col = mod(i,side);

        box_index = side*side*(classes + num) + (i*num + n)*4 + 1;
        bxX = (predictions(box_index + 0) + col) / side;
        bxY = (predictions(box_index + 1) + row) / side;

        bxW = (predictions(box_index + 2)^2);
        bxH = (predictions(box_index + 3)^2);

        prob(j+1) = tempProb;
        probs = [probs,tempProb];

        boxX = (bxX-bxW/2)*w+1;
        boxY = (bxY-bxH/2)*h+1;
        boxW = bxW*w;
        boxH = bxH*h;
        boxes = [boxes; boxX,boxY,boxW,boxH];
    end
end
end
end

%% Run Non-Maximal Suppression on the detected bounding boxess
coder.varsize('selectedBbox',[98, 4],[1 0]);
[selectedBbox,~] = selectStrongestBbox(round(boxes),probs);

%% Recognition

persistent recognitionnet;
if isempty(recognitionnet)
    recognitionnet = coder.loadDeepLearningNetwork('RecognitionNet.mat','Recognition');
end

idx = zeros(size(selectedBbox,1),1);
inpImg = coder.nullcopy(zeros(48,48,3,size(selectedBbox,1)));
for i = 1:size(selectedBbox,1)

    ymin = selectedBbox(i,2);
    ymax = ymin+selectedBbox(i,4);
    xmin = selectedBbox(i,1);
    xmax = xmin+selectedBbox(i,3);

    % Resize Image
    inpImg(:,:,:i) = imresize(img(ymin:ymax,xmin:xmax,:),[48,48]);
end

for i = 1:size(selectedBbox,1)
    output = recognitionnet.predict(inpImg(:,:,:i));

```

```
[~,idx(i)]=max(output);  
end
```

### 为 tsdr\_predict 函数生成 CUDA MEX

为 MEX 目标创建一个 GPU 配置对象，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。要生成 CUDA MEX，请使用 `codegen` 命令并指定输入大小为 [480,704,3]。该值对应于 `tsdr_predict` 函数的输入图像大小。

```
cfg = coder.gpuConfig('mex');  
cfg.TargetLang = 'C++';  
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');  
codegen -config cfg tsdr_predict -args {ones(480,704,3,'uint8')} -report
```

Code generation successful: To view the report, open('codegen/mex/tsdr\_predict/html/report.mldatx').

要使用 TensorRT 生成代码，请将 `coder.DeepLearningConfig('tensorrt')` 作为选项传递给编码器配置对象来代替 'cudnn'。

### 运行生成的 MEX

加载输入图像。

```
im = imread('stop.jpg');  
imshow(im);
```



对输入图像调用 `tsdr_predict_mex`。

```
im = imresize(im, [480,704]);
[bboxes,classes] = tsdr_predict_mex(im);
```

将类编号映射到类字典中的交通标志名称。

```
classNames = {'addedLane','slow','dip','speedLimit25','speedLimit35','speedLimit40','speedLimit45',...
'speedLimit50','speedLimit55','speedLimit65','speedLimitUrdbl','doNotPass','intersection',...
'keepRight','laneEnds','merge','noLeftTurn','noRightTurn','stop','pedestrianCrossing',...
'stopAhead','rampSpeedAdvisory20','rampSpeedAdvisory45','truckSpeedLimit55',...
'rampSpeedAdvisory50','turnLeft','rampSpeedAdvisoryUrdbl','turnRight','rightLaneMustTurn',...
'yield','yieldAhead','school','schoolSpeedLimit25','zoneAhead45','signalAhead'};
```

```
classRec = classNames(classes);
```

显示检测到的交通标志。

```
outputImage = insertShape(im,'Rectangle',bboxes,'LineWidth',3);
```

```
for i = 1:size(bboxes,1)
    outputImage = insertText(outputImage,[bboxes(i,1)+bboxes(i,3) bboxes(i,2)-20],classRec{i}, 'FontSize',20, 'TextColor','red');
end
```

```
imshow(outputImage);
```



### 对视频进行交通标志检测和识别

所包含的辅助文件 `tsdr_testVideo.m` 从测试视频中抓取帧，执行交通标志检测和识别，并绘制测试视频的每个帧的结果。

```
% Input video
v = VideoReader('stop.avi');
fps = 0;

while hasFrame(v)
    % Take a frame
    picture = readFrame(v);
    picture = imresize(picture,[920,1632]);
    % Call MEX function for Traffic Sign Detection and Recognition
    tic;
    [bboxes,clases] = tsdr_predict_mex(picture);
    newt = toc;

    % fps
    fps = .9*fps + .1*(1/newt);

    % display
    displayDetections(picture,bboxes,clases,fps);
end
```

清除已加载到内存中的静态网络对象。

```
clear mex;
```

### 另请参阅

#### 相关示例

- “在 MATLAB 中进行深度学习”（第 1-2 页）
- “深度学习在计算机视觉领域的应用”

# 徽标识别网络

此示例说明如何为使用深度学习的徽标分类应用程序生成代码。它使用 **codegen** 命令生成 MEX 函数，该函数对名为 LogoNet 的 **SeriesNetwork** 对象执行预测。

## 第三方前提条件

### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅“Third-Party Hardware”(GPU Coder)和“Setting Up the Prerequisite Products”(GPU Coder)。

## 验证 GPU 环境

使用 **coder.checkGpuInstall** (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

## 徽标识别网络

徽标帮助用户进行品牌辨识和识别。许多公司在广告、文档材料和促销活动中使用其徽标。徽标识别网络 (logonet) 是在 MATLAB® 中开发的，可以在各种光照条件下和相机移动下识别 32 个徽标。由于该网络仅专注于识别，因此可以在不需要本地化的应用程序中使用。

## 训练网络

该网络是在 MATLAB 中进行训练的，所使用的训练数据中每种徽标大约有 200 个图像。由于用来训练网络的图像数量很少，因此使用数据增强来增加训练样本数。使用四种类型的数据增强：对比度归一化、高斯模糊、随机翻转和剪切。此数据增强有助于识别在不同光照条件下和镜头角度下捕获的图像中的徽标。logonet 的输入大小为 [227 227 3]。标准 SGDM 使用 0.0001 的学习率和小批量大小 45 进行 40 轮训练。**trainLogonet.m** 辅助脚本演示了针对示例图像的数据增强、logonet 的架构以及训练选项。

## 获得预训练的 SeriesNetwork

下载 logonet 网络并将其保存到 **LogoNet.mat**。

```
getLogonet();
```

保存的网络包含 22 个层，包括卷积层、全连接层和分类输出层。

```
load('LogoNet.mat');
convnet.Layers
```

```

ans =
22×1 Layer array with layers:

1 'imageinput'  Image Input      227×227×3 images with 'zerocenter' normalization and 'randfliplr' augmentation
2 'conv_1'       Convolution     96 5×5×3 convolutions with stride [1 1] and padding [0 0 0 0]
3 'relu_1'       ReLU           ReLU
4 'maxpool_1'    Max Pooling   3×3 max pooling with stride [2 2] and padding [0 0 0 0]
5 'conv_2'       Convolution     128 3×3×96 convolutions with stride [1 1] and padding [0 0 0 0]
6 'relu_2'       ReLU           ReLU
7 'maxpool_2'    Max Pooling   3×3 max pooling with stride [2 2] and padding [0 0 0 0]
8 'conv_3'       Convolution     384 3×3×128 convolutions with stride [1 1] and padding [0 0 0 0]
9 'relu_3'       ReLU           ReLU
10 'maxpool_3'   Max Pooling   3×3 max pooling with stride [2 2] and padding [0 0 0 0]
11 'conv_4'       Convolution     128 3×3×384 convolutions with stride [2 2] and padding [0 0 0 0]
12 'relu_4'       ReLU           ReLU
13 'maxpool_4'   Max Pooling   3×3 max pooling with stride [2 2] and padding [0 0 0 0]
14 'fc_1'        Fully Connected 2048 fully connected layer
15 'relu_5'       ReLU           ReLU
16 'dropout_1'    Dropout        50% dropout
17 'fc_2'        Fully Connected 2048 fully connected layer
18 'relu_6'       ReLU           ReLU
19 'dropout_2'    Dropout        50% dropout
20 'fc_3'        Fully Connected 32 fully connected layer
21 'softmax'     Softmax         softmax
22 'classoutput' Classification Output crossentropyex with 'adidas' and 31 other classes

```

### logonet\_predict 入口函数

`logonet_predict.m` 入口函数以图像作为输入，并使用保存在 `LogoNet.mat` 文件中的深度学习网络对图像执行预测。该函数将 `LogoNet.mat` 中的网络对象加载到持久变量 `logonet` 中，并在后续的预测调用中重用该持久变量。

```
type(logonet_predict.m)
```

```

function out = logonet_predict(in)
 %#codegen

% Copyright 2017-2019 The MathWorks, Inc.

persistent logonet;

if isempty(logonet)

    logonet = coder.loadDeepLearningNetwork('LogoNet.mat','logonet');
end

out = logonet.predict(in);

end

```

### 为 `logonet_predict` 函数生成 CUDA MEX

为 MEX 目标创建一个 GPU 配置对象，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的

**DeepLearningConfig** 属性。要生成 CUDA MEX，请使用 **codegen** 命令并指定输入大小为 [227,227,3]。该值对应于 logonet 网络的输入层大小。

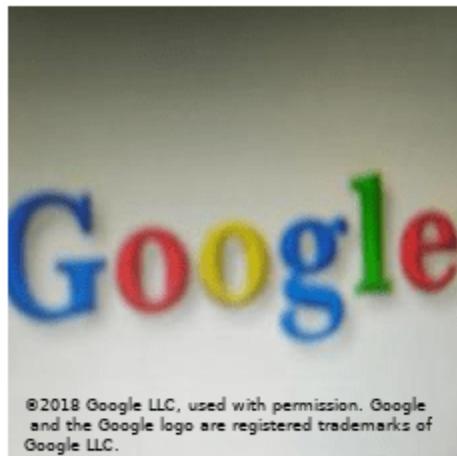
```
cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg logonet_predict -args {ones(227,227,3,'uint8')} -report
```

Code generation successful: To view the report, open('codegen/mex/logonet\_predict/html/report.mldatx').

## 运行生成的 MEX

加载输入图像。对输入图像调用 `logonet_predict_mex`。

```
im = imread('test.png');
imshow(im);
im = imresize(im, [227,227]);
predict_scores = logonet_predict_mex(im);
```



将排名前五的预测分数映射到 Wordnet 字典 synset 中的单词（徽标）。

```

synsetOut = {'adidas', 'aldi', 'apple', 'becks', 'bmw', 'carlsberg', ...
    'chimay', 'cocacola', 'corona', 'dhl', 'erdinger', 'esso', 'fedex',...
    'ferrari', 'ford', 'fosters', 'google', 'guinness', 'heineken', 'hp',...
    'milka', 'nvidia', 'paulaner', 'pepsi', 'rittersport', 'shell', 'singha', 'starbucks', 'stellaartois', 'texaco', 'tsingtao', 'up...
[val,indx] = sort(predict_scores, 'descend');
scores = val(1:5)*100;
top5labels = synsetOut(indx(1:5));

```

显示排名前五的分类标签。

```
outputImage = zeros(227,400,3, 'uint8');  
for k = 1:3  
    outputImage(:,174:end,k) = im(:, :, k);
```

```
end

scol = 1;
srow = 20;

for k = 1:5
    outputImage = insertText(outputImage, [scol, srow], [top5labels{k}, ',', num2str(scores(k), '%.2f'), '%'], 'TextColor');
    srow = srow + 20;
end

imshow(outputImage);
```



清除内存中已加载的静态网络对象。

```
clear mex;
```

## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)

# 行人检测

此示例说明如何为使用深度学习的行人检测应用程序生成代码。行人检测是计算机视觉的一个关键问题。行人检测在自动驾驶、监控、机器人等领域有诸多应用。

## 前提条件

- 支持 CUDA® 的 NVIDIA® GPU。
- NVIDIA CUDA 工具包和驱动程序。
- NVIDIA cuDNN。
- 编译器和库的环境变量。有关支持的编译器和库的版本的信息，请参阅“Third-Party Hardware”(GPU Coder)。有关设置环境变量的信息，请参阅“Setting Up the Prerequisite Products”(GPU Coder)。
- GPU Coder Interface for Deep Learning Libraries 支持包。要安装此支持包，请使用附加功能资源管理器。

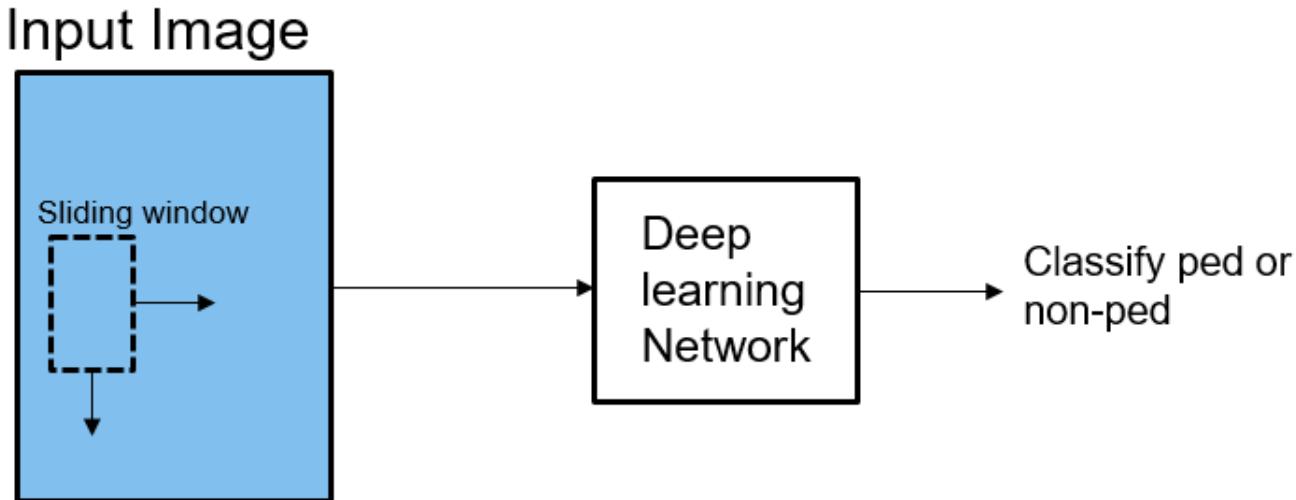
## 验证 GPU 环境

使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

## 行人检测网络

行人检测网络是使用行人和非行人的图像训练得出的。在 MATLAB® 中使用 `trainPedNet.m` 辅助脚本来训练该网络。滑动窗方法从大小为 [64 32] 的图像中裁切补片。补片维度从表示数据集图像中的行人分布的热图中获得。它指示行人在图像中以各种尺寸在不同位置的出现情况。在此示例中，将对靠近相机的行人的补片进行裁切和处理。对获得的补片应用非极大值抑制 (NMS)，以将它们合并在一起并检测完整的行人。



行人检测网络包含 12 个层，包括卷积层、全连接层和分类输出层。

```

load('PedNet.mat');
PedNet.Layers

ans =
12×1 Layer array with layers:

1 'imageinput'  Image Input      64×32×3 images with 'zerocenter' normalization
2 'conv_1'       Convolution    20 5×5×3 convolutions with stride [1 1] and padding [0 0 0]
3 'relu_1'       ReLU          ReLU
4 'maxpool_1'   Max Pooling   2×2 max pooling with stride [2 2] and padding [0 0 0 0]
5 'crossnorm'   Cross Channel Normalization cross channel normalization with 5 channels per element
6 'conv_2'       Convolution    20 5×5×20 convolutions with stride [1 1] and padding [0 0 0 0]
7 'relu_2'       ReLU          ReLU
8 'maxpool_2'   Max Pooling   2×2 max pooling with stride [2 2] and padding [0 0 0 0]
9 'fc_1'         Fully Connected 512 fully connected layer
10 'fc_2'        Fully Connected 2 fully connected layer
11 'softmax'    Softmax        softmax
12 'classoutput' Classification Output crossentropyex with classes 'NonPed' and 'Ped'

```

### pedDetect\_predict 入口函数

**pedDetect\_predict.m** 入口函数以图像作为输入，并使用保存在 **PedNet.mat** 文件中的深度学习网络对图像执行预测。该函数将 **PedNet.mat** 文件中的网络对象加载到持久变量 **pednet** 中。该函数在后续调用中将重用该持久性对象。

```
type('pedDetect_predict.m')
```

```

function selectedBbox = pedDetect_predict(img)
%#codegen

% Copyright 2017-2019 The MathWorks, Inc.

coder.gpu.kernelfun;

persistent pednet;
if isempty(pednet)
    pednet = coder.loadDeepLearningNetwork(coder.const('PedNet.mat'),'Pedestrian_Detection');
end

[imgHt , imgWd , ~] = size(img);
VrHt = [imgHt - 30 , imgHt]; % Two bands of vertical heights are considered

% patchHt and patchWd are obtained from heat maps (heat map here refers to
% pedestrians data represented in the form of a map with different
% colors. Different colors indicate presence of pedestrians at various
% scales).
patchHt = 300;
patchWd = patchHt/3;

% PatchCount is used to estimate number of patches per image
PatchCount = ((imgWd - patchWd)/20) + 2;
maxPatchCount = PatchCount * 2;
Itmp = zeros(64 , 32 , 3 , maxPatchCount);
ltMin = zeros(maxPatchCount);
lttop = zeros(maxPatchCount);

```

```

idx = 1; % To count number of image patches obtained from sliding window
cnt = 1; % To count number of patches predicted as pedestrians

bbox = zeros(maxPatchCount , 4);
value = zeros(maxPatchCount , 1);

%% Region proposal for two bands
for VrStride = 1 : 2
    for HrStride = 1 : 20 : (imgWd - 60) % Obtain horizontal patches with stride 20.
        ltMin(idx) = HrStride + 1;
        rtMax = min(ltMin(idx) + patchWd , imgWd);
        lttop(idx) = (VrHt(VrStride) - patchHt);
        It = img(lttop(idx): VrHt(VrStride) , ltMin(idx) : rtMax , :);
        Itmp(:,:,:idx) = imresize(It,[64,32]);
        idx = idx + 1;
    end
end

for j = 1 : size (Itmp,4)
    score = pednet.predict(Itmp(:,:,:j)); % Classify ROI
    % accuracy of detected box should be greater than 0.90
    if (score(1,2) > 0.80)
        bbox(cnt,:) = [ltMin(j),lttop(j), patchWd , patchHt];
        value(cnt,:) = score(1,2);
        cnt = cnt + 1;
    end
end

%% NMS to merge similar boxes
if ~isempty(bbox)
    [selectedBbox,~] = selectStrongestBbox(bbox(1:cnt-1,:),
                                             value(1:cnt-1,:),'OverlapThreshold',0.002);
end

```

### 为 pedDetect\_predict 函数生成 CUDA MEX

为 MEX 目标创建一个 GPU 配置对象，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。要生成 CUDA MEX，请使用 `codegen` 命令并指定输入图像大小。该值对应于行人检测网络的输入层大小。

```

% Load an input image.
im = imread('test.jpg');
im = imresize(im,[480,640]);

cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg pedDetect_predict -args {im} -report

```

Code generation successful: To view the report, open('codegen/mex/pedDetect\_predict/html/report.mldatx').

### 运行生成的 MEX

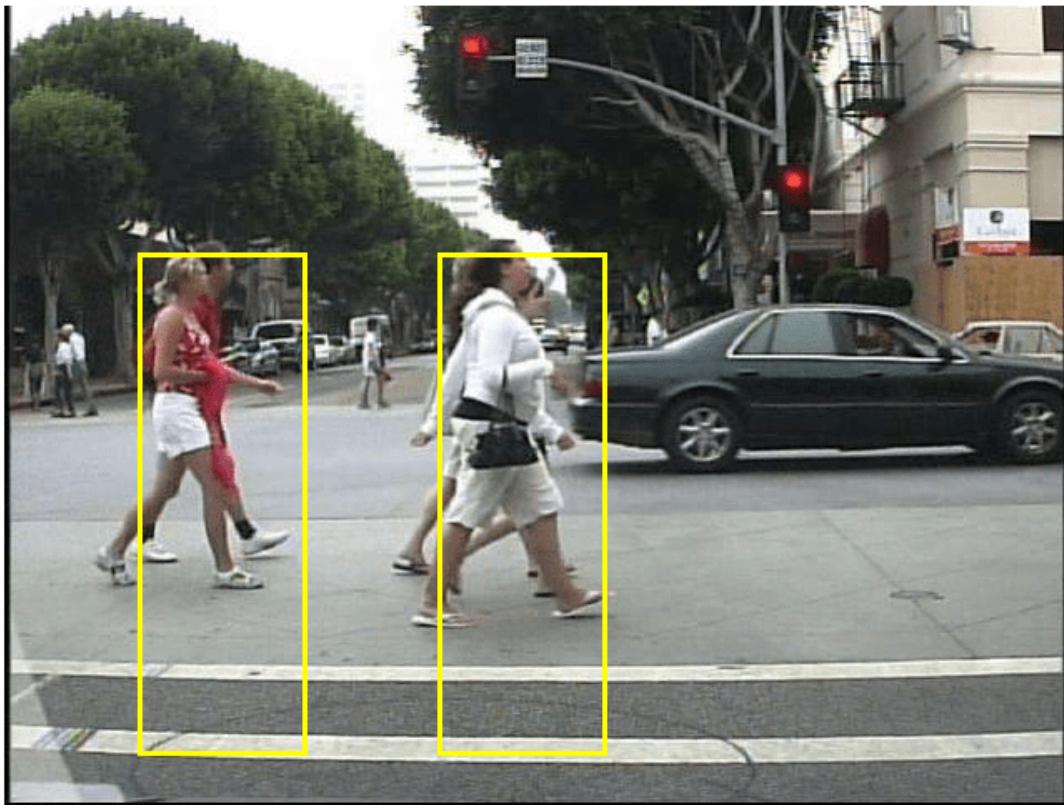
对输入图像调用 `pednet_predict_mex`.

```
imshow(im);
ped_bboxes = pedDetect_predict_mex(im);
```



显示最终预测值。

```
outputImage = insertShape(im,'Rectangle',ped_bboxes,'LineWidth',3);
imshow(outputImage);
```



### 对视频分类

所包含的辅助文件 pedDetect\_predict.m 从视频抓取帧，执行预测，并在每个捕获的视频帧上显示分类结果。

```
v = VideoReader('LiveData.avi');
fps = 0;
while hasFrame(v)
    % Read frames from video
    im = readFrame(v);
    im = imresize(im,[480,640]);

    % Call MEX function for pednet prediction
    tic;
    ped_bboxes = pedDetect_predict_mex(im);
    newt = toc;

    % fps
    fps = .9*fps + .1*(1/newt);

    % display
    outputImage = insertShape(im,'Rectangle',ped_bboxes,'LineWidth',3);
    imshow(outputImage)
    pause(0.2)
end
```

清除内存中已加载的静态网络对象。

```
clear mex;
```

## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “深度学习在计算机视觉领域的应用”

# 去噪深度神经网络的代码生成

此示例说明如何从 MATLAB® 代码生成 CUDA® MEX，以及如何使用去噪卷积神经网络 (DnCNN [1]) 对灰度图像进行去噪。您可以使用去噪网络估计含噪图像中的噪声，然后将其去除以获得去噪图像。

## 第三方前提条件

### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅“Third-Party Hardware” (GPU Coder) 和 “Setting Up the Prerequisite Products” (GPU Coder)。

## 验证 GPU 环境

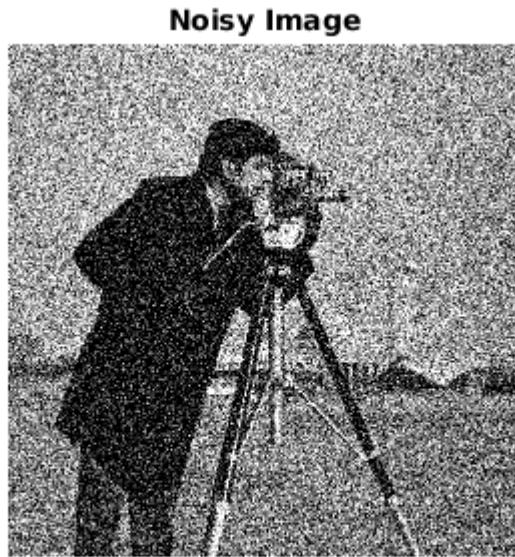
使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

## 加载含噪图像

将含噪灰度图像加载到工作区并显示图像。

```
noisyI = imread('noisy_cameraman.png');
figure
imshow(noisyI);
title('Noisy Image');
```



### 获得预训练的去噪网络

调用 `getDenoisingNetwork` 辅助函数可获得预训练的图像去噪深度神经网络。

```
net = getDenoisingNetwork;
```

`getDenoisingNetwork` 函数返回预训练的 DnCNN [1]，您可以使用它来检测未知级别的加性高斯白噪声 (AWGN)。该网络是前馈去噪卷积网络，它实现一种残差学习方法来预测残差图像。换句话说，DnCNN [1] 会计算噪声图像与潜在清洁图像之间的差异。

该网络包含 59 个层，包括卷积层、批量归一化层和回归输出层。要以交互可视方式呈现深度学习网络架构，请使用 `analyzeNetwork` 函数。

```
analyzeNetwork(net);
```

### denoisenet\_predict 函数

`denoisenet_predict` 入口函数以含噪图像作为输入，并使用预训练的去噪网络返回去噪图像。

该函数将 `getDenoisingNetwork` 返回的网络对象加载到持久变量 `mynet` 中，并在后续的预测调用中重用该持久变量。

```
type denoisenet_predict
```

```
function I = denoisenet_predict(in)
%#codegen
% Copyright 2018-2019 The MathWorks, Inc.

persistent mynet;

if isempty(mynet)
```

```

mynet = coder.loadDeepLearningNetwork('getDenoisingNetwork', 'DnCNN');
end

% The activations methods extracts the output from the last layer. The
% 'OutputAs' 'channels' name-value pair argument is used in order to call
% activations on an image whose input dimensions are greater than or equal
% to the network's imageInputLayer.InputSize.

res = mynet.activations(in, 59,'OutputAs','channels');

% Once the noise is estimated, we subtract the noise from the original
% image to obtain a denoised image.

I = in - res;

```

此处调用了 **activations** 方法（层数值索引为 59）来从网络的最终层提取激活值。**'OutputAs'** '**'channels'** 名称-值对组参数用于计算大于网络的 **imageInputLayer.InputSize** 的图像上的激活值。

**activations** 方法使用预训练的去噪图像返回输入图像中的噪声估计值。

估计出噪声后，从原始图像减去噪声以获得去噪图像。

## 运行 MEX 代码生成

要为 **denoisenet\_predict.m** 入口函数生成 CUDA 代码，请为 MEX 目标创建一个 GPU 代码配置对象，并将目标语言设置为 C++。使用 **coder.DeepLearningConfig** (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 **DeepLearningConfig** 属性。运行 **codegen** 命令以指定输入大小为 [256,256]。该值对应于要去噪的含噪图像的大小。

```

cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg denoisenet_predict -args {ones(256,256,'single')} -report

```

Code generation successful: To view the report, open('codegen/mex/denoisenet\_predict/html/report.mldatx').

## 运行生成的 MEX

针对输入范围为 [0,1] 的输入图像训练 DnCNN [1]。对 **noisyI** 调用 **im2single** (Image Processing Toolbox) 函数以将值从 [0,255] 重新调整为 [0,1]。

对经过重新调整的输入图像调用 **denoisenet\_predict\_predict**。

```
denoisedI = denoisenet_predict_mex(im2single(noisyI));
```

## 查看去噪图像

```

figure
imshowpair(noisyI,denoisedI,'montage');
title('Noisy Image (left) and Denoised Image (right)');

```

Noisy Image (left) and Denoised Image (right)



## 参考资料

[1] Zhang, K., W. Zuo, Y. Chen, D. Meng, and L. Zhang."Beyond a Gaussian Denoiser:Residual Learning of Deep CNN for Image Denoising."IEEE Transactions on Image Processing.Vol. 26, Number 7, Feb. 2017, pp. 3142-3155.

## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习” (第 1-2 页)
- “深度学习在图像处理领域的应用”

# 训练和部署用于语义分割的全卷积网络

此示例说明如何使用 GPU Coder™ 在 NVIDIA® GPU 上训练和部署全卷积语义分割网络。

语义分割网络对图像中的每个像素进行分类，从而生成按类分割的图像。语义分割的应用包括自动驾驶中的道路分割以及医疗诊断中的癌细胞分割。要了解详细信息，请参阅“Getting Started with Semantic Segmentation Using Deep Learning”(Computer Vision Toolbox)。

为了说明训练过程，此示例将训练 FCN-8s [1]，这是一种专门用于语义图像分割的卷积神经网络(CNN)。其他类型的用于语义分割的网络包括全卷积网络，如 SegNet 和 U-Net。您也可以将此训练过程应用于这些网络。

此示例使用剑桥大学的 CamVid 数据集 [2] 进行训练。此数据集是包含驾驶时获得的街道级视图的图像集合。该数据集提供了 32 个语义类的像素级标签，包括汽车、行人和道路。

## 第三方前提条件

### 必需

- 支持 CUDA® 的 NVIDIA GPU 和兼容驱动程序。

### 可选

- NVIDIA CUDA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关支持的编译器和库的版本的信息，请参阅“Third-Party Hardware”(GPU Coder)。有关设置环境变量的信息，请参阅“Setting Up the Prerequisite Products”(GPU Coder)。

## 验证 GPU 环境

使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

## 设置

此示例创建的全卷积语义分割网络具有从 VGG-16 网络初始化的权重。`vgg16` 函数检查是否存在 Deep Learning Toolbox Model for VGG-16 Network 支持包，并返回预训练 VGG-16 模型。

```
vgg16();
```

下载 FCN 的预训练版本。借助此预训练模型，您无需等待训练完成，即可运行整个示例。`doTraining` 标志控制示例是使用示例的训练网络还是预训练的 FCN 网络来生成代码。

```
doTraining = false;
if ~doTraining
    pretrainedURL = 'https://www.mathworks.com/supportfiles/gpucoder/cnn\_models/fcn/FCN8sCamVid.mat';
    disp('Downloading pretrained FCN (448 MB...)');
    websave('FCN8sCamVid.mat',pretrainedURL);
end
```

Downloading pretrained FCN (448 MB)...

### 下载 CamVid 数据集

从以下 URL 下载 CamVid 数据集。

```
 imageURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/files/701_StillsRaw_full.zip';
 labelURL = 'http://web4.cs.ucl.ac.uk/staff/g.brostow/MotionSegRecData/data/LabeledApproved_full.zip';

 outputFolder = fullfile(pwd,'CamVid');

 if ~exist(outputFolder, 'dir')

 mkdir(outputFolder)
 labelsZip = fullfile(outputFolder,'labels.zip');
 imagesZip = fullfile(outputFolder,'images.zip');

 disp('Downloading 16 MB CamVid dataset labels...');
 websave(labelsZip, labelURL);
 unzip(labelsZip, fullfile(outputFolder,'labels'));

 disp('Downloading 557 MB CamVid dataset images...');
 websave(imagesZip, imageURL);
 unzip(imagesZip, fullfile(outputFolder,'images'));
end
```

数据下载时间取决于您的 Internet 连接。下载操作完成后，示例执行才会继续。您也可以使用 Web 浏览器先将数据集下载到本地磁盘。然后，使用 outputFolder 变量指向所下载文件的位置。

### 加载 CamVid 图像

使用 imageDatastore 加载 CamVid 图像。通过 imageDatastore 可将大量图像高效加载至磁盘。

```
 imgDir = fullfile(outputFolder,'images','701_StillsRaw_full');
 imds = imageDatastore(imgDir);
```

显示其中一个图像。

```
I = readimage(imds,25);
I = histeq(I);
imshow(I)
```



### 加载 CamVid 像素标注图像

使用 `pixelLabelDatastore` (Computer Vision Toolbox) 加载 CamVid 像素标签图像数据。`pixelLabelDatastore` 将像素标签数据和标签 ID 封装到类名映射中。

按照 SegNet 论文 [3] 中描述的训练方法，将 CamVid 中的 32 个原始类分组为 11 个类。指定这些类。

```
classes = [  
    "Sky"  
    "Building"  
    "Pole"  
    "Road"  
    "Pavement"  
    "Tree"  
    "SignSymbol"  
    "Fence"  
    "Car"  
    "Pedestrian"  
    "Bicyclist"  
];
```

要将 32 个类减少为 11 个类，需要将原始数据集中的多个类组合在一起。例如，“Car”是“Car”、“SUVPickupTruck”、“Truck\_Bus”、“Train”和“OtherMoving”的组合。使用 camvidPixelLabelIDs 支持函数返回分组的标签 ID。

```
labelIDs = camvidPixelLabelIDs();
```

使用类和标签 ID 创建 pixelLabelDatastore。

```
labelDir = fullfile(outputFolder,'labels');  
pxds = pixelLabelDatastore(labelDir,classes,labelIDs);
```

读取一个像素标注图像，并将其叠加在图像上方显示。

```
C = readimage(pxds,25);  
cmap = camvidColorMap;  
B = labeloverlay(I,C,'ColorMap',cmap);  
imshow(B)  
pixelLabelColorbar(cmap,classes);
```



没有颜色叠加的区域没有像素标签，在训练过程中不被使用。

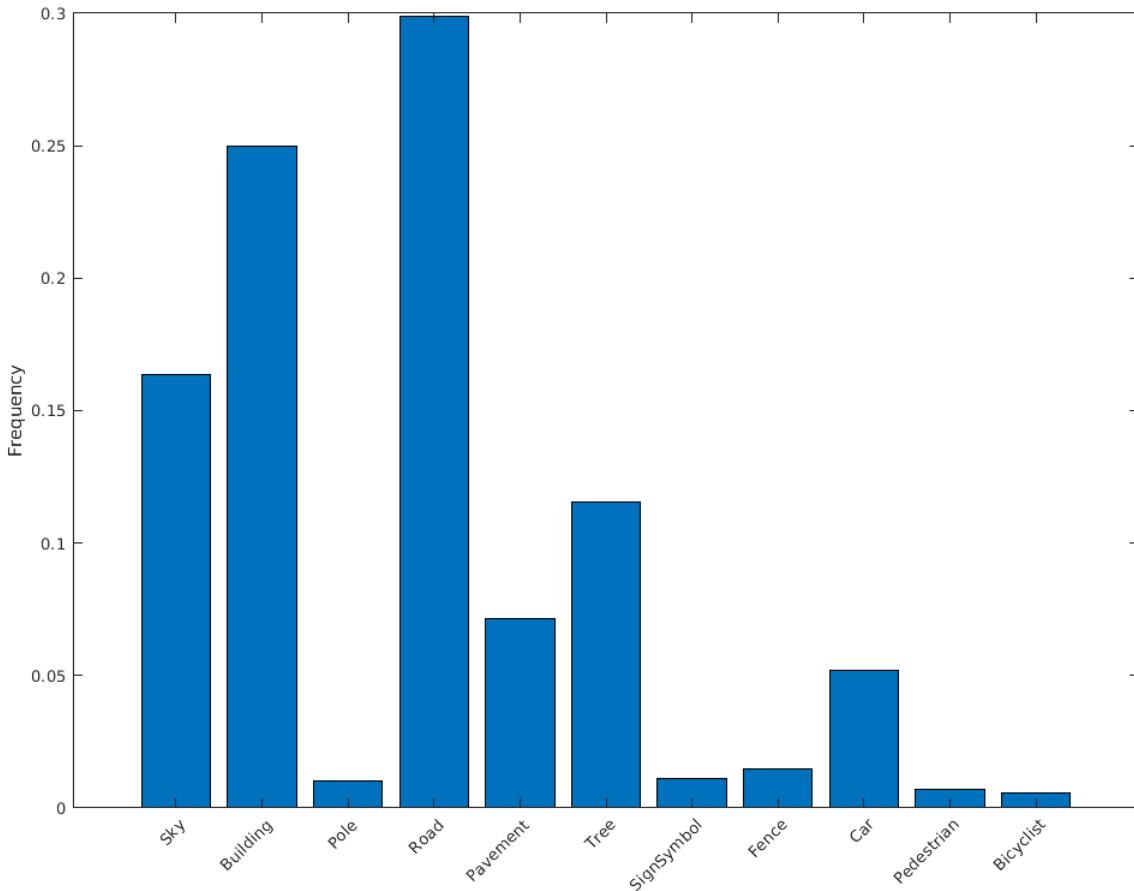
## 分析数据集统计信息

要查看 CamVid 数据集中类标签的分布，请使用 **countEachLabel** (Computer Vision Toolbox)。此函数按类标签计算像素数。

```
tbl = countEachLabel(pxds)  
tbl=11×3 table  
    Name      PixelCount   ImagePixelCount  
    _____  
    {'Sky'}    7.6801e+07  4.8315e+08  
    {'Building'} 1.1737e+08  4.8315e+08  
    {'Pole'}    4.7987e+06  4.8315e+08  
    {'Road'}    1.4054e+08  4.8453e+08  
    {'Pavement'} 3.3614e+07  4.7209e+08  
    {'Tree'}    5.4259e+07  4.479e+08  
    {'SignSymbol'} 5.2242e+06  4.6863e+08  
    {'Fence'}    6.9211e+06  2.516e+08  
    {'Car'}     2.4437e+07  4.8315e+08  
    {'Pedestrian'} 3.4029e+06  4.4444e+08  
    {'Bicyclist'} 2.5912e+06  2.6196e+08
```

按类可视化像素计数。

```
frequency = tbl.PixelCount/sum(tbl.PixelCount);  
  
bar(1:numel(classes),frequency)  
xticks(1:numel(classes))  
xticklabels(tbl.Name)  
xtickangle(45)  
ylabel('Frequency')
```



理想情况下，所有类都有相同数量的观测值。CamVid 中的类是不平衡的，这是街景汽车数据集的常见问题。此类场景的天空、建筑物和道路像素比行人和骑车人像素多，因为天空、建筑物和道路覆盖了图像中的更多区域。如果处理不当，这种不平衡可能对学习过程不利，因为学习会偏向于占主导的类。在此示例的稍后部分，您将使用类权重来处理此问题。

### 调整 CamVid 数据的大小

CamVid 数据集中的图像大小为  $720 \times 960$ 。为减少训练时间和内存使用量，请使用 `resizeCamVidImages` 和 `resizeCamVidPixelLabels` 支持函数将图像和像素标签图像的大小调整为  $360 \times 480$ 。

```
imageFolder = fullfile(outputFolder,'imagesResized',filesep);
imds = resizeCamVidImages(imds,imageFolder);

labelFolder = fullfile(outputFolder,'labelsResized',filesep);
pxds = resizeCamVidPixelLabels(pxds,labelFolder);
```

### 准备训练集和测试集

使用数据集中 60% 的图像训练 SegNet。其余图像用于测试。以下代码将图像和像素标签数据随机分成训练集和测试集。

```
[imdsTrain,imdsTest,pxdsTrain,pxdsTest] = partitionCamVidData(imds,pxds);
```

60/40 拆分产生以下数量的训练和测试图像：

```
numTrainingImages = numel(imdsTrain.Files)
numTrainingImages = 421
numTestingImages = numel(imdsTest.Files)
numTestingImages = 280
```

## 创建网络

使用 **fcnLayers** (Computer Vision Toolbox) 创建使用 VGG-16 权重初始化的全卷积网络层。**fcnLayers** 函数执行网络变换以从 VGG-16 传递权重，并添加语义分割所需的其他层。**fcnLayers** 函数的输出是一个表示 FCN 的 LayerGraph 对象。LayerGraph 对象封装了网络层及各层之间的连接。

```
imageSize = [360 480];
numClasses = numel(classes);
lgraph = fcnLayers(imageSize,numClasses);
```

图像大小的选择基于数据集中的图像大小。类数量的选择基于 CamVid 中的类。

## 使用类权重平衡类

CamVid 中的类并不平衡。为了改进训练，您可以使用先前通过 **countEachLabel** (Computer Vision Toolbox) 函数计算的像素标签计数，计算具有中位数频率的类的权重 [3]。

```
imageFreq = tbl.PixelCount ./ tbl.ImagePixelCount;
classWeights = median(imageFreq) ./ imageFreq;
```

使用 **pixelClassificationLayer** (Computer Vision Toolbox) 指定类权重。

```
pxLayer = pixelClassificationLayer('Name','labels','Classes',tbl.Name,'ClassWeights',classWeights)
```

```
pxLayer =
  PixelClassificationLayer with properties:
```

```
  Name: 'labels'
  Classes: [11×1 categorical]
  ClassWeights: [11×1 double]
  OutputSize: 'auto'
```

```
  Hyperparameters
  LossFunction: 'crossentropyex'
```

通过删除当前 **pixelClassificationLayer** 并添加新层，来更新具有新 **pixelClassificationLayer** 的 SegNet 网络。当前 **pixelClassificationLayer** 命名为 '**pixelLabels**'。使用 **removeLayers** 函数将其删除，使用 **addLayers** 函数添加一个新层，然后使用 **connectLayers** 函数将新层连接到网络的其余部分。

```
lgraph = removeLayers(lgraph,'pixelLabels');
lgraph = addLayers(lgraph, pxLayer);
lgraph = connectLayers(lgraph,'softmax','labels');
```

## 选择训练选项

训练的优化算法是 Adam (派生自自适应矩估计)。使用 **trainingOptions** 函数指定用于 Adam 的超参数。

```
options = trainingOptions('adam', ...
    'InitialLearnRate',1e-3, ...
    'MaxEpochs',100, ...
    'MiniBatchSize',4, ...
    'Shuffle','every-epoch', ...
    'CheckpointPath', tempdir, ...
    'VerboseFrequency',2);
```

大小为 4 的 'MiniBatchSize' 可减少训练时的内存使用量。您可以根据系统中的 GPU 内存量增大或减小此值。

'CheckpointPath' 设置为临时位置。此名称-值对组让您能够在每轮训练结束时保存网络检查点。如果由于系统故障或停电而导致训练中断，您可以从保存的检查点处恢复训练。确保 'CheckpointPath' 指定的位置有足够的空间来存储网络检查点。

### 数据增强

使用数据增强为网络提供更多样本，因为这有助于提高网络的准确度。此处使用随机左/右翻转和随机 X/Y 平移 +/- 10 个像素来进行数据增强。使用  `imageDataAugmenter` 函数指定这些数据增强参数。

```
augmenter = imageDataAugmenter('RandXReflection',true, ...
    'RandXTranslation',[-10 10],'RandYTranslation',[-10 10]);
```

The  `imageDataAugmenter` 函数支持多种其他类型的数据增强。具体选择哪一种需要根据经验分析，而且涉及另一级别的超参数调整。

### 开始训练

使用  `pixelLabelImageDatastore` (Computer Vision Toolbox) 函数组合训练数据和数据增强选择。 `pixelLabelImageDatastore` 函数会读取批量训练数据，应用数据增强，并将增强后的数据发送给训练算法。

```
pximds = pixelLabelImageDatastore(imdsTrain,pxdsTrain, ...
    'DataAugmentation',augmenter);
```

如果 `doTraining` 标志为 `true`，请使用  `trainNetwork` 函数开始训练。

训练在具有 12 GB GPU 内存的 NVIDIA™ Titan Xp 上进行了验证。如果您的 GPU 内存较少，则可能内存不足。如果系统的内存不足，请尝试将  `trainingOptions` 中的  `MiniBatchSize` 属性降低到 1。根据您的 GPU 硬件情况，训练此网络需要大约 5 个小时或更长时间。

```
if doTraining
    [net, info] = trainNetwork(pximds,lgraph,options);
    save('FCN8sCamVid.mat','net');
end
```

将 DAG 网络对象保存为名为  `FCN8sCamVid.mat` 的 MAT 文件。在代码生成过程中将使用此 MAT 文件。

### 执行 MEX 代码生成

`fcn_predict.m` 函数以图像作为输入，并使用保存在  `FCN8sCamVid.mat` 文件中的深度学习网络对图像执行预测。该函数将  `FCN8sCamVid.mat` 中的网络对象加载到持久变量 `mynet` 中，并在后续的预测调用中重用该持久性对象。

```
type('fcn_predict.m')
```

```
function out = fcn_predict(in)
%#codegen
% Copyright 2018-2019 The MathWorks, Inc.

persistent mynet;

if isempty(mynet)
    mynet = coder.loadDeepLearningNetwork('FCN8sCamVid.mat');
end

% pass in input
out = predict(mynet,in);
```

为 MEX 对象生成一个 GPU 配置对象以将目标语言设置为 C++。使用 **coder.DeepLearningConfig** (GPU Coder) 函数创建一个 cuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 **DeepLearningConfig** 属性。运行 **codegen** (MATLAB Coder) 命令以指定输入大小为 [360, 480, 3]。此大小对应于 FCN 的输入层。

```
cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg fcn_predict -args {ones(360,480,3,'uint8')} -report
```

Code generation successful: View report

### 运行生成的 MEX

加载并显示输入图像。

```
im = imread('testImage.png');
imshow(im);
```



对输入图像调用 `fcn_predict_mex` 来运行预测。

```
predict_scores = fcn_predict_mex(im);
```

`predict_scores` 变量是一个三维矩阵，它具有 11 个通道，分别对应于每个类的像素级预测分数。使用最高预测分数计算通道以获得像素级标签。

```
[~,argmax] = max(predict_scores,[],3);
```

在输入图像上叠加分割标签并显示分割区域。

```
classes = [  
    "Sky"  
    "Building"  
    "Pole"  
    "Road"  
    "Pavement"  
    "Tree"  
    "SignSymbol"  
    "Fence"  
    "Car"  
    "Pedestrian"  
    "Bicyclist"  
];
```

```
cmap = camvidColorMap();
```

```
SegmentedImage = labeloverlay(im,argmax,'ColorMap',cmap);
figure
imshow(SegmentedImage);
pixelLabelColorbar(cmap,classes);
```



## 清理

清除内存中已加载的静态网络对象。

```
clear mex;
```

## 参考资料

- [1] Long, J., E. Shelhamer, and T. Darrell."Fully Convolutional Networks for Semantic Segmentation."Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.
- [2] Brostow, G. J., J. Fauqueur, and R. Cipolla."Semantic object classes in video:A high-definition ground truth database."Pattern Recognition Letters.Vol. 30, Issue 2, 2009, pp 88-97.
- [3] Badrinarayanan, V., A. Kendall, and R. Cipolla."SegNet:A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation." arXiv preprint arXiv:1511.00561, 2015.

## 使用 U-Net 的语义分割网络的代码生成

此示例说明如何为使用深度学习的图像分割应用程序生成代码。它使用 `codegen` 命令生成一个基于 U-Net 的 DAG 网络对象执行预测的 MEX 函数。U-Net 是一种用于图像分割的深度学习网络。

有关使用 U-Net 进行图像分割但不使用 `codegen` 命令的类似示例，请参阅 “Semantic Segmentation of Multispectral Images Using Deep Learning” (Image Processing Toolbox)。

### 第三方前提条件

#### 必需

此示例生成 CUDA MEX，并具有以下第三方要求。

- CUDA® 支持 NVIDIA® GPU 和兼容驱动程序。

#### 可选

对于非 MEX 编译，如静态、动态库或可执行文件，此示例有以下附加要求。

- NVIDIA 工具包。
- NVIDIA cuDNN 库。
- 编译器和库的环境变量。有关详细信息，请参阅 “Third-Party Hardware” (GPU Coder) 和 “Setting Up the Prerequisite Products” (GPU Coder)。

### 验证 GPU 环境

使用 `coder.checkGpuInstall` (GPU Coder) 函数验证运行此示例所需的编译器和库是否已正确设置。

```
envCfg = coder.gpuEnvConfig('host');
envCfg.DeepLibTarget = 'cudnn';
envCfg.DeepCodegen = 1;
envCfg.Quiet = 1;
coder.checkGpuInstall(envCfg);
```

### 分割网络

U-Net [1] 是一种卷积神经网络 (CNN)，专为语义图像分割而设计。U-Net 中的初始卷积层序列与最大池化层交叠，从而逐步降低输入图像的分辨率。这些层后跟一系列使用上采样算子散布的卷积层，从而会继续增加输入图像的分辨率。将这两个系列路径组合在一起将形成一个 U 形图。该网络最初是为生物医学图像分割应用执行预测而训练和使用的。本示例中演示的是该网络跟踪森林植被随时间变化的能力。环保机构跟踪森林采伐，以评估和鉴定地区的环境和生态健康。

基于深度学习的语义分割可以通过高分辨率航拍照片精确测量植被覆盖度。一个挑战是区分具有相似视觉特性的类，例如尝试将绿色像素分类为草、灌木或树。为了提高分类准确度，一些数据集包含多光谱图像，这些图像提供关于每个像素的附加信息。例如，Hamlin Beach 国家公园数据集用近红外通道为彩色图像补充信息，以提供更清晰的分类。

本示例使用 Hamlin Beach 国家公园数据 [2] 以及预训练 U-Net 网络，以便正确地对每个像素进行分类。

使用的 U-Net 经过训练，用于对属于以下 18 个类的像素进行分割：

- |                             |                     |                  |
|-----------------------------|---------------------|------------------|
| 0. Other Class/Image Border | 7. Picnic Table     | 14. Grass        |
| 1. Road Markings            | 8. Black Wood Panel | 15. Sand         |
| 2. Tree                     | 9. White Wood Panel | 16. Water (Lake) |

- |                                 |                        |                                   |
|---------------------------------|------------------------|-----------------------------------|
| 3. Building                     | 10. Orange Landing Pad | 17. Water (Pond)                  |
| 4. Vehicle (Car, Truck, or Bus) | 11. Water Buoy         | 18. Asphalt (Parking Lot/Walkway) |
| 5. Person                       | 12. Rocks              |                                   |
| 6. Lifeguard Chair              | 13. Other Vegetation   |                                   |

### segmentImageUnet 入口函数

**segmentImageUnet.m** 入口函数使用 **multispectralUnet.mat** 文件中的 multispectralUnet 网络对输入图像执行逐块语义分割。该函数将 **multispectralUnet.mat** 文件中的网络对象加载到持久变量 **mynet** 中，并在后续的预测调用中重用该持久变量。

```
type('segmentImageUnet.m')
```

```
% OUT = segmentImageUnet(IM, PATCHSIZE) returns a semantically segmented
% image, segmented using the network multispectralUnet. The segmentation
% is performed over each patch of size PATCHSIZE.
%
% Copyright 2019-2020 The MathWorks, Inc.
function out = segmentImageUnet(im, patchSize)

 %#codegen

 persistent mynet;

 if isempty(mynet)
     mynet = coder.loadDeepLearningNetwork('trainedUnet/multispectralUnet.mat');
 end

 [height, width, nChannel] = size(im);
 patch = coder.nullcopy(zeros([patchSize, nChannel-1]));

 % pad image to have dimensions as multiples of patchSize
 padSize = zeros(1,2);
 padSize(1) = patchSize(1) - mod(height, patchSize(1));
 padSize(2) = patchSize(2) - mod(width, patchSize(2));

 im_pad = padarray (im, padSize, 0, 'post');
 [height_pad, width_pad, ~] = size(im_pad);

 out = zeros([size(im_pad,1), size(im_pad,2)], 'uint8');

 for i = 1:patchSize(1):height_pad
     for j = 1:patchSize(2):width_pad
         for p = 1:nChannel-1
             patch(:,:,p) = squeeze( im_pad( i:i+patchSize(1)-1, ...
                 j:j+patchSize(2)-1, ...
                 p));
         end

         % pass in input
         segmentedLabels = activations(mynet, patch, 'Segmentation-Layer');

         % Takes the max of each channel (6 total at this point)
         [~,L] = max(segmentedLabels,[],3);
         patch_seg = uint8(L);

         % populate section of output
     end
 end
```

```
    out(i:i+patchSize(1)-1, j:j+patchSize(2)-1) = patch_seg;  
end  
end  
  
% Remove the padding  
out = out(1:height, 1:width);
```

### 获取预训练的 U-Net DAG 网络对象

```
trainedUnet_url = 'https://www.mathworks.com/supportfiles/vision/data/multispectralUnet.mat';  
downloadTrainedUnet(trainedUnet_url, pwd);
```

Downloading Pre-trained U-net for Hamlin Beach dataset...  
This will take several minutes to download...  
done.

```
ld = load("trainedUnet/multispectralUnet.mat");  
net = ld.net;
```

DAG 网络包含 58 个层，其中包括卷积层、最大池化层、深度串联层和像素分类输出层。要以交互可视方式呈现深度学习网络架构，请使用 **analyzeNetwork** 函数。`analyzeNetwork(net)`

### 准备数据

下载 Hamlin Beach 国家公园的数据。

```
if ~exist(fullfile(pwd,'data'))  
    url = 'http://www.cis.rit.edu/~rmk6217/rit18\_data.mat';  
    downloadHamlinBeachMSIData(url,pwd+"/data");  
end
```

Downloading Hamlin Beach dataset...  
This will take several minutes to download...  
done.

在 MATLAB 中加载并检查数据。

```
load(fullfile(pwd,'data','rit18_data','rit18_data.mat'));
```

```
% Examine data  
whos test_data
```

Name	Size	Bytes	Class	Attributes
test_data	7x12446x7654	1333663576	uint16	

该图像有七个通道。RGB 颜色通道是第四、第五和第六个图像通道。前三个通道对应于近红外波段，并基于其热能特征突出显示图像的不同分量。通道 7 是指示有效分割区域的掩膜。

多光谱图像数据排列为通道数×宽×高数组。在 MATLAB 中，多通道图像排列为宽×高×通道数数组。要重构数据以使通道处于第三个维度中，请使用辅助函数 **switchChannelsToThirdPlane**。

```
test_data = switchChannelsToThirdPlane(test_data);
```

```
% Confirm data has the correct structure (channels last).
whos test_data
```

Name	Size	Bytes	Class	Attributes
test_data	12446x7654x7	1333663576	uint16	

## 运行 MEX 代码生成

要为 `segmentImageUnet.m` 入口函数生成 CUDA 代码，请为 MEX 目标创建一个 GPU 代码配置对象，并将目标语言设置为 C++。使用 `coder.DeepLearningConfig` (GPU Coder) 函数创建一个 CuDNN 深度学习配置对象，并将其赋给 GPU 代码配置对象的 `DeepLearningConfig` 属性。运行 `codegen` 命令，指定输入大小为 [12446,7654,7]，补片大小为 [1024,1024]。这些值对应于整个 `test_data` 大小。较小的补片大小可加速推断。要了解如何计算补片，请参阅 `segmentImageUnet.m` 入口函数。

```
cfg = coder.gpuConfig('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn');
codegen -config cfg segmentImageUnet -args {ones(size(test_data),'uint16'),coder.Constant([1024 1024])} -report
```

Code generation successful: To view the report, open('codegen/mex/segmentImageUnet/html/report.mldatx').

## 运行生成的 MEX 来预测 `test_data` 的结果

此 `segmentImageUnet` 函数接受要测试的数据 (`test_data`) 和一个向量，向量中包含要使用的补片的大小。获取图像的补片，预测特定补片中的像素，然后将所有补片组合在一起。由于 `test_data` 大小为 (12446×7654×7)，以补片形式处理如此大的图像更容易。

```
segmentedImage = segmentImageUnet_mex(test_data,[1024 1024]);
```

为了只提取分割的有效部分，将分割的图像乘以测试数据的封装通道。

```
segmentedImage = uint8(test_data(:,:,7)~=0) .* segmentedImage;
```

由于语义分割的输出含有噪声，请使用 `medfilt2` 函数去除噪声和杂散像素。

```
segmentedImage = medfilt2(segmentedImage,[5,5]);
```

## 显示 U-Net 分割后的 `test_data`

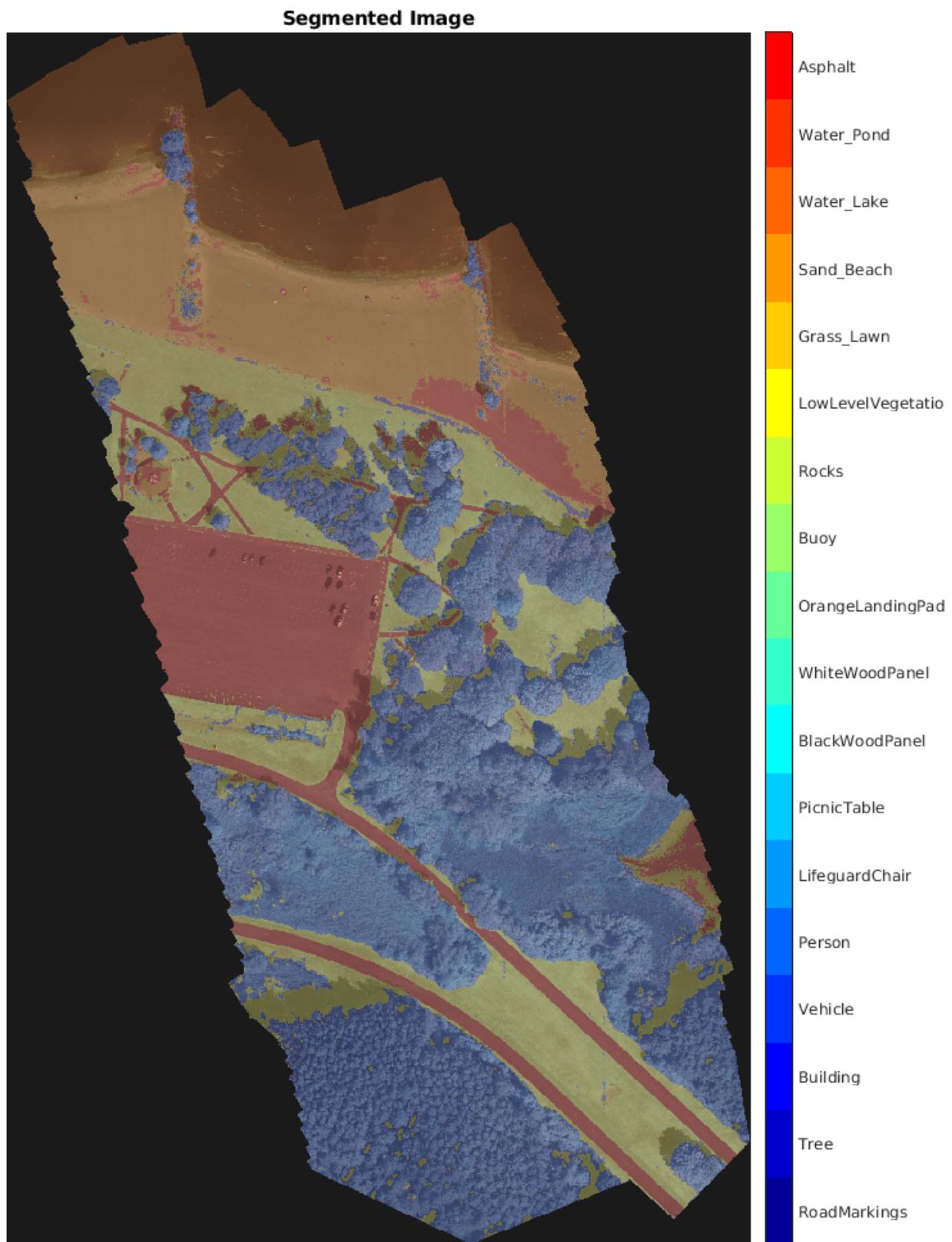
以下代码行创建类名的向量。

```
classNames = [ "RoadMarkings","Tree","Building","Vehicle","Person", ...
    "LifeguardChair","PicnicTable","BlackWoodPanel",...
    "WhiteWoodPanel","OrangeLandingPad","Buoy","Rocks",...
    "LowLevelVegetation","Grass_Lawn","Sand_Beach",...
    "Water_Lake","Water_Pond","Asphalt"];
```

在分割的 RGB 测试图像上叠加标签，并向分割图像添加颜色栏。

```
cmap = jet(numel(classNames));
B = labeloverlay(imadjust(test_data(:,:,4:6),[0 0.6],[0.1 0.9],0.55),segmentedImage,'Transparency',0.8,'Colormap',cmap);
figure
imshow(B)
```

```
N = numel(classNames);
ticks = 1/(N*2):1/N:1;
colorbar('TickLabels',cellstr(classNames),'Ticks',ticks,'TickLength',0,'TickLabelInterpreter','none');
colormap(cmap)
title('Segmented Image');
```



### 参考资料

- [1] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox."U-Net:Convolutional Networks for Biomedical Image Segmentation." arXiv preprint arXiv:1505.04597, 2015.
- [2] Kemker, R., C. Salvaggio, and C. Kanan."High-Resolution Multispectral Dataset for Semantic Segmentation."CoRR, abs/1703.01918, 2017.

# ARM 目标上的深度学习代码生成

此示例说明如何在不使用硬件支持包的情况下，在基于 ARM® 的设备上为预测生成和部署代码。

当您使用 ARM Compute Library 和硬件支持包为预测生成代码时，`codegen` 会在主机上生成代码，将生成的文件复制到目标硬件，并在目标硬件上编译可执行文件。在没有硬件支持包的情况下，`codegen` 会在主机上生成代码。您必须运行命令来复制文件并在目标硬件上编译可执行程序。

此示例使用 `packNGo` 函数将所有相关文件打包到一个压缩的 zip 文件中。使用此示例了解如何使用 `packNGo` 在没有硬件支持包的 ARM Neon 目标上部署生成的代码。

## 前提条件

- 支持 NEON 扩展的 ARM 处理器
- ARM Compute Library (在目标 ARM 硬件上)
- 开源计算机视觉库 (Open CV)
- 编译器和库的环境变量。
- MATLAB® Coder™
- MATLAB Coder Interface for Deep Learning 支持包
- Deep Learning Toolbox™

此示例使用的 ARM Compute Library 版本可能不是代码生成支持的最新版本。有关受支持的库版本和有关设置环境变量的信息，请参阅“使用 MATLAB Coder 进行深度学习的前提条件”(MATLAB Coder)。

MATLAB Online 不支持此示例。

## squeezeNet\_predict 函数

此示例通过 ARM Compute Library 使用 DAG 网络 SqueezeNet 显示图像分类。Deep Learning Toolbox 中提供了针对 MATLAB 的预训练 SqueezeNet。`squeezeNet_predict` 函数将 SqueezeNet 网络加载到一个持久性网络对象中。在对该函数的后续调用中，将重复使用该持久性对象。

```
type squeezeNet_predict

% Copyright 2018 The MathWorks, Inc.

function out = squeezeNet_predict(in)
%#codegen

% A persistent object mynet is used to load the DAG network object.
% At the first call to this function, the persistent object is constructed and
% set up. When the function is called subsequent times, the same object is reused
% to call predict on inputs, avoiding reconstructing and reloading the
% network object.

persistent mynet;
if isempty(mynet)
    mynet = coder.loadDeepLearningNetwork('squeezeNet','squeezeNet');
end

out = mynet.predict(in);
```

### 为静态库设置代码生成配置对象

当您针对基于 ARM 的设备生成代码并且不使用硬件支持包时，请为库创建一个配置对象。不要为可执行程序创建配置对象。

为生成 C++ 代码和仅生成代码设置配置对象。

```
cfg = coder.config('lib');
cfg.TargetLang = 'C++';
cfg.GenCodeOnly = true;
```

### 为深度学习代码生成设置配置对象

创建一个 `coder.ARMNEONConfig` 对象。指定目标 ARM 处理器的库版本和架构。例如，假设目标板是具有 ARMv8 架构和 ARM Compute Library 19.05 版本的 HiKey/Rock960 板。

```
dlcfg = coder.DeepLearningConfig('arm-compute');
dlcfg.ArmComputeVersion = '19.05';
dlcfg.ArmArchitecture = 'armv8';
```

### 将深度学习配置对象连接到代码生成配置对象

将代码生成配置对象的 `DeepLearningConfig` 属性设置为深度学习配置对象。

```
cfg.DeepLearningConfig = dlcfg;
```

### 使用 codegen 生成 C++ 源代码

```
codegen -config cfg squeezenet_predict -args {ones(227, 227, 3, 'single')} -d arm_compute
```

代码在主机上当前工作文件夹的 `arm_compute` 文件夹中生成。

### 使用 packNGo 函数生成 Zip 文件

`packNGo` 函数将所有相关文件打包到一个压缩的 zip 文件中。

```
zipFileName = 'arm_compute.zip';
bInfo = load(fullfile('arm_compute','buildInfo.mat'));
packNGo(bInfo.buildInfo, {'fileName', zipFileName,'minimalHeaders', false, 'ignoreFileMissing',true});
```

代码生成为 zip 文件。

### 将生成的 Zip 文件复制到目标硬件

复制该 Zip 文件并提取到文件夹中，然后在硬件中删除 Zip 文件

在以下命令中，进行以下替换：

- 将 `password` 替换为您的密码
- 将 `username` 替换为您的用户名
- 将 `targetname` 替换为您设备的名称
- 将 `targetloc` 替换为文件的目标文件夹

执行以下步骤以从 Linux 复制和提取 zip 文件。

```
if isunix, system(['sshpass -p password scp -r ' fullfile(pwd,zipFileName) ' username@targetname:targetloc/']);
else
    if isunix, system('sshpass -p password ssh username@targetname "if [ -d targetloc/arm_compute ]; then rm -rf targetloc/arm_compute; fi"');
    else
        if iswin, system(['sshpass -p password scp -r ' fullfile(pwd,zipFileName) ' username@targetname:/targetloc/']);
    end
end
```

```
if isunix, system(['sshpass -p password ssh username@targetname "unzip targetloc/' zipFileName ' -d targetloc/arm_compute'']);
if isunix, system(['sshpass -p password ssh username@targetname "rm -rf targetloc' zipFileName "'']), end
```

执行以下步骤以从 Windows 复制和提取 zip 文件。

```
if ispc, system(['pscp.exe -pw password -r ' fullfile(pwd,zipFileName) ' username@targetname:targetloc/']), end
if ispc, system(['plink.exe -l username -pw password targetname "if [ -d targetloc/arm_compute ]; then rm -rf targetloc/arm_compute; fi'']), end
if ispc, system(['plink.exe -l username -pw password targetname "unzip targetloc/' zipFileName ' -d targetloc/arm_compute'']);
if ispc, system(['plink.exe -l username -pw password targetname "rm -rf targetloc' zipFileName "'']), end
```

### 将示例文件复制到目标硬件

将下列支持文件从主机复制到目标硬件：

- 输入图像， **coffeemug.png**
- 用于生成库的联编文件， **squeezeenet\_predict\_rtw.mk**
- 用于编译可执行程序的联编文件， **makefile\_squeezeenet\_arm\_generic.mk**
- Synset 字典， **synsetWords.txt**

在以下命令中，进行以下替换：

- 将 **password** 替换为您的密码
- 将 **username** 替换为您的用户名
- 将 **targetname** 替换为您设备的名称
- 将 **targetloc** 替换为文件的目标文件夹

从 Linux 运行时，执行以下步骤以复制所有必需的文件

```
if isunix, system('sshpass -p password scp squeezeenet_predict_rtw.mk username@targetname:targetloc/arm_compute');
if isunix, system('sshpass -p password scp coffeemug.png username@targetname:targetloc/arm_compute/'), end
if isunix, system('sshpass -p password scp makefile_squeezeenet_arm_generic.mk username@targetname:targetloc/arm_compute');
if isunix, system('sshpass -p password scp synsetWords.txt username@targetname:targetloc/arm_compute/'), end
```

从 Windows 运行时，执行以下步骤以复制所有必需的文件

```
if ispc, system(['pscp.exe -pw password squeezeenet_predict_rtw.mk username@targetname:targetloc/arm_compute']);
if ispc, system(['pscp.exe -pw password coffeemug.png username@targetname:targetloc/arm_compute/']), end
if ispc, system(['pscp.exe -pw password makefile_squeezeenet_arm_generic.mk username@targetname:targetloc/arm_compute']);
if ispc, system(['pscp.exe -pw password synsetWords.txt username@targetname:targetloc/arm_compute/']), end
```

### 在目标硬件上编译库

要在目标硬件上编译库，请在 ARM 硬件上执行生成的联编文件。

确保您在目标硬件上设置环境变量 **ARM\_COMPUTELIB** 和 **LD\_LIBRARY\_PATH**。请参阅“使用 MATLAB Coder 进行深度学习的前提条件”(MATLAB Coder)。在联编文件中使用 **ARM\_ARCH** 变量来基于 Arm 架构传递编译器标志。在联编文件中使用 **ARM\_VER** 变量来基于 Arm Compute 版本编译代码。按照与上述步骤类似的方式更换硬件凭据和路径。

执行以下步骤从 Linux 编译库。

```
if isunix, system('sshpass -p password scp main_squeezeenet_arm_generic.cpp username@targetname:targetloc/arm_compute');
if isunix, system(['sshpass -p password ssh username@targetname "make -C targetloc/arm_compute/ -f squeezeenet_predict_rtw.mk"']), end
```

执行以下步骤从 Windows 编译库。

```
if ispc, system('pscp.exe -pw password main_squeezeenet_arm_generic.cpp username@targetname:targetloc/arm_compute')
if ispc, system(['plink.exe -l username -pw password targetname "make -C targetloc/arm_compute/ -f squeezeenetc'])
```

### 从目标硬件上的库中创建可执行文件

用源主封装程序文件编译库来创建可执行文件。`main_squeezeenet_arm_generic.cpp` 是 C++ 主封装程序文件，它调用 `squeezeenet_predict` 函数来创建可执行文件。

运行以下命令以从 Linux 创建可执行文件。

```
if isunix, system('sshpass -p password ssh username@targetname "make -C targetloc/arm_compute/ -f makefile_sq')
```

运行以下命令以从 Windows 创建可执行文件。

```
if ispc, system('plink.exe -l username -pw password targetname "make -C targetloc/arm_compute/ -f makefile_sq')
```

### 在目标硬件上运行可执行文件

Run the executable from Linux using below command.

```
if isunix, system('sshpass -p password ssh username@targetname "cd targetloc/arm_compute/; ./squeezeenet coffeeen')
```

Run the executable from Windows using below command.

```
if ispc, system('plink.exe -l username -pw password targetname "cd targetloc/arm_compute/; ./squeezeenet coffeeen')
```

### Top 5 Predictions:

---

88.299% coffee mug

7.309% cup

1.098% candle

0.634% paper towel

0.591% water jug



# 通过 ARM 计算使用 **codegen** 进行深度学习预测

此示例说明如何使用 **codegen** 为在 ARM® 处理器上使用深度学习的徽标分类应用程序生成代码。徽标分类应用程序使用 **LogoNet** 串行网络从图像中执行徽标识别。生成的代码利用 ARM Compute Library 进行计算机视觉和机器学习。

## 前提条件

- 支持 NEON 扩展的 ARM 处理器
- 开源计算机视觉库 (OpenCV) v3.1
- ARM Compute 和 OpenCV 库的环境变量
- MATLAB® Coder™，用于生成 C++ 代码
- MATLAB Coder Interface for Deep Learning 支持包
- Deep Learning Toolbox™，在使用 **SeriesNetwork** 对象时会用到

此示例使用的 ARM Compute Library 版本可能不是代码生成支持的最新版本。有关受支持的库版本和有关设置环境变量的信息，请参阅“使用 MATLAB Coder 进行深度学习的前提条件”(MATLAB Coder)。

此示例在 Linux® 和 Windows® 平台上受支持，不受 MATLAB Online 支持。

## 获得预训练的 **SeriesNetwork**

下载经过预训练的 **LogoNet** 网络，并将其另存为 **logonet.mat** (如果它不存在的话)。该网络是在 MATLAB® 中开发的，其架构类似于 AlexNet 架构。该网络可以在各种光照条件和相机角度下识别 32 个徽标。

```
net = getLogonet();
```

该网络包含 22 个层，包括卷积层、全连接层和分类输出层。

```
net.Layers
```

```
ans =
```

```
22×1 Layer array with layers:
```

1 'imageinput'	Image Input	227×227×3 images with 'zerocenter' normalization and 'randfliplr' augmentation
2 'conv_1'	Convolution	96 5×5×3 convolutions with stride [1 1] and padding [0 0 0 0]
3 'relu_1'	ReLU	ReLU
4 'maxpool_1'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 0 0 0]
5 'conv_2'	Convolution	128 3×3×96 convolutions with stride [1 1] and padding [0 0 0 0]
6 'relu_2'	ReLU	ReLU
7 'maxpool_2'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 0 0 0]
8 'conv_3'	Convolution	384 3×3×128 convolutions with stride [1 1] and padding [0 0 0 0]
9 'relu_3'	ReLU	ReLU
10 'maxpool_3'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 0 0 0]
11 'conv_4'	Convolution	128 3×3×384 convolutions with stride [2 2] and padding [0 0 0 0]
12 'relu_4'	ReLU	ReLU
13 'maxpool_4'	Max Pooling	3×3 max pooling with stride [2 2] and padding [0 0 0 0]
14 'fc_1'	Fully Connected	2048 fully connected layer
15 'relu_5'	ReLU	ReLU
16 'dropout_1'	Dropout	50% dropout
17 'fc_2'	Fully Connected	2048 fully connected layer
18 'relu_6'	ReLU	ReLU

```

19 'dropout_2' Dropout      50% dropout
20 'fc_3' Fully Connected  32 fully connected layer
21 'softmax' Softmax       softmax
22 'classoutput' Classification Output crossentropyex with 'adidas' and 31 other classes

```

## 设置环境变量

在 ARM 目标硬件上，确保设置了 `ARM_COMPUTELIB`，并且 `LD_LIBRARY_PATH` 包含指向 ARM Compute Library 文件夹的路径。

请参阅“使用 MATLAB Coder 进行深度学习的前提条件”(MATLAB Coder)。

## `logonet_predict` 函数

`logonet_predict.m` 入口函数以图像作为输入，并使用保存在 `LogoNet` MAT 文件中的深度学习网络对图像执行预测。该函数将网络对象从 `LogoNet.mat` 加载到持久性网络变量 `logonet` 中。在对该函数的后续调用中，将重复使用该持久性对象。

```

type logonet_predict

function out = logonet_predict(in)
%#codegen

% Copyright 2017-2020 The MathWorks, Inc.

persistent logonet;

if isempty(logonet)

    logonet = coder.loadDeepLearningNetwork('LogoNet.mat','logonet');
end

out = logonet.predict(in);

end

```

## 为静态库设置代码生成配置对象

当您针对基于 ARM 的设备生成代码并且不使用硬件支持包时，请为库创建一个配置对象。不要为可执行程序创建配置对象。

为生成 C++ 代码和仅生成代码设置配置对象。

```

cfg = coder.config('lib');
cfg.TargetLang = 'C++';
cfg.GenCodeOnly = true;

```

## 为深度学习代码生成设置配置对象

创建一个 `coder.ARMNEONConfig` 对象。指定目标 ARM 处理器的库版本和架构。例如，假设目标板是具有 ARMv8 架构和 ARM Compute Library 19.05 版本的 HiKey/Rock960 板。

```

dlcfg = coder.DeepLearningConfig('arm-compute');
dlcfg.ArmComputeVersion = '19.05';
dlcfg.ArmArchitecture = 'armv8';

```

## 将深度学习配置对象连接到代码生成配置对象

将代码生成配置对象的 DeepLearningConfig 属性设置为深度学习配置对象。

```
cfg.DeepLearningConfig = dlcfg;
```

## 使用 codegen 生成 C++ 源代码

```
codegen -config cfg logonet_predict -args {ones(227, 227, 3, 'single')} -d arm_compute
```

代码在主机上当前工作文件夹的 arm\_compute 文件夹中生成。

## 使用 packNGo 函数生成 Zip 文件

packNGo 函数将所有相关文件打包到一个压缩的 zip 文件中。

```
zipFileName = 'arm_compute.zip';
bInfo = load(fullfile('arm_compute','buildInfo.mat'));
packNGo(bInfo.buildInfo, {'fileName', zipFileName,'minimalHeaders', false, 'ignoreFileMissing',true});
```

## 将生成的 Zip 文件复制到目标硬件

复制 Zip 文件并提取到一个文件夹中。从目标硬件中删除 Zip 文件。

在以下命令中，进行以下替换：

- 将 password 替换为您的密码
- 将 username 替换为您的用户名
- 将 targetname 替换为您设备的名称
- 将 targetloc 替换为文件的目标文件夹

运行以下命令从 Linux 中复制并提取 zip 文件。

```
if isunix, system(['sshpass -p password scp -r ' fullfile(pwd,zipFileName) ' username@targetname:targetloc/']), end
if isunix, system('sshpass -p password ssh username@targetname "if [ -d targetloc/arm_compute ]; then rm -rf targetloc/arm_compute; fi"'), end
if isunix, system(['sshpass -p password ssh username@targetname "unzip targetloc/' zipFileName ' -d targetloc/arm_compute"']), end
if isunix, system(['sshpass -p password ssh username@targetname "rm -rf targetloc' zipFileName '"']), end
```

运行以下命令从 Windows 中复制并提取 zip 文件。

```
if ispc, system(['pscp.exe -pw password -r ' fullfile(pwd,zipFileName) ' username@targetname:targetloc/']), end
if ispc, system('plink.exe -l username -pw password targetname "if [ -d targetloc/arm_compute ]; then rm -rf targetloc/arm_compute; fi"'), end
if ispc, system(['plink.exe -l username -pw password targetname "unzip targetloc/' zipFileName ' -d targetloc/arm_compute"']), end
if ispc, system(['plink.exe -l username -pw password targetname "rm -rf targetloc' zipFileName '"']), end
```

## 将示例文件复制到目标硬件

将下列支持文件从主机复制到目标硬件：

- 输入图像，coderdemo\_google.png
- 用于生成库的联编文件，logonet\_predict\_rtw.mk
- 用于编译可执行程序的联编文件，makefile\_arm\_logo.mk
- Synset 字典，synsetWordsLogoDet.txt

在以下命令中，进行以下替换：

- 将 **password** 替换为您的密码
- 将 **username** 替换为您的用户名
- 将 **targetname** 替换为您设备的名称
- 将 **targetloc** 替换为文件的目标文件夹

从 Linux 运行时，执行以下步骤以复制所有必需的文件

```
if isunix, system('sshpass -p password scp logonet_predict_rtw.mk username@targetname:targetloc/arm_compute')
if isunix, system('sshpass -p password scp coderdemo_google.png username@targetname:targetloc/arm_compute')
if isunix, system('sshpass -p password scp makefile_arm_logo.mk username@targetname:targetloc/arm_compute')
if isunix, system('sshpass -p password scp synsetWordsLogoDet.txt username@targetname:targetloc/arm_compute')
```

从 Windows 运行时，执行以下步骤以复制所有必需的文件

```
if ispc, system('pscp.exe -pw password logonet_predict_rtw.mk username@targetname:targetloc/arm_compute')
if ispc, system('pscp.exe -pw password coderdemo_google.png username@targetname:targetloc/arm_compute')
if ispc, system('pscp.exe -pw password makefile_arm_logo.mk username@targetname:targetloc/arm_compute')
if ispc, system('pscp.exe -pw password synsetWordsLogoDet.txt username@targetname:targetloc/arm_compute')
```

### 在目标硬件上编译库

要在目标硬件上编译库，请在 ARM 硬件上执行生成的联编文件。

确保您在目标硬件上设置环境变量 **ARM\_COMPUTELIB** 和 **LD\_LIBRARY\_PATH**。请参阅“使用 MATLAB Coder 进行深度学习的前提条件”(MATLAB Coder)。在联编文件中使用 **ARM\_ARCH** 变量来基于 Arm 架构传递编译器标志。在联编文件中使用 **ARM\_VER** 变量来基于 Arm Compute 版本编译代码。按照与上一节中类似的步骤替换下列命令中的硬件凭据和路径。

执行以下步骤从 Linux 编译库。

```
if isunix, system('sshpass -p password scp main_arm_logo.cpp username@targetname:targetloc/arm_compute')
if isunix, system(['sshpass -p password ssh username@targetname "make -C targetloc/arm_compute/ -f logonet_p'])
```

执行以下步骤从 Windows 编译库。

```
if ispc, system('pscp.exe -pw password main_arm_logo.cpp username@targetname:targetloc/arm_compute')
if ispc, system(['plink.exe -l username -pw password targetname "make -C targetloc/arm_compute/ -f logonet_p'])
```

### 从目标硬件上的库中创建可执行文件

用源主封装程序文件编译库来创建可执行文件。**main\_arm\_logo.cpp** 是调用 **logonet\_predict** 函数的 C++ 主封装程序文件。

运行以下命令以从 Linux 创建可执行文件。

```
if isunix, system('sshpass -p password ssh username@targetname "make -C targetloc/arm_compute/ -f makefile_a')
```

运行以下命令以从 Windows 创建可执行文件。

```
if ispc, system('plink.exe -l username -pw password targetname "make -C targetloc/arm_compute/ -f makefile_a")
```

### 在目标硬件上运行可执行文件

Run the executable from Linux using below command.

```
if isunix, system('sshpass -p password ssh username@targetname "cd targetloc/arm_compute/; ./logonet coderden")
```

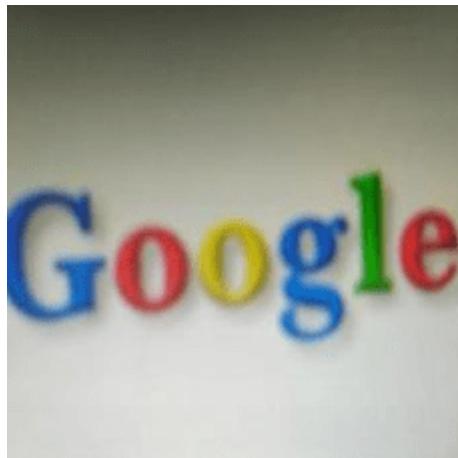
Run the executable from Windows using below command.

```
if ispc, system('plink.exe -l username -pw password targetname "cd targetloc/arm_compute/; ./logonet coderdemo
```

**Top 5 Predictions:**

---

```
99.992% google
0.003% corona
0.003% singha
0.001% esso
0.000% fedex
```



## 针对不同批量大小在 Intel 目标上进行深度学习代码生成

此示例说明如何使用 `codegen` 命令为在 Intel® 处理器上使用深度学习的图像分类应用程序生成代码。生成的代码使用了用于深度神经网络的 Intel 数学核心函数库 (MKL-DNN)。此示例由两部分组成：

- 第一部分说明如何生成接受一批图像作为输入的 MEX 函数。
- 第二部分说明如何生成接受一批图像作为输入的可执行文件。

### 前提条件

- 支持 Intel Advanced Vector Extensions 2 (Intel AVX2) 指令的 Intel 处理器
- 用于深度神经网络的 Intel 数学核心函数库 (MKL-DNN)
- 编译器和库的环境变量。有关支持的编译器版本的信息，请参阅支持的编译器。有关设置环境变量的信息，请参阅“使用 MATLAB Coder 进行深度学习的前提条件” (MATLAB Coder)。

此示例可在 Linux®、Windows® 和 Mac® 平台上运行，但不支持在 MATLAB Online 中运行。

### 下载输入视频文件

下载示例视频文件。

```
if ~exist('./object_class.avi', 'file')
    url = 'https://www.mathworks.com/supportfiles/gpucoder/media/object_class.avi.zip';
    websave('object_class.avi.zip',url);
    unzip('object_class.avi.zip');
end
```

### 定义 resnet\_predict 函数

此示例在 Intel 桌面上使用 DAG 网络 ResNet-50 显示图像分类。Deep Learning Toolbox Model for ResNet-50 Network 支持包是适用于 MATLAB 的预训练 ResNet-50 模型的一部分。

`resnet_predict` 函数将 ResNet-50 网络加载到一个持久性网络对象中，然后对输入执行预测。对该函数的后续调用会重用该持久性网络对象。

```
type resnet_predict

% Copyright 2020 The MathWorks, Inc.

function out = resnet_predict(in)
%#codegen

% A persistent object mynet is used to load the series network object.
% At the first call to this function, the persistent object is constructed and
% setup. When the function is called subsequent times, the same object is reused
% to call predict on inputs, avoiding reconstructing and reloading the
% network object.

persistent mynet;

if isempty(mynet)
    % Call the function resnet50 that returns a DAG network
    % for ResNet-50 model.
    mynet = coder.loadDeepLearningNetwork('resnet50','resnet');
end
```

```
% pass in input
out = mynet.predict(in);
```

### 为 resnet\_predict 生成 MEX

要为 `resnet_predict` 函数生成 MEX 函数, 请将 `codegen` 与针对 MKL-DNN 库的深度学习配置对象结合使用。将该深度学习配置对象附加到传递给 `codegen` 的 MEX 代码生成配置对象。运行 `codegen` 命令, 并将输入指定为 [224,224,3,|batchSize|] 大小的四维矩阵。该值对应于 ResNet-50 网络的输入层大小。

```
batchSize = 5;
cfg = coder.config('mex');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('mkldnn');
codegen -config cfg resnet_predict -args {ones(224,224,3,batchSize,'single')} -report
```

Code generation successful: To view the report, open('codegen\mex\resnet\_predict\html\report.mldatx').

### 对一批图像执行预测

假设已下载 `Object_class.avi` 视频文件。创建 `videoReader` 对象, 并使用 `videoReader` 读取函数读取五个帧。由于 `batchSize` 设置为 5, 因此读取 5 个图像。将该批输入图像的大小调整为 `resnet50` 所需的大小, 即 `ResNet50` 网络预期的大小。

```
videoReader = VideoReader('Object_class.avi');
imBatch = read(videoReader,[1 5]);
imBatch = imresize(imBatch, [224,224]);
```

调用生成的 `resnet_predict_mex` 函数, 该函数为您提供提供的输入输出分类结果。

```
predict_scores = resnet_predict_mex(single(imBatch));
```

获取批量图像中每个图像的前 5 个概率分数及其标签。

```
[val,indx] = sort(transpose(predict_scores), 'descend');
scores = val(1:5,:)*100;
net = resnet50;
classnames = net.Layers(end).ClassNames;
for i = 1:batchSize
    labels = classnames(indx(1:5,i));
    disp(['Top 5 predictions on image, ', num2str(i)]);
    for j=1:5
        disp([labels{j},',',num2str(scores(j,i), '%2.2f'),','])
    end
end
```

将针对第一个图像的前五个预测分数映射到 `synset` 字典中的单词。

```
fid = fopen('synsetWords.txt');
synsetOut = textscan(fid,'%s', 'delimiter', '\n');
synsetOut = synsetOut{1};
fclose(fid);
[val,indx] = sort(transpose(predict_scores), 'descend');
scores = val(1:5,1)*100;
top5labels = synsetOut(indx(1:5,1));
```

在图像上显示排名前五的分类标签。

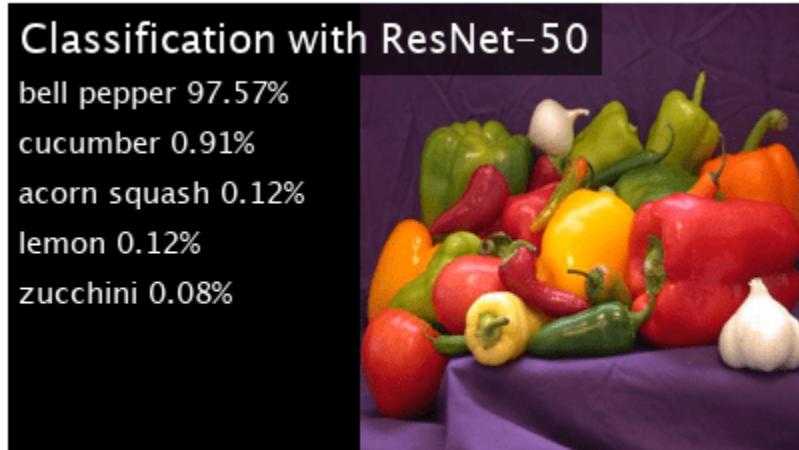
```

outputImage = zeros(224,400,3, 'uint8');
for k = 1:3
    outputImage(:,177:end,k) = imBatch(:,:,k,1);
end

scol = 1;
srow = 1;
outputImage = insertText(outputImage, [scol, srow], 'Classification with ResNet-50', 'TextColor', 'w','FontSize',14);
srow = srow + 30;
for k = 1:5
    outputImage = insertText(outputImage, [scol, srow], [top5labels{k}, ' ',num2str(scores(k), '%2.2f'), '%'], 'TextColor', 'w','FontSize',12);
    srow = srow + 25;
end

imshow(outputImage);

```



从内存中清除持久性网络对象。

```
clear mex;
```

#### 定义 resnet\_predict\_exe 入口函数

要从 MATLAB 代码生成可执行文件，请定义一个新的入口函数 `resnet_predict_exe`。此函数与之前的入口函数 `resent_predict` 相似，但还包含用于预处理和后处理的代码。`resnet_predict_exe` 使用的 API 与平台无关。此函数接受视频和批量大小作为输入参数。这些参数是编译时常量。

```
type resnet_predict_exe
```

```
% Copyright 2020 The MathWorks, Inc.
```

```
function resnet_predict_exe(inputVideo,batchSize)
 %#codegen
```

```
% A persistent object mynet is used to load the series network object.
```

```
% At the first call to this function, the persistent object is constructed and
% setup. When the function is called subsequent times, the same object is reused
% to call predict on inputs, avoiding reconstructing and reloading the
% network object.
persistent mynet;

if isempty(mynet)
    % Call the function resnet50 that returns a DAG network
    % for ResNet-50 model.
    mynet = coder.loadDeepLearningNetwork('resnet50','resnet');
end

% Create video reader and video player objects %
videoReader = VideoReader(inputVideo);
depVideoPlayer = vision.DeployableVideoPlayer;

% Read the classification label names %
synsetOut = readImageClassLabels('synsetWords.txt');

i=1;
% Read frames until end of video file %
while ~ (i+batchSize > (videoReader.NumFrames+1))
    % Read and resize batch of frames as specified by input argument%
    reSizedImagesBatch = readImageInputBatch(videoReader,batchSize,i);

    % run predict on resized input images %
    predict_scores = mynet.predict(reSizedImagesBatch);

    % overlay the prediction scores on images and display %
    overlayResultsOnImages(predict_scores,synsetOut,reSizedImagesBatch,batchSize,depVideoPlayer)

    i = i+ batchSize;
end
release(depVideoPlayer);
end

function synsetOut = readImageClassLabels(classLabelsFile)
% Read the classification label names from the file
%
% Inputs :
% classLabelsFile - supplied by user
%
% Outputs :
% synsetOut      - cell array filled with 1000 image class labels

synsetOut = cell(1000,1);
fid = fopen(classLabelsFile);
for i = 1:1000
    synsetOut{i} = fgetl(fid);
end
fclose(fid);
end

function reSizedImagesBatch = readImageInputBatch(videoReader,batchSize,i)
% Read and resize batch of frames as specified by input argument%
%
```

```
% Inputs :
% videoReader - Object used for reading the images from video file
% batchSize - Number of images in batch to process. Supplied by user
% i - index to track frames read from video file
%
% Outputs :
% reSizedImagesBatch - Batch of images resized to 224x224x3xbatchsize

img = read(videoReader,[i (i+batchSize-1)]);
reSizedImagesBatch = coder.nullcopy(ones(224,224,3,batchSize,'like',img));
resizeTo = coder.const([224,224]);
reSizedImagesBatch(:,:,:,:) = imresize(img,resizeTo);
end

function overlayResultsOnImages(predict_scores,synsetOut,reSizedImagesBatch,batchSize,depVideoPlayer)
% Read and resize batch of frames as specified by input argument%
%
% Inputs :
% predict_scores - classification results for given network
% synsetOut - cell array filled with 1000 image class labels
% reSizedImagesBatch - Batch of images resized to 224x224x3xbatchsize
% batchSize - Number of images in batch to process. Supplied by user
% depVideoPlayer - Object for displaying results
%
% Outputs :
% Predicted results overlayed on input images

% sort the predicted scores %
[val,indx] = sort(transpose(predict_scores), 'descend');

for j = 1:batchSize
    scores = val(1:5,j)*100;
    outputImage = zeros(224,400,3, 'uint8');
    for k = 1:3
        outputImage(:,177:end,k) = reSizedImagesBatch(:,:,k,j);
    end

    % Overlay the results on image %
    scol = 1;
    srow = 1;
    outputImage = insertText(outputImage, [scol, srow], 'Classification with ResNet-50', 'TextColor', [255 255 255]);
    srow = srow + 30;
    for k = 1:5
        scoreStr = sprintf('%2.2f',scores(k));
        outputImage = insertText(outputImage, [scol, srow], [synsetOut{indx(k,j)} ',scoreStr,'%'], 'TextColor', [255 255 255]);
        srow = srow + 25;
    end

    depVideoPlayer(outputImage);
end
end
```

### resnet\_predict\_exe 函数的结构体

函数 resnet\_predict\_exe 包含四个子部分，分别用于执行下列操作：

- 从提供的输入文本文件中读取分类标签
- 读取输入的一批图像，并根据网络需要调整其大小
- 对输入图像批量运行推断
- 在图像上叠加结果

有关上述每个步骤的详细信息，请参阅后续章节。

### readImageClassLabels 函数

此函数接受 `synsetWords.txt` 文件作为输入参数。它读取分类标签并填充元胞数组。

```
function synsetOut = readImageClassLabels(classLabelsFile)
% Read the classification label names from the file
%
% Inputs :
% classLabelsFile - supplied by user
%
% Outputs :
% synsetOut      - cell array filled with 1000 image class labels

synsetOut = cell(1000,1);
fid = fopen(classLabelsFile);
for i = 1:1000
    synsetOut{i} = fgetl(fid);
end
fclose(fid);
end
```

### readImageInputBatch 函数

此函数从作为输入参数传递给函数的视频输入文件中读取图像并调整其大小。它读取指定的输入图像，并将其大小调整为  $224 \times 224 \times 3$ ，这是 resnet50 网络预期的大小。

```
function reSizedImagesBatch = readImageInputBatch(videoReader,batchSize,i)
% Read and resize batch of frames as specified by input argument%
%
% Inputs :
% videoReader - Object used for reading the images from video file
% batchSize - Number of images in batch to process. Supplied by user
% i          - index to track frames read from video file
%
% Outputs :
% reSizedImagesBatch - Batch of images resized to 224x224x3xbatchsize

img = read(videoReader,[i (i+batchSize-1)]);
reSizedImagesBatch = coder.nullcopy(ones(224,224,3,batchSize,'like',img));
resizeTo = coder.const([224,224]);
reSizedImagesBatch(:,:,:,:) = imresize(img,resizeTo);
end
```

### mynet.predict 函数

此函数接受调整大小后的图像批量作为输入，并返回预测结果。

```
% run predict on resized input images %
predict_scores = mynet.predict(reSizedImagesBatch);
```

### overlayResultsOnImages 函数

此函数接受预测结果，并按降序对它们进行排序。它在输入图像上叠加这些结果并显示它们。

```
function overlayResultsOnImages(predict_scores,synsetOut,reSizedImagesBatch,batchSize,depVideoPlayer)
% Read and resize batch of frames as specified by input argument%
%
% Inputs :
% predict_scores - classification results for given network
% synsetOut      - cell array filled with 1000 image class labels
% reSizedImagesBatch - Batch of images resized to 224x224x3xbatchsize
% batchSize       - Number of images in batch to process. Supplied by user
% depVideoPlayer - Object for displaying results
%
% Outputs :
% Predicted results overlayed on input images

% sort the predicted scores %
[val,indx] = sort(transpose(predict_scores), 'descend');

for j = 1:batchSize
    scores = val(1:5,j)*100;
    outputImage = zeros(224,400,3, 'uint8');
    for k = 1:3
        outputImage(:,177:end,k) = reSizedImagesBatch(:,:,k,j);
    end

    % Overlay the results on image %
    scol = 1;
    srow = 1;
    outputImage = insertText(outputImage, [scol, srow], 'Classification with ResNet-50', 'TextColor', [255 255 255]);
    srow = srow + 30;
    for k = 1:5
        scoreStr = sprintf('%2.2f',scores(k));
        outputImage = insertText(outputImage, [scol, srow], [synsetOut{indx(k,j)}], scoreStr, 'TextColor');
        srow = srow + 25;
    end

    depVideoPlayer(outputImage);
end
end
```

### 编译和运行可执行文件

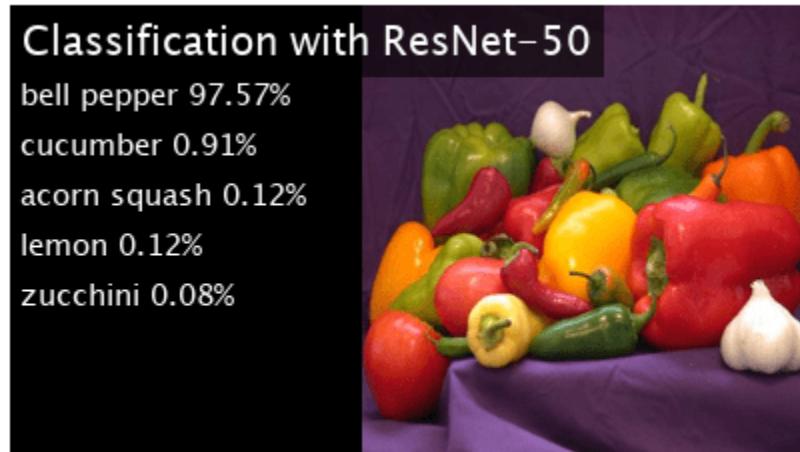
创建一个用于生成可执行文件的代码配置对象。将一个深度学习配置对象与之关联。设置 `batchSize` 和 `inputVideoFile` 变量。

如果不打算创建自定义 C++ 主函数，而是使用生成的示例 C++ 主函数，请将 `GenerateExampleMain` 参数设置为 '`GenerateCodeAndCompile`'。此外，禁用 `cfg.EnableOpenMP` 以确保在从桌面终端运行可执行文件时不存在对 openmp 库的依存关系。

```
cfg = coder.config('exe');
cfg.TargetLang = 'C++';
cfg.DeepLearningConfig = coder.DeepLearningConfig('mkldnn');
batchSize = 5;
inputVideoFile = 'object_class.avi';
cfg.GenerateExampleMain = 'GenerateCodeAndCompile';
cfg.EnableOpenMP = 0;
```

运行 `codegen` 命令以编译可执行文件。在 MATLAB 命令行中或在桌面终端上运行生成的可执行文件 `resnet_predict_exe`。

```
codegen -config cfg resnet_predict_exe -args {coder.Constant(inputVideoFile), coder.Constant(batchSize)} -r  
system('./resnet_predict_exe')
```



## 另请参阅

### 相关示例

- “在 MATLAB 中进行深度学习”（第 1-2 页）



# 神经网络设计书籍

Deep Learning Toolbox 软件的开发人员编写了一本教科书，《Neural Network Design》(Hagan, Demuth, and Beale, ISBN 0-9717321-0-8)。这本书介绍了神经网络的理论，讨论了其设计和应用，并大量使用 MATLAB 环境和 Deep Learning Toolbox 软件。本文档在不同部分都引用了该书中的示例程序。  
(通过键入 `nnd`，您可以在 Deep Learning Toolbox 软件中找到该书的所有示例程序。)

要获取该书，请致电 (303) 492-3648 联系 John Stovall，或向 [John.Stovall@colorado.edu](mailto:John.Stovall@colorado.edu) 发送电子邮件。

这本《Neural Network Design》教科书包括：

- 为将本书用作课堂教材的人员编写的教师手册
- 供班级教学使用的概略文档

如果您想将本书用于班级授课并希望获得教师手册（包括书中练习的答案），请致电 (303) 492-3648 联系 John Stovall，或向 [John.Stovall@colorado.edu](mailto:John.Stovall@colorado.edu) 发送电子邮件。

要查看该书的示例章节并获得概略文档，请直接访问该神经网络设计页面，请使用以下网址：

<https://hagan.okstate.edu/nnd.html>

从这个链接，您可以获得该书的 PDF 格式的示例章节，还可以通过点击 **Transparency Masters (3.6MB)** 下载概略文档。

您可以获得 PowerPoint 或 PDF 格式的概略文档。



# 神经网络对象、数据和训练风格

---

- “神经网络设计的工作流” (第 20-2 页)
- “创建神经网络对象” (第 20-3 页)
- “配置浅层神经网络输入和输出” (第 20-6 页)

## 神经网络设计的工作流

神经网络设计过程的工作流包含七个主要步骤。参考主题讨论了步骤 2、3 和 5 背后的基本思想。

- 1 收集数据
- 2 创建网络 - “创建神经网络对象” (第 20-3 页)
- 3 配置网络 - “配置浅层神经网络输入和输出” (第 20-6 页)
- 4 初始化权重和偏置
- 5 训练网络 - “Neural Network Training Concepts”
- 6 验证网络
- 7 使用网络

步骤 1 中的数据采集通常在 Deep Learning Toolbox 软件的框架之外进行，但在“多层浅层神经网络与反向传播训练”(第 21-2 页)中对其进行了一般性讨论。其他步骤的详细信息以及步骤 4、6 和 7 的讨论在特定于网络类型的主题中进行说明。

Deep Learning Toolbox 软件使用网络对象来存储用于定义神经网络的所有信息。本主题介绍神经网络的基本组件，并展示如何创建这些如今以及如何将它们存储在网络对象中。

在创建神经网络后，需要对其进行配置，然后进行训练。配置包括安排网络使其与您要求解的问题兼容，如示例数据所定义的那样。在网络配置完成后，需要调整可调网络参数（称为权重和偏置），以便优化网络性能。此调整过程称为网络训练。配置和训练要求向网络提供示例数据。本主题说明如何格式化数据以提交给网络。它还介绍了网络配置和两种形式的网络训练：增量训练和批量训练。

## 另请参阅

### 详细信息

- “Four Levels of Neural Network Design”
- “Neuron Model”
- “Neural Network Architectures”
- “Understanding Shallow Network Data Structures”

## 创建神经网络对象

本主题是“神经网络设计的工作流”（第 20-2 页）中所述的设计工作流的一部分。

创建神经网络的最简单方法是使用一个网络创建函数。为了研究如何做到这一点，您可以使用命令 **feedforwardnet** 创建一个简单的两层前馈网络：

```
net = feedforwardnet  
  
net =  
  
Neural Network  
  
    name: 'Feed-Forward Neural Network'  
    userdata: (your custom info)  
  
dimensions:  
  
    numInputs: 1  
    numLayers: 2  
    numOutputs: 1  
    numInputDelays: 0  
    numLayerDelays: 0  
    numFeedbackDelays: 0  
    numWeightElements: 10  
    sampleTime: 1  
  
connections:  
  
    biasConnect: [1; 1]  
    inputConnect: [1; 0]  
    layerConnect: [0 0; 1 0]  
    outputConnect: [0 1]  
  
subobjects:  
  
    inputs: {1x1 cell array of 1 input}  
    layers: {2x1 cell array of 2 layers}  
    outputs: {1x2 cell array of 1 output}  
    biases: {2x1 cell array of 2 biases}  
    inputWeights: {2x1 cell array of 1 weight}  
    layerWeights: {2x2 cell array of 1 weight}  
  
functions:  
  
    adaptFcn: 'adaptwb'  
    adaptParam: (none)  
    derivFcn: 'defaultderiv'  
    divideFcn: 'dividerand'  
    divideParam: .trainRatio, .valRatio, .testRatio  
    divideMode: 'sample'  
    initFcn: 'initlay'  
    performFcn: 'mse'  
    performParam: .regularization, .normalization  
    plotFcn: {'plotperform', plottrainstate, ploterrhist,  
              plotregression}  
    plotParams: {1x4 cell array of 4 params}
```

```

trainFcn: 'trainlm'
trainParam: .showWindow, .showCommandLine, .show, .epochs,
    .time, .goal, .min_grad, .max_fail, .mu, .mu_dec,
    .mu_inc, .mu_max

```

weight and bias values:

```

IW: {2x1 cell} containing 1 input weight matrix
LW: {2x2 cell} containing 1 layer weight matrix
b: {2x1 cell} containing 2 bias vectors

```

methods:

```

adapt: Learn while in continuous use
configure: Configure inputs & outputs
gensim: Generate Simulink model
init: Initialize weights & biases
perform: Calculate performance
sim: Evaluate network outputs given inputs
train: Train network with examples
view: View diagram
unconfigure: Unconfigure inputs & outputs

evaluate: outputs = net(inputs)

```

此处显示的内容简要说明了网络对象，它用于存储定义神经网络的所有信息。此处有很多细节，但有几个关键部分可以帮助您了解网络对象是如何组织的。

维度部分存储网络的整体结构。在此处，您可以看到一个网络输入（尽管一个输入可以是包含许多元素的向量）、一个网络输出和两个层。

连接部分存储网络组件之间的连接。例如，每一层都连接了一个偏置，输入连接到层 1，输出来自层 2。您还可以看到层 1 连接到层 2。（`net.layerConnect` 的行表示目标层，列表示源层。此矩阵中的 1 表示有连接，0 表示无连接。对于此示例，矩阵的元素 2,1 处有一个 1。）

网络对象的关键子对象包括 `inputs`、`layers`、`outputs`、`biases`、`inputWeights` 和 `layerWeights`。使用以下命令查看第一层的 `layers` 子对象

```
net.layers{1}
```

Neural Network Layer

```

name: 'Hidden'
dimensions: 10
distanceFcn: (none)
distanceParam: (none)
distances: []
initFcn: 'initnw'
netInputFcn: 'netsum'
netInputParam: (none)
positions: []
range: [10x2 double]
size: 10
topologyFcn: (none)
transferFcn: 'tansig'
transferParam: (none)
userdata: (your custom info)

```

一个层中神经元的数量由其 `size` 属性给出。在本例中，该层有 10 个神经元，这是 `feedforwardnet` 命令的默认大小。`net` 输入函数是 `netsum`（求和），传递函数是 `tansig`。例如，如果您要将传递函数更改为您 `logsig`，可以执行以下命令：

```
net.layers{1}.transferFcn = 'logsig';
```

要查看 `layerWeights` 子对象以了解层 1 和层 2 之间的权重，请使用以下命令：

```
net.layerWeights{2,1}
```

Neural Network Weight

```
delays: 0
initFcn: (none)
initConfig: .inputSize
learn: true
learnFcn: 'learngdm'
learnParam: .lr, .mc
size: [0 10]
weightFcn: 'dotprod'
weightParam: (none)
userdata: (your custom info)
```

权重函数是 `dotprod`，它表示标准矩阵乘法（点积）。请注意，该层权重的大小为  $0 \times 10$ 。我们有全部为 0 的行，是因为网络尚未针对特定数据集进行配置。输出神经元的数量等于目标向量中的行数。在配置过程中，您将为网络提供示例输入和目标，然后可以分配输出神经元的数量。

这可以让您大致了解网络对象是如何组织的。对于许多应用，您不需要直接对网络对象进行更改，因为这由网络创建函数负责。通常仅当您要覆盖系统默认值时，才需要直接访问网络对象。其他主题将说明如何针对特定网络和训练方法做到这一点。

要更详细地研究网络对象，您可能会发现对象列表（如上所示）包含每个子对象的帮助链接。点击链接，您可以有选择地调查对象中您关注的部分。

## 配置浅层神经网络输入和输出

本主题是“神经网络设计的工作流”（第 20-2 页）中所述的设计工作流的一部分。

在创建神经网络后，必须对其进行配置。配置步骤包括检查输入和目标数据，设置网络的输入和输出大小以匹配数据，以及选择能够实现最佳网络性能的输入和输出处理设置。配置步骤通常在调用训练函数时自动完成。但也可以使用配置函数手动完成配置步骤。例如，要将之前创建的网络配置为逼近一个正弦函数，请发出以下命令：

```
p = -2:1:2;
t = sin(pi*p/2);
net1 = configure(net,p,t);
```

您已为网络提供一组示例输入和目标（所需的网络输出）。使用这些信息，**configure** 函数可以设置网络输入和输出大小以匹配数据。

配置完成后，如果您再次查看层 1 和层 2 之间的权重，可以看到权重的维度是  $1 \times 20$ 。这是因为该网络的目标是标量。

```
net1.layerWeights{2,1}
```

### Neural Network Weight

```
delays: 0
initFcn: (none)
initConfig: .inputSize
learn: true
learnFcn: 'learngdm'
learnParam: .lr, .mc
size: [1 10]
weightFcn: 'dotprod'
weightParam: (none)
userdata: (your custom info)
```

除了为权重设置合适的维度以外，配置步骤还定义输入和输出处理的设置。输入处理可以位于 **inputs** 子对象中：

```
net1.inputs{1}
```

### Neural Network Input

```
name: 'Input'
feedbackOutput: []
processFcns: {'removeconstantrows', mapminmax}
processParams: {1x2 cell array of 2 params}
processSettings: {1x2 cell array of 2 settings}
processedRange: [1x2 double]
processedSize: 1
range: [1x2 double]
size: 1
userdata: (your custom info)
```

在将输入应用于网络之前，它将由以下两个函数进行处理：**removeconstantrows** 和 **mapminmax**。这些内容在“多层次浅层神经网络与反向传播训练”（第 21-2 页）中进行了充分讨论，此处不再赘述。这些处理函数可能有一些处理参数，这些参数包含在子对象 **net1.inputs{1}.processParam** 中。这些参数都有可以覆盖的默认值。处理函数还可以根据样本数据进行配置设置。这些设置包含在

`net1.inputs{1}.processSettings` 中，并在配置过程中进行设置。例如，`mapminmax` 处理函数对数据进行归一化，使所有输入都在  $[-1, 1]$  范围内。其配置设置包括样本数据中的最小值和最大值，它需要这些值来执行正确的归一化。这将在“多层浅层神经网络与反向传播训练”（第 21-2 页）中进行更深入的讨论。

一般来说，我们使用“参数”（例如在过程参数、训练参数等表述中）一词来表示具有默认值的常量，这些默认值在创建网络时由软件赋予（您可以覆盖这些默认值）。我们使用“配置设置”（例如在过程配置设置中）表示由软件从样本数据分析中指定的常量。这些设置没有默认值，通常不应覆盖。

有关详细信息，另请参阅“Understanding Shallow Network Data Structures”。



# 多层浅层神经网络与反向传播训练

---

- “多层次神经网络与反向传播训练”（第 21-2 页）
- “多层次神经网络架构”（第 21-3 页）
- “选择神经网络输入输出处理函数”（第 21-6 页）
- “划分数据以实现最优神经网络训练”（第 21-8 页）
- “创建、配置和初始化多层次神经网络”（第 21-9 页）
- “训练与应用多层次神经网络”（第 21-11 页）
- “分析训练后的浅层神经网络性能”（第 21-15 页）

## 多层浅层神经网络与反向传播训练

浅层多层前馈神经网络可用于函数拟合和模式识别问题。通过增加抽头延迟线，它还可用于预测问题，如“Design Time Series Time-Delay Neural Networks”中所述。本主题说明如何使用多层网络。它还说明设计任何神经网络的基本过程。

**注意** 本主题中所述的训练函数不限于在多层网络中使用。它们可用于训练任意架构（甚至自定义网络），前提是网络架构的组件可区分。

---

一般神经网络设计过程的工作流包含七个主要步骤：

- 1 收集数据
- 2 创建网络
- 3 配置网络
- 4 初始化权重和偏置
- 5 训练网络
- 6 验证网络（训练后的分析）
- 7 使用网络

步骤 1 可能发生在 Deep Learning Toolbox 软件的框架之外，但此步骤对设计过程的成功至关重要。

此工作流的详细信息将在以下章节中讨论：

- “多层次浅层神经网络架构”（第 21-3 页）
- “Prepare Data for Multilayer Shallow Neural Networks”
- “创建、配置和初始化多层次浅层神经网络”（第 21-9 页）
- “训练与应用多层次浅层神经网络”（第 21-11 页）
- “分析训练后的浅层神经网络性能”（第 21-15 页）
- “使用网络”（第 21-14 页）
- “Limitations and Cautions”

可选的工作流步骤将在以下各节中讨论：

- “选择神经网络输入输出处理函数”（第 21-6 页）
- “划分数据以实现最优神经网络训练”（第 21-8 页）
- “使用并行和 GPU 计算的浅层神经网络”（第 27-2 页）

对于时序、动态建模和预测的信息，请参阅本节：

- “How Dynamic Neural Networks Work”

# 多层浅层神经网络架构

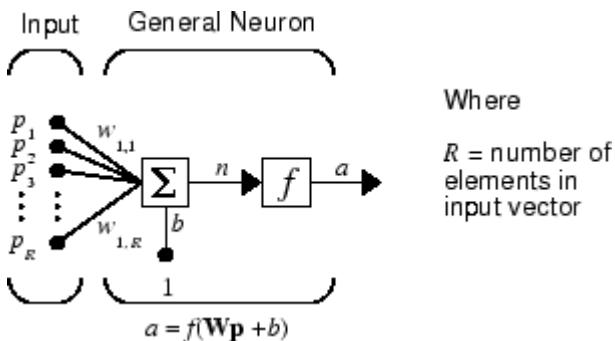
## 本节内容

- “神经元模型 (logsig、tansig、purelin) ” (第 21-3 页)
- “前馈神经网络” (第 21-4 页)

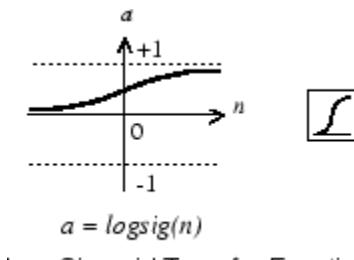
本主题介绍典型多层浅层网络工作流的一部分。有关详细信息和其他步骤，请参阅“多层浅层神经网络与反向传播训练”(第 21-2 页)。

## 神经元模型 (logsig、tansig、purelin)

一个具有 R 个输入的基本神经元如下所示。每个输入都用适当的 w 进行加权。加权输入和偏置之和构成传递函数 f 的输入。神经元可以使用任何可微分的传递函数 f 来产生其输出。

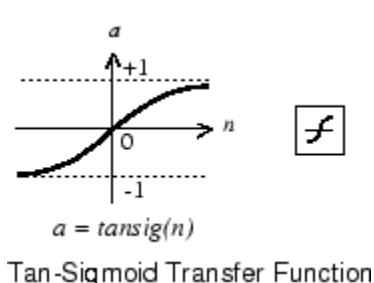


多层网络经常使用 log-sigmoid 传递函数 logsig。

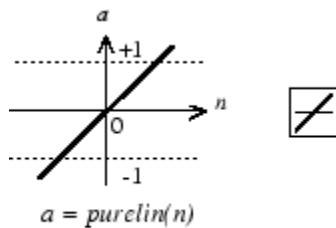


当神经元的净输入从负无穷大变为正无穷大时，函数 logsig 生成 0 到 1 之间的输出。

多层网络也可以使用 tan-sigmoid 传递函数 tansig。



Sigmoid 输出神经元经常用于模式识别问题，而线性输出神经元用于函数拟合问题。线性传递函数 **purelin** 如下所示。

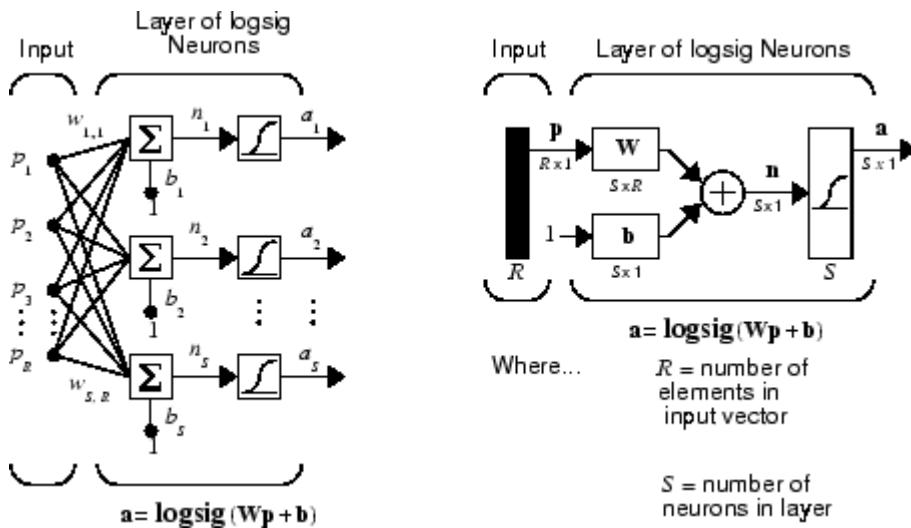


Linear Transfer Function

此处所述的三个传递函数是多层网络中最常用的传递函数，但也可以根据需要创建和使用其他可微分的传递函数。

## 前馈神经网络

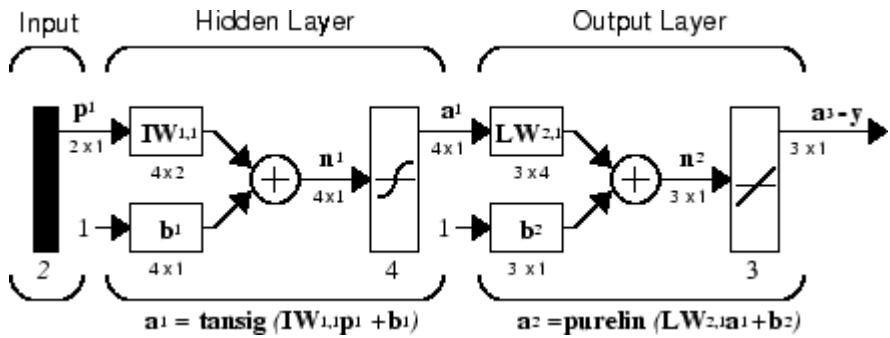
下图左侧详细显示了具有  $R$  个输入的由  $S$  logsig 神经元组成的单层网络，右侧为层次图。



前馈网络通常有一个或多个由 sigmoid 神经元组成的隐藏层，后跟一个由线性神经元组成的输出层。具有非线性传递函数的由神经元组成的多个层允许网络学习输入和输出向量之间的非线性关系。线性输出层最常用于函数拟合（或非线性回归）问题。

另一方面，如果您要约束网络的输出（例如在 0 和 1 之间），则输出层应使用 sigmoid 传递函数（例如 logsig）。当网络用于模式识别问题时就是这种情况（在这种情况下，由网络作出决定）。

对于多层网络，层编号决定权重矩阵中的上标。下面显示的两层 tansig/purelin 网络中使用了适当的表示法。



该网络可用作通用函数逼近器。只要隐藏层中有足够多的神经元，它就可以逼近任何具有任意有限数量的不连续点的函数。

现在已定义多层网络的架构，下面几节将介绍设计过程。

## 选择神经网络输入输出处理函数

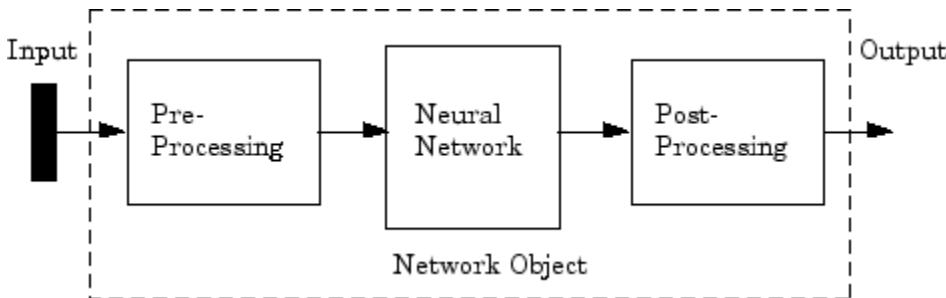
本主题介绍典型多层网络工作流的一部分。有关详细信息和其他步骤，请参阅“[多层浅层神经网络与反向传播训练](#)”（第 21-2 页）。

如果对网络输入和目标执行某些预处理步骤，神经网络训练会更高效。本节介绍几个您可以使用的预处理例程。（其中最常见的预处理例程是在您创建网络时自动提供的，它们成为网络对象的一部分，因此无论何时使用网络，进入网络的数据都会以相同的方式进行预处理。）

例如，在多层网络中，`sigmoid` 传递函数通常用于隐含层。当净输入大于 3 ( $\exp(-3) \approx 0.05$ ) 时，这些函数基本饱和。如果在训练过程开始就发生这种情况，梯度将非常小，并且网络训练将非常慢。在网络的第一层，净输入是输入乘以权重的乘积加上偏置。如果输入非常大，则权重必须非常小，以防止传递函数变得饱和。标准做法是在将输入应用于网络之前对其进行归一化。

通常，对数据集中的输入向量和目标向量都应用归一化步骤。这样，网络输出始终在归一化后的范围内。然后，当网络投入实际使用时，网络输出可以反向变换回原始目标数据的单位。

对神经网络最形象的描述是：在输入和网络的第一层之间有一个预处理模块，在网络的最后一层和输出之间有一个后处理模块，如下图所示。



工具箱中的大多数网络创建函数，包括多层网络创建函数，如 `feedforwardnet`，会自动为您的网络输入和输出分配处理函数。这些函数将您提供的输入值和目标值变换为更适合网络训练的值。

创建网络后，您可以通过调整网络属性来覆盖默认的输入和输出处理函数。

要查看分配给网络输入的处理函数的元胞数组列表，请访问以下属性：

```
net.inputs{1}.processFcns
```

其中索引 1 引用第一个输入向量。（前馈网络只有一个输入向量。）要查看两层网络的输出返回的处理函数，请访问以下网络属性：

```
net.outputs{2}.processFcns
```

其中索引 2 引用来自第二层的输出向量。（对于前馈网络，只有一个输出向量，它来自最终层。）您可以使用这些属性来更改您希望网络应用于输入和输出的处理函数。然而，默认值通常能提供出色性能。

几个处理函数具有可自定义其运算的参数。您可以访问或更改网络输入的第 *i* 个输入处理函数的参数，如下所示：

```
net.inputs{1}.processParams{i}
```

对于与第二层相关联的网络输出，您可以访问或更改第 *i* 个输出处理函数的参数，如下所示：

```
net.outputs{2}.processParams{i}
```

对于多层网络创建函数，如 **feedforwardnet**，默认输入处理函数是 **removeconstantrows** 和 **mapminmax**。对于输出，默认处理函数也是 **removeconstantrows** 和 **mapminmax**。

下表列出了最常见的预处理和后处理函数。在大多数情况下，您不需要直接使用它们，因为预处理步骤成为网络对象的一部分。当您对网络进行仿真或训练时，预处理和后处理将自动完成。

函数	算法
<b>mapminmax</b>	将输入/目标归一化到 [-1, 1] 范围内
<b>mapstd</b>	归一化输入/目标，使其具有零均值和单位方差
<b>processpca</b>	从输入向量中提取主成分
<b>fixunknowns</b>	处理未知输入
<b>removeconstantrows</b>	删除作为常量的输入/目标

## 表示未知或不关心的目标

未知或“不关心”的目标可以用 NaN 值表示。我们不希望未知的目标值对训练有影响，但如果网络有几个输出，则可能有目标向量的一些元素已知而其他元素未知。一个解决方案是从训练集中删除部分未知的目标向量及其相关联的输入向量，但这可能导致丢失某些良好目标值。更好的解决方案是用 NaN 值来表示那些未知目标。为了计算性能和性能的导数，工具箱的所有性能函数将忽略那些目标。

## 划分数据以实现最优神经网络训练

本主题介绍典型多层网络工作流的一部分。有关详细信息和其他步骤，请参阅“多层浅层神经网络与反向传播训练”（第 21-2 页）。

在训练多层网络时，通常的做法是首先将数据分成三个子集。第一个子集是训练集，用于计算梯度和更新网络权重及偏置。第二个子集是验证集。在训练过程中会监控基于验证集的误差。验证误差通常在训练的初始阶段减小，训练集误差也是如此。然而，当网络开始过拟合数据时，基于验证集的误差通常开始增大。网络权重和偏置以最小的验证集误差保存。这种方法在“提高浅层神经网络泛化能力，避免过拟合”（第 27-24 页）中进行更详细的讨论。

训练期间不使用测试集误差，但测试集误差用于比较不同模型。在训练过程中绘制测试集误差也很有用。如果测试集误差与验证集误差达到最小值所需的迭代次数显著不同，这可能表示数据集的划分不佳。

可使用四个函数将数据划分为训练集、验证集和测试集。它们是 **dividerand**（默认值）、**divideblock**、**divideint** 和 **divideind**。数据划分通常在您训练网络时会自动执行。

函数	算法
<b>dividerand</b>	随机划分数据（默认值）
<b>divideblock</b>	将数据划分为连续数据块
<b>divideint</b>	使用交错选择划分数据
<b>divideind</b>	按索引划分数据

您可以使用以下属性访问或更改网络的划分函数：

`net.divideFcn`

每个划分函数都采用自定义其行为的参数。这些值会被存储，且可以通过以下网络属性进行更改：

`net.divideParam`

每当对网络进行训练时，都会自动访问划分函数，该函数用于将数据划分为训练子集、验证子集和测试子集。如果 `net.divideFcn` 设置为 '`dividerand`'（默认值），则使用划分参数 `net.divideParam.trainRatio`、`net.divideParam.valRatio` 和 `net.divideParam.testRatio` 将数据随机分成三个子集。归入训练集的数据占比为 `trainRatio/(trainRatio+valRatio+testRatio)`，其他两个子集的公式类似。训练、测试和验证的默认比率分别为 0.7、0.15 和 0.15。

如果 `net.divideFcn` 设置为 '`divideblock`'，则使用原始数据集的三个连续数据块将数据分成三个子集（训练取第一个数据块，验证取第二个数据块，测试取第三个数据块）。归入每个子集的原始数据占比同样由用于 `dividerand` 的三个划分参数决定。

如果 `net.divideFcn` 设置为 '`divideint`'，则数据将通过交错方法进行划分，就像发牌一样。这样做是为了让不同百分比的数据归入三个子集。归入每个子集的原始数据占比同样由用于 `dividerand` 的三个划分参数决定。

当 `net.divideFcn` 设置为 '`divideind`' 时，数据按索引划分。三个子集的索引由划分参数 `net.divideParam.trainInd`、`net.divideParam.valInd` 和 `net.divideParam.testInd` 定义。这些索引的默认赋值是空数组，因此在使用该选项时必须设置索引。

# 创建、配置和初始化多层浅层神经网络

## 本节内容

[“其他相关架构” \(第 21-9 页\)](#)

[“初始化权重 \(init\)” \(第 21-10 页\)](#)

本主题介绍典型多层浅层网络工作流的一部分。有关详细信息和其他步骤，请参阅“[多层浅层神经网络与反向传播训练](#)” (第 21-2 页)。

在采集数据后，训练网络的下一步是创建网络对象。函数 **feedforwardnet** 创建一个多层前馈网络。如果在不带输入参数的情况下调用此函数，则会创建一个尚未进行配置的默认网络对象。然后可以使用 **configure** 命令配置生成的网络。

例如，文件 **bodyfat\_dataset.mat** 包含一组预定义的输入向量和目标向量。输入向量定义关于人的身体属性的数据，目标值定义人的身体脂百分比。使用以下命令加载数据：

```
load bodyfat_dataset
```

加载该文件会创建两个变量。输入矩阵 **bodyfatInputs** 由 252 个不同人的 13 个身体属性变量组成的 252 个列向量组成。目标矩阵 **bodyfatTargets** 由对应的 252 个身体脂百分比组成。

下一步是创建网络。以下对 **feedforwardnet** 的调用创建一个两层网络，在隐藏层中有 10 个神经元。(在配置步骤中，输出层中的神经元数量设置为 1，即每个目标向量中的元素数。)

```
net = feedforwardnet;
net = configure(net, bodyfatInputs, bodyfatTargets);
```

可选参数可提供给 **feedforwardnet**。例如，第一个参数是包含每个隐藏层中神经元数量的数组。（默认设置为 10，这意味着一个隐藏层有 10 个神经元。一个隐含层通常就能产生很好的效果，但如果一个隐含层的效果不够好，您可以尝试使用两个隐含层。增加隐藏层中神经元的数量可增强网络的能力，但需要更多计算，并且更可能产生过拟合。）第二个参数包含要使用的训练函数的名称。如果没有提供参数，则默认层数为 2，隐藏层中神经元的默认数量为 10，默认训练函数为 **trainlm**。隐含层的默认传递函数是 **tansig**，输出层的默认传递函数是 **purelin**。

**configure** 命令配置网络对象，它还初始化网络的权重和偏置；因此，网络已准备好接受训练。有时，您可能需要重新初始化权重或执行自定义初始化。“[初始化权重 \(init\)" \(第 21-10 页\)](#)”说明了初始化过程的细节。您也可以跳过配置步骤，直接开始训练网络。**train** 命令将自动配置网络并初始化权重。

## 其他相关架构

虽然两层前馈网络有能力学习几乎任何输入输出关系，但具有更多层的前馈网络可以更快地学习复杂的关系。对于大多数问题，最好从两层开始，如果两层的性能不令人满意，再增加到三层。

函数 **cascadeforwardnet** 创建级联前向网络。这些网络类似于前馈网络，但包括从输入到每层以及从每层到后续层的权重连接。例如，一个三层网络具有从第 1 层到第 2 层、从第 2 层到第 3 层以及从第 1 层到第 3 层的连接。该三层网络还具有从输入到所有三层的连接。额外的连接可能会提高网络学习所需关系的速度。

函数 **patternnet** 创建非常类似于 **feedforwardnet** 的网络，不同之处是它在最后一层使用 **tansig** 传递函数。该网络通常用于模式识别。其他网络可以学习动态或时序关系。

### 初始化权重 (init)

在训练前馈网络之前，您必须初始化权重和偏置。`configure` 命令会自动初始化权重，但您可能需要重新初始化它们。您可以使用 `init` 命令来执行此操作。该函数将网络对象作为输入，并返回已初始化所有权重和偏置的网络对象。以下是网络初始化（或重新初始化）的方式：

```
net = init(net);
```

# 训练与应用多层浅层神经网络

## 本节内容

- “训练算法” (第 21-11 页)
- “训练示例” (第 21-12 页)
- “使用网络” (第 21-14 页)

**提示** 要训练深度学习网络, 请使用 `trainNetwork`。

本主题介绍典型多层浅层网络工作流的一部分。有关详细信息和其他步骤, 请参阅 “多层浅层神经网络与反向传播训练” (第 21-2 页)。

当初始化网络权重和偏置后, 网络已准备好进行训练。多层前馈网络可以训练用于函数逼近 (非线性回归) 或模式识别。训练过程需要一组适当的网络行为示例 - 网络输入  $p$  和目标输出  $t$ 。

根据网络性能函数 `net.performFcn` 的定义, 训练神经网络的过程包括调整网络的权重和偏置值以优化网络性能。前馈网络的默认性能函数是均方误差 `mse` - 网络输出  $a$  和目标输出  $t$  之间的平均平方误差。其定义如下:

$$F = mse = \frac{1}{N} \sum_{i=1}^N (e_i)^2 = \frac{1}{N} \sum_{i=1}^N (t_i - a_i)^2$$

(单个平方误差也可以加权。请参阅 “Train Neural Networks with Error Weights”。实现训练有两种不同方式: 增量模式和批量模式。在增量模式下, 在将每个输入应用于网络后计算梯度并更新权重。在批量模式下, 在训练集中的所有输入都应用于网络后再更新权重。本主题介绍使用 `train` 命令的批量模式训练。“Incremental Training with `adapt`” 中讨论使用 `adapt` 命令的增量训练。对于大多数问题, 当使用 Deep Learning Toolbox 软件时, 批量训练比增量训练快得多, 产生的误差也更小。)

对于训练多层前馈网络, 任何标准数值优化算法都可用于优化性能函数, 但有几个关键算法已显示出对神经网络训练有优异性能。这些优化方法或者使用网络性能关于网络权重的梯度, 或者使用网络误差关于权重的 Jacobian 矩阵。

梯度和 Jacobian 矩阵使用的计算方法称为反向传播算法, 该算法涉及在网络中反向执行计算。反向传播计算是使用微积分的链式法则求导的, 在 [HDB96] 的第 11 章 (针对梯度) 和第 12 章 (针对 Jacobian 矩阵) 中讲述。

## 训练算法

为了说明训练的工作原理, 以最简单的优化算法 - 梯度下降为例。它会沿着性能函数下降最快的方向 (梯度的负方向) 上更新网络权重和偏置。该算法的一次迭代可以写为如下形式

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$$

其中,  $\mathbf{x}_k$  是当前权重和偏置的向量,  $\mathbf{g}_k$  是当前梯度,  $\alpha_k$  是学习率。此方程会一直迭代, 直到网络收敛。

下表列出了 Deep Learning Toolbox 软件中可用的使用基于梯度或 Jacobian 矩阵方法的训练算法。

有关这些方法的详细说明, 另请参阅以下著作: Hagan, M.T., H.B. Demuth, and M.H. Beale, Neural Network Design, Boston, MA:PWS Publishing, 1996, 第 11 和 12 章。

函数	算法
<b>trainlm</b>	Levenberg-Marquardt
<b>trainbr</b>	贝叶斯正则化
<b>trainbfg</b>	BFGS 拟牛顿
<b>trainrp</b>	弹性反向传播
<b>trainscg</b>	量化共轭梯度
<b>traincgb</b>	带 Powell/Beale 重启的共轭梯度
<b>traincfg</b>	Fletcher-Powell 共轭梯度
<b>traincgp</b>	Polak-Ribiére 共轭梯度
<b>trainoss</b>	单步正割
<b>traingdx</b>	可变学习率梯度下降
<b>traingdm</b>	带动量的梯度下降
<b>traingd</b>	梯度下降

最快的训练函数通常是 **trainlm**，它是 **feedforwardnet** 的默认训练函数。拟牛顿法 **trainbfg** 也相当快。这两种方法对于大型网络（具有数千个权重）往往效率较低，因为它们需要更多内存和更多计算时间。此外，**trainlm** 在函数拟合（非线性回归）问题上的性能优于模式识别问题。

在训练大型网络和模式识别网络时，**trainscg** 和 **trainrp** 是不错的选择。其内存要求相对较小，但比标准梯度下降算法要快得多。

请参阅“选择多层神经网络训练函数”（第 27-13 页），了解上表所示的训练算法性能的全面比较。

注意，术语“反向传播”在应用于神经网络训练时，有时特指梯度下降算法。此处不使用该术语，因为上述所有训练函数都涉及在网络中反向执行计算来求出梯度和 Jacobian 矩阵的过程。所以，相较于反向传播，使用具体优化算法的名称会更清楚。

此外，多层网络有时也称为反向传播网络。不过，用于计算多层网络中的梯度和 Jacobian 矩阵的反向传播方法也可应用于许多不同网络架构。事实上，任何具有可微分传递函数、权重函数和净输入函数的网络的梯度和 Jacobian 矩阵都可以通过反向传播过程使用 Deep Learning Toolbox 软件来计算。您甚至可以创建自己的自定义网络，然后使用上表中的任何训练函数对它们进行训练。梯度和 Jacobian 矩阵将自动为您计算。

## 训练示例

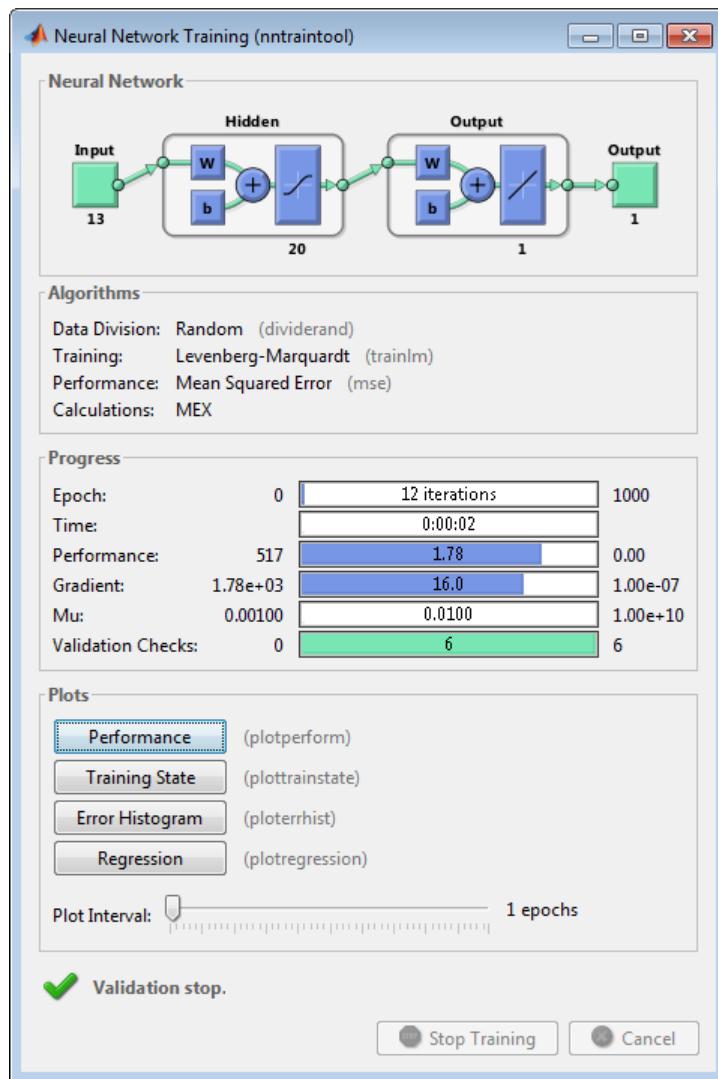
要演示训练过程，请执行以下命令：

```
load bodyfat_dataset
net = feedforwardnet(20);
[net,tr] = train(net,bodyfatInputs,bodyfatTargets);
```

请注意，您不需要发出 **configure** 命令，因为配置是由 **train** 函数自动完成的。训练期间将出现训练窗口，如下图所示。（如果您不想在训练期间显示此窗口，可以将参数 **net.trainParam.showWindow** 设置为 **false**。如果您要在命令行中显示训练信息，可以将参数 **net.trainParam.showCommandLine** 设置为 **true**。）

此窗口显示数据已使用 **dividerand** 函数进行划分，并且使用了 Levenberg-Marquardt (**trainlm**) 训练方法和均方误差性能函数。上面提到过，这些是 **feedforwardnet** 的默认设置。

在训练期间，进度会在训练窗口中不断更新。最关注的是性能、性能梯度的量级和验证检查的数量。梯度的量级和验证检查的数量用于终止训练。当训练达到性能的最小值时，梯度将变得非常小。如果梯度的量级小于  $1e-5$ ，训练将停止。该限制可通过设置参数 `net.trainParam.min_grad` 进行调整。验证检查的数量表示验证性能无法降低的连续迭代次数。如果该数字达到 6（默认值），训练将停止。在本次运行中，您可以看到鉴于验证检查的数量，训练确实停止了。您可以通过设置参数 `net.trainParam.max_fail` 来更改此条件。（请注意，因为初始权重和偏置的随机设置，您的结果可能不同于下图所示的结果。）



还有其他条件可用于停止网络训练。下表列出了这些条件。

参数	停止条件
<code>min_grad</code>	最小梯度幅值
<code>max_fail</code>	验证增加的最大数量
<code>time</code>	最大训练时间
<code>goal</code>	最低性能值

参数	停止条件
epochs	最大训练轮数 (迭代数)

如果您点击训练窗口中的 **Stop Training** 按钮，训练也将停止。如果性能函数在多次迭代中未显著降低，您可能需要这样做。通过重新发出上面所示的 `train` 命令，始终可以继续训练。它将从上一次运行完成的状态继续训练网络。

在训练窗口中，您可以访问四个图：性能、训练状态、误差直方图和回归。性能图显示性能函数值对迭代次数的图。它对训练、验证和测试性能绘图。训练状态图显示其他训练变量的进度，例如梯度幅值、验证检查的数量等。误差直方图显示网络误差的分布。回归图显示网络输出和网络目标之间的回归。您可以使用直方图和回归图来验证网络性能，如“分析训练后的浅层神经网络性能”（第 21-15 页）中所述。

## 使用网络

在网络经过训练和验证后，网络对象可用于计算网络对任何输入的响应。例如，如果您要在构造数据集中找到对第五个输入向量的网络响应，可以使用以下命令

```
a = net(bodyfatInputs(:,5))
```

```
a =
```

```
27.3740
```

如果尝试此命令，输出可能会有所不同，具体取决于网络初始化时随机数生成器的状态。下面，调用网络对象来计算体脂数据集中所有输入向量的并发集合的输出。这是批量模式的仿真，其中所有输入向量都放在一个矩阵中。这比一次提交一个向量要高效得多。

```
a = net(bodyfatInputs);
```

每次训练神经网络时，由于初始权重和偏置值不同，并且将数据划分为训练集、验证集和测试集的方式也不同，可能会产生不同的解。因此，针对同一问题训练的不同神经网络对同一输入可能给出不同输出。为确保找到准确度良好的神经网络，需要多次重新训练。

如果需要更高的准确度，可以采用几种其他方法来改进初始解。有关详细信息，请参阅“提高浅层神经网络泛化能力，避免过拟合”（第 27-24 页）。

## 分析训练后的浅层神经网络性能

本主题介绍典型浅层神经网络工作流的一部分。有关详细信息和其他步骤，请参阅“多层浅层神经网络与反向传播训练”（第 21-2 页）。要了解如何监控深度学习训练进度，请参阅“监控深度学习训练进度”（第 5-22 页）。

当“训练与应用多层浅层神经网络”（第 21-11 页）中所述的训练结束后，您可以检查网络性能，并确定是否需要对训练过程、网络架构或数据集进行任何更改。首先检查训练记录 `tr`，这是训练函数返回的第二个参数。

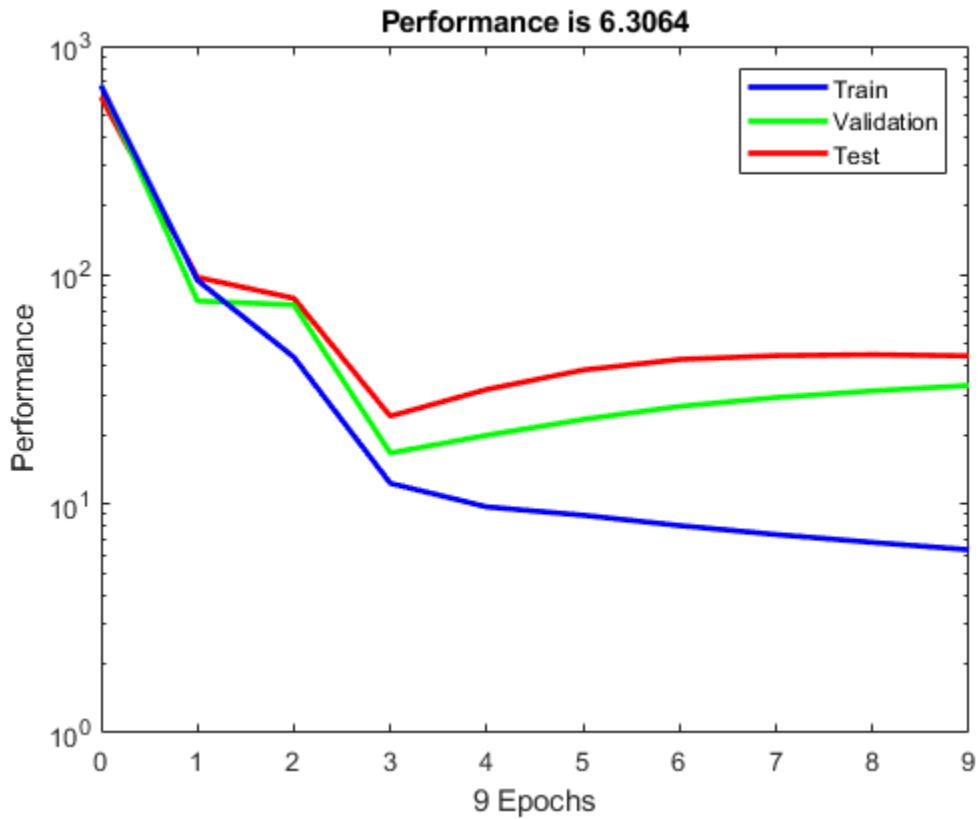
```
tr
```

```
tr = struct with fields:
    trainFcn: 'trainlm'
    trainParam: [1x1 struct]
    performFcn: 'mse'
    performParam: [1x1 struct]
        derivFcn: 'defaultderiv'
        divideFcn: 'dividerand'
        divideMode: 'sample'
    divideParam: [1x1 struct]
        trainInd: [1x176 double]
        valInd: [1x38 double]
        testInd: [1x38 double]
        stop: 'Validation stop.'
    num_epochs: 9
    trainMask: {[1x252 double]}
        valMask: {[1x252 double]}
        testMask: {[1x252 double]}
    best_epoch: 3
        goal: 0
    states: {1x8 cell}
        epoch: [0 1 2 3 4 5 6 7 8 9]
        time: [1x10 double]
        perf: [1x10 double]
        vperf: [1x10 double]
        tperf: [1x10 double]
        mu: [1x10 double]
    gradient: [1x10 double]
    val_fail: [0 0 0 1 2 3 4 5 6]
    best_perf: 12.3078
    best_vperf: 16.6857
    best_tperf: 24.1796
```

此结构体包含关于网络训练的所有信息。例如，`tr.trainInd`、`tr.valInd` 和 `tr.testInd` 分别包含训练、验证和测试集中使用的数据点的索引。如果您要使用相同的数据划分重新训练网络，可以将 `net.divideFcn` 设置为 '`divideInd`'，`net.divideParam.trainInd` 设置为 `tr.trainInd`，`net.divideParam.valInd` 设置为 `tr.valInd`，`net.divideParam.testInd` 设置为 `tr.testInd`。

`tr` 结构体还在训练过程中跟踪几个变量，如性能函数值、梯度幅值等。您可以通过 `plotperf` 命令，使用训练记录来绘制性能进度图：

```
plotperf(tr)
```

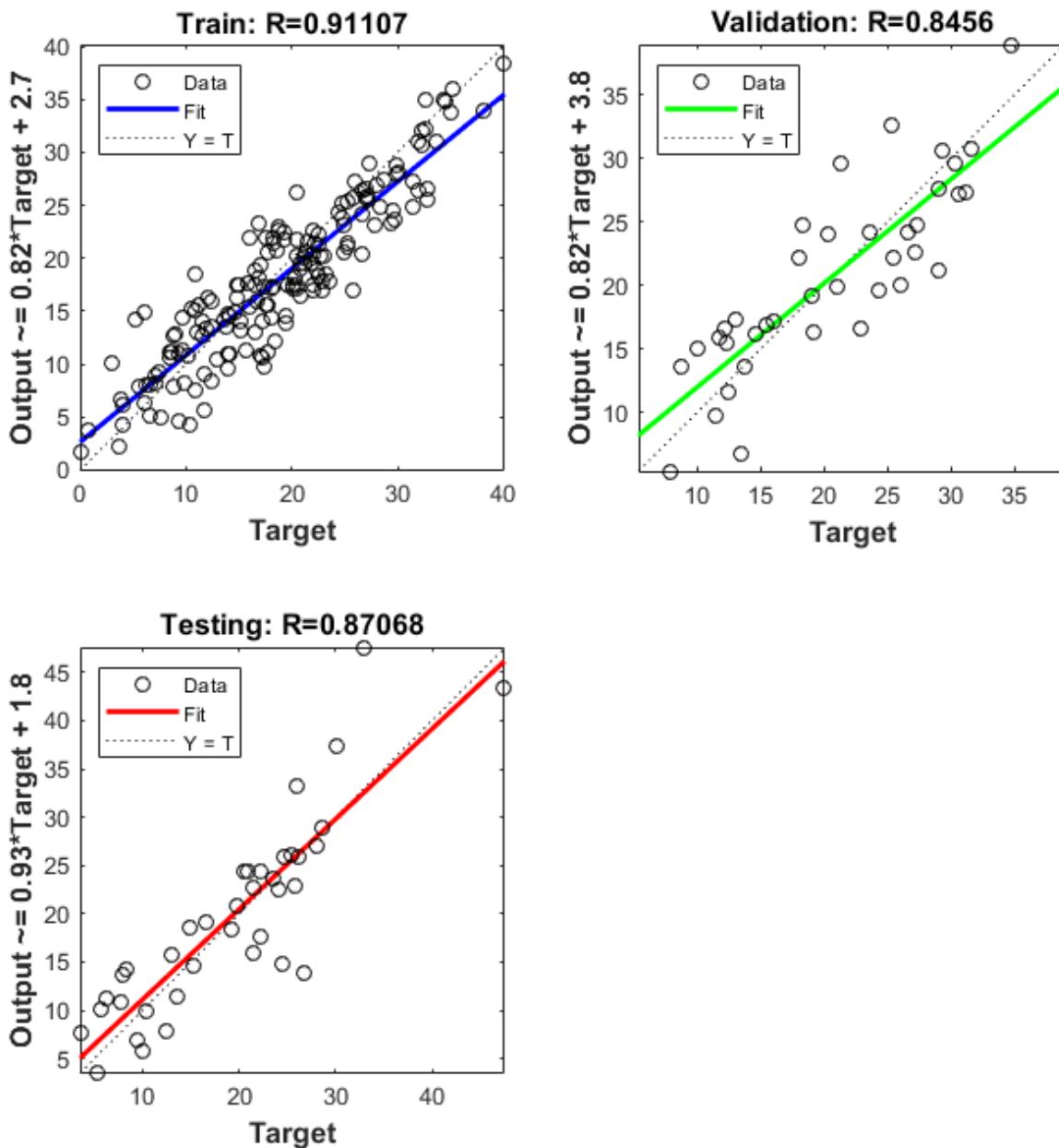


属性 `tr.best_epoch` 表示验证性能达到最小值时所处的迭代。在训练停止之前，训练继续进行了 6 次迭代。

此图并未表明训练有任何重大问题。验证曲线和测试曲线非常相似。如果测试曲线在验证曲线增大之前显著增大，则可能发生了一些过拟合。

验证网络的下一步是创建一个回归图，该图显示网络输出和目标之间的关系。如果训练是完美的，则网络输出和目标将完全相等，但在实践中极少出现这种完美关系。对于体脂示例，我们可以使用以下命令创建回归图。第一个命令计算经过训练的网络对数据集中所有输入的响应。接下来的六个命令提取属于训练、验证和测试子集的输出和目标。最后一个命令为训练、测试和验证创建三个回归图。

```
bodyfatOutputs = net(bodyfatInputs);
trOut = bodyfatOutputs(tr.trainInd);
vOut = bodyfatOutputs(tr.valInd);
tsOut = bodyfatOutputs(tr.testInd);
trTarg = bodyfatTargets(tr.trainInd);
vTarg = bodyfatTargets(tr.valInd);
tsTarg = bodyfatTargets(tr.testInd);
plotregression(trTarg, trOut, 'Train', vTarg, vOut, 'Validation', tsTarg, tsOut, 'Testing')
```



这三个图表示训练、验证和测试数据。每个图中的虚线表示完美结果 - 输出 = 目标。实线表示输出和目标之间的最佳拟合线性回归线。R 值表示输出和目标之间的关系。如果 R = 1，这表明输出和目标之间存在准确的线性关系。如果 R 接近于零，则输出和目标之间没有线性关系。

对于此示例，训练数据表明拟合良好。验证和测试结果也显示较大的 R 值。散点图有助于显示某些数据点的拟合效果很差。例如，测试集中有一个数据点的网络输出接近 35，而对应的目标值约为 12。下一步将是调查此数据点，以确定它是否表示外插点（即，它是否在训练数据集之外）。如果它是外插点，则应将其包括在训练集中，并且应采集额外的数据以用于测试集中。

### 改进结果

如果网络不够准确，您可以尝试重新初始化网络并进行训练。每次初始化前馈网络时，网络参数都不同，并且可能产生不同解。

```
net = init(net);
net = train(net, bodyfatInputs, bodyfatTargets);
```

作为另一种方法，您可以将隐藏神经元的数量增加到 20 个以上。隐藏层中的神经元数量越多，网络的灵活度就越高，因为网络可以优化更多参数。（请逐渐增加层大小。如果将隐含层设置得太大，可能会导致问题的特征不清晰，并且网络必须优化的参数数目多于约束这些参数的数据向量数目。）

第三个选择是尝试不同训练函数。例如，与使用早停法相比，使用 `trainbr` 的贝叶斯正则化训练有时可以提供更好的泛化能力。

最后，尝试使用额外的训练数据。为网络提供额外数据，则生成的网络更可能对新数据具有更强的泛化能力。

# 动态神经网络

---

- “设计时序 NARX 反馈神经网络” (第 22-2 页)
- “多步神经网络预测” (第 22-8 页)

## 设计时序 NARX 反馈神经网络

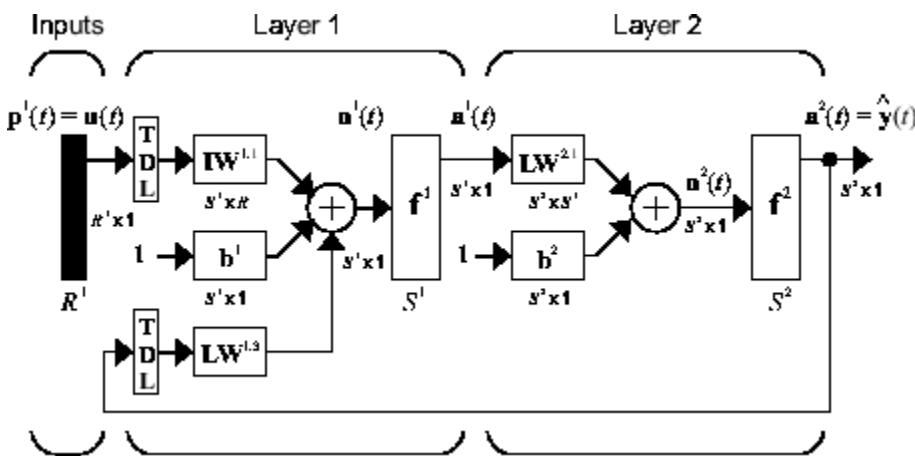
要查看在开环形式、闭环形式和开环/闭环多步预测中应用 NARX 网络的示例，请参阅“多步神经网络预测”（第 22-8 页）。

到目前为止讨论的所有特定动态网络或者是仅在输入层具有动态的专用网络，或者是前馈网络。外因输入非线性自回归网络 (NARX) 是循环动态网络，其反馈连接包含网络的几个层。NARX 模型基于线性 ARX 模型，该模型通常用于时序建模。

NARX 模型的定义方程是

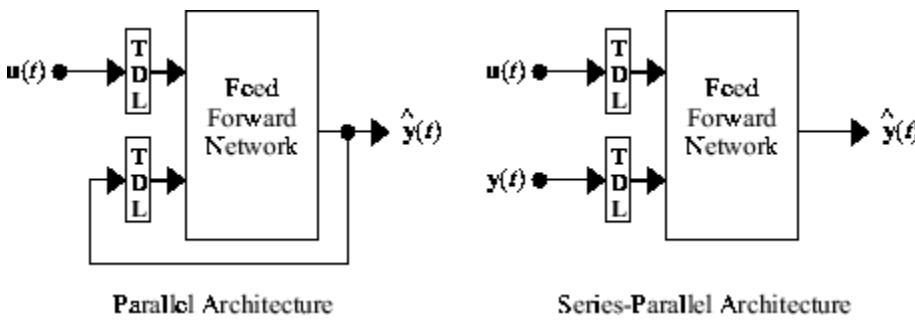
$$y(t) = f(y(t-1), y(t-2), \dots, y(t-n_y), u(t-1), u(t-2), \dots, u(t-n_u))$$

其中从属输出信号  $y(t)$  的下一个值根据输出信号的先前值和独立（外因）输入信号的先前值进行回归。您可以通过使用前馈神经网络来逼近函数  $f$ ，从而实现 NARX 模型。生成的网络的图如下所示，其中两层前馈网络用于逼近。这种实现还支持向量 ARX 模型，其中输入和输出可以是多维的。



NARX 网络有许多应用。它可以用作预测器，预测输入信号的下一个值。它也可用于非线性滤波，其中目标输出是输入信号的无噪声版本。NARX 网络可用于另一个重要应用，即非线性动态系统的建模。

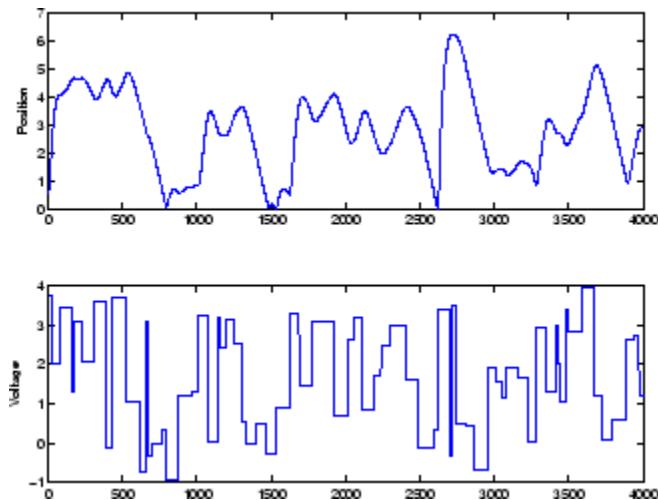
在说明 NARX 网络的训练之前，需要解释在训练中有用的一项重要配置。您可以将 NARX 网络的输出视为您尝试建模的某个非线性动态系统的输出估计值。输出会反馈到前馈神经网络的输入，这是标准 NARX 架构的一部分，如下方左图所示。由于在网络训练期间真实输出是可用的，因此您可以创建一个串并行架构（请参阅 [NaPa91]），其中使用真实输出而不是反馈估计的输出，如下方右图所示。这样有两个优点。第一，前馈网络的输入更准确。第二，生成的网络为纯前馈架构，可以使用静态反向传播进行训练。



下面显示如何使用串并行架构来训练 NARX 网络以进行动态系统建模。

“Use the NARMA-L2 Controller Block” 中开始描述的磁悬浮系统是一个 NARX 网络示例。下图中的底部图显示施加到电磁铁上的电压，顶部图显示永磁体的位置。以 0.01 秒的采样间隔采集数据，形成两个时序。

目标是为此磁悬浮系统开发一个 NARX 模型。



首先，加载训练数据。使用将输入和输出都延迟两个数据点的抽头延迟线，因此训练从第三个数据点开始。串并行网络有两个输入，即  $u(t)$  序列和  $y(t)$  序列。

```
load magdata
y = con2seq(y);
u = con2seq(u);
```

使用函数 `narxnet` 创建串并行 NARX 网络。在隐藏层中使用 10 个神经元，并使用 `trainlm` 进行训练，然后使用 `preperts` 准备数据：

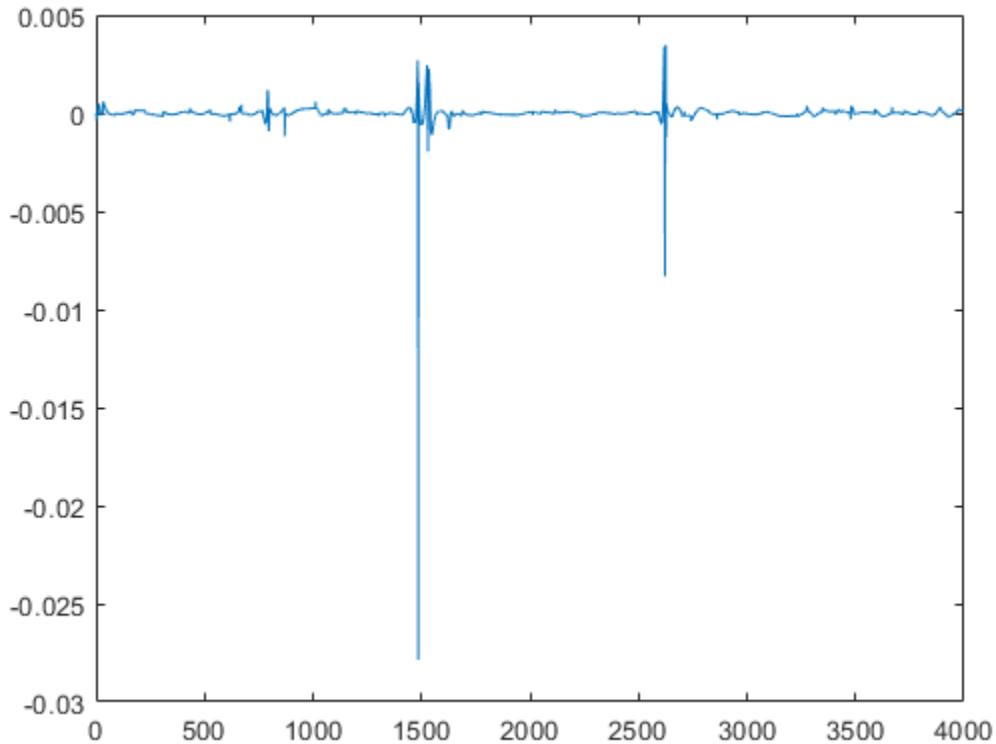
```
d1 = [1:2];
d2 = [1:2];
narx_net = narxnet(d1,d2,10);
narx_net.divideFcn = "";
narx_net.trainParam.min_grad = 1e-10;
[p,Pi,Ai,t] = preperts(narx_net,u,{},y);
```

(请注意， $y$  序列被视为反馈信号，它既是输入也是输出（目标）。稍后，当您实现闭环时，适当的输出将连接到适当的输入。) 现在您已准备好训练网络。

```
narx_net = train(narx_net,p,t,Pi);
```

您现在可以仿真网络，并绘制这一串并行网络产生的误差。

```
yp = sim(narx_net,p,Pi);
e = cell2mat(yp)-cell2mat(t);
plot(e)
```



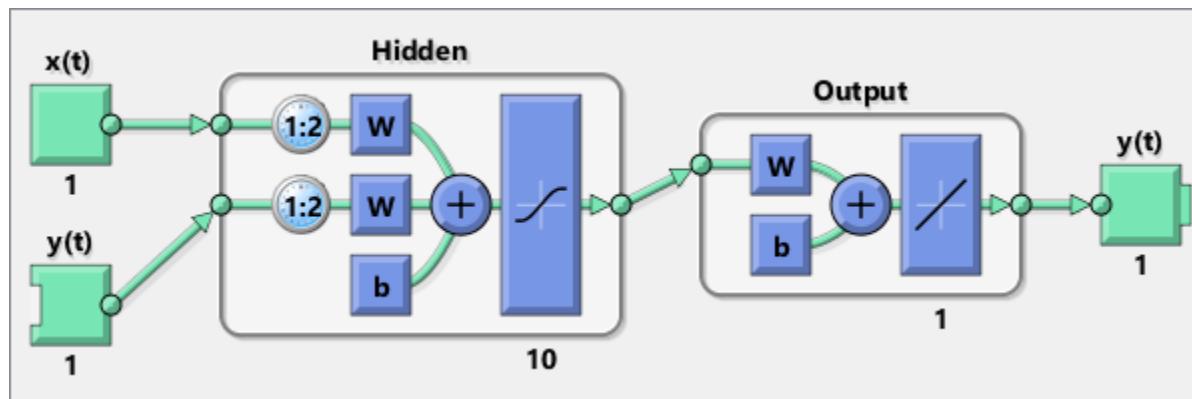
您可以看到误差非常小。然而，由于采用串并行配置，这些只是单步预测的误差。更严格的测试是将网络重新排列成原来的并行形式（闭环），然后执行多时间步的迭代预测。这就是出现了并行运算。

可使用工具箱函数 (**closeloop**) 将 NARX (及其他) 网络从串并行配置 (开环) 转换为并行配置 (闭环)，前者适用于训练，后者适用于多步预测。以下命令演示如何将您刚刚训练的网络转换为并行形式：

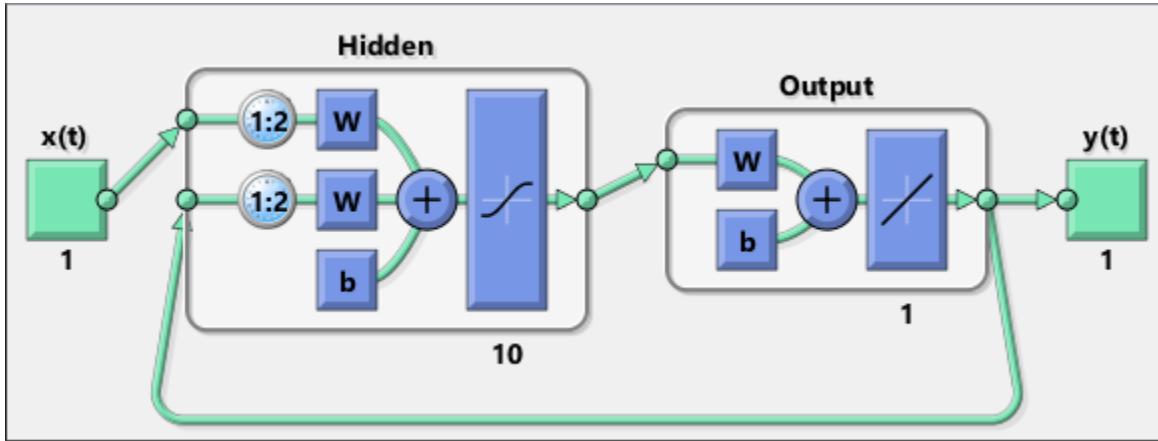
```
narx_net_closed = closeloop(narx_net);
```

要查看两个网络之间的差异，可以使用查看命令：

```
view(narx_net)
```



```
view(narx_net_closed)
```



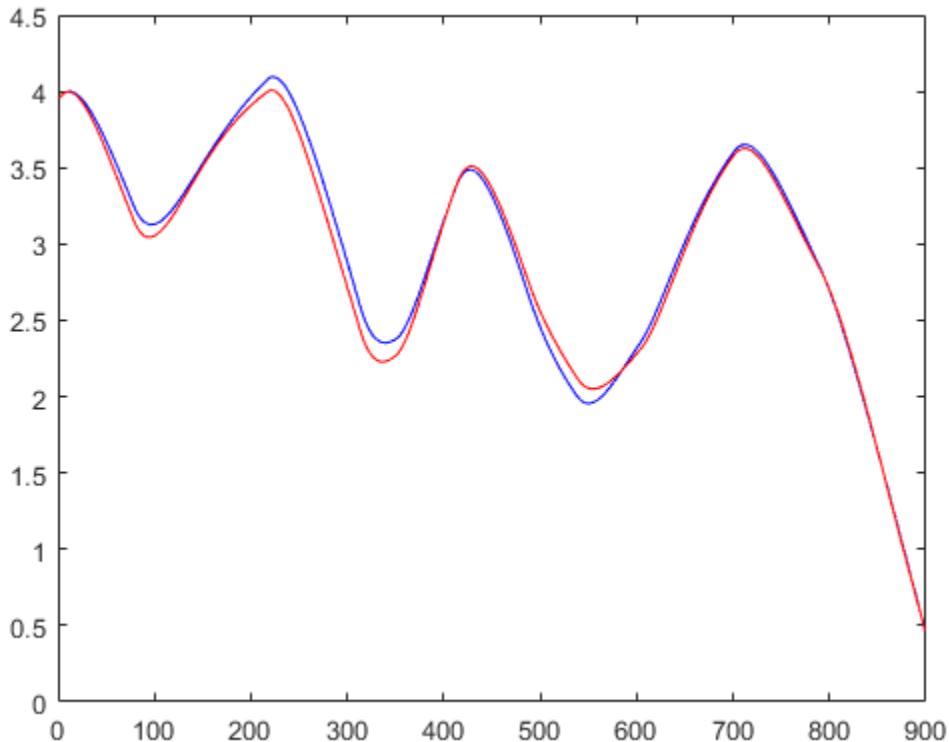
所有训练以开环（也称为串并行架构）形式完成，包括验证和测试步骤。典型的工作流是完全以开环方式创建网络，并且仅当网络经过训练（包括验证和测试步骤）后，它才会变换为闭环，以进行向前多步预测。同样，GUI 中的 **R** 值是基于开环训练结果进行计算的。

您现在可以使用闭环（并行）配置来执行 900 个时间步的迭代预测。在此网络中，您需要加载两个初始输入和两个初始输出作为初始条件。您可以使用 **preparets** 函数准备数据。它将使用网络结构来确定如何适当地划分和偏移数据。

```

y1 = y(1700:2600);
u1 = u(1700:2600);
[p1,Pi1,Ai1,t1] = preparets(narx_net_closed,u1,[],y1);
yp1 = narx_net_closed(p1,Pi1,Ai1);
TS = size(t1,2);
plot(1:TS,cell2mat(t1),'b',1:TS,cell2mat(yp1),'r')

```



该图说明了迭代预测。蓝线表示磁体的实际位置，红线表示由 NARX 神经网络预测的位置。尽管网络向前预测了 900 个时间步，但预测非常准确。

为了使并行响应（迭代预测）准确，重要的是要对网络进行训练，使串并行配置（单步预测）中的误差非常小。

您还可以使用 `narxnet` 命令且将第四个输入参数设置为 '`closed`' 来创建并行（闭环）NARX 网络，并直接训练该网络。一般情况下，与串并行训练相比，此训练需要的时间更长，获得的实现性能也较差。

每次训练神经网络时，由于初始权重和偏置值不同，并且将数据划分为训练集、验证集和测试集的方式也不同，可能会产生不同的解。因此，针对同一问题训练的不同神经网络对同一输入可能给出不同输出。为确保找到准确度良好的神经网络，需要多次重新训练。

如果需要更高的准确度，可以采用几种其他方法来改进初始解。有关详细信息，请参阅“提高浅层神经网络泛化能力，避免过拟合”（第 27-24 页）。

## 多个外部变量

磁悬浮示例说明如何用单一外部输入值对随时间变化的时序建模。但是，NARX 网络可以解决具有多个外部输入元素的问题，并预测具有多个元素的序列。在这些情况下，输入和目标由表示时间的行元胞数组组成，但每个元胞元素都是输入或目标信号的  $N \times 1$  向量。

例如，此处有一个数据集，它由包含两个元素的外部变量组成，用于预测由单个元素组成的序列。

```
[X,T] = ph_dataset;
```

外部输入  $X$  格式化为由二元素向量组成的行元胞数组，每个向量表示酸和基底溶液流。目标表示溶液随时间变化而得到的 pH 值。

您可以使用函数 `con2seq` 将自己的多元素序列数据从矩阵形式重新格式化为神经网络时序形式。

训练网络的过程就像上面处理磁悬浮问题一样进行。

```
net = narxnet(10);
[x,xi,ai,t] = preparets(net,X,{},T);
net = train(net,x,t,xi,ai);
y = net(x,xi,ai);
e = gsubtract(t,y);
```

要查看在开环形式、闭环形式和开环/闭环多步预测中应用 NARX 网络的示例，请参阅“多步神经网络预测”（第 22-8 页）。

## 多步神经网络预测

### 本节内容

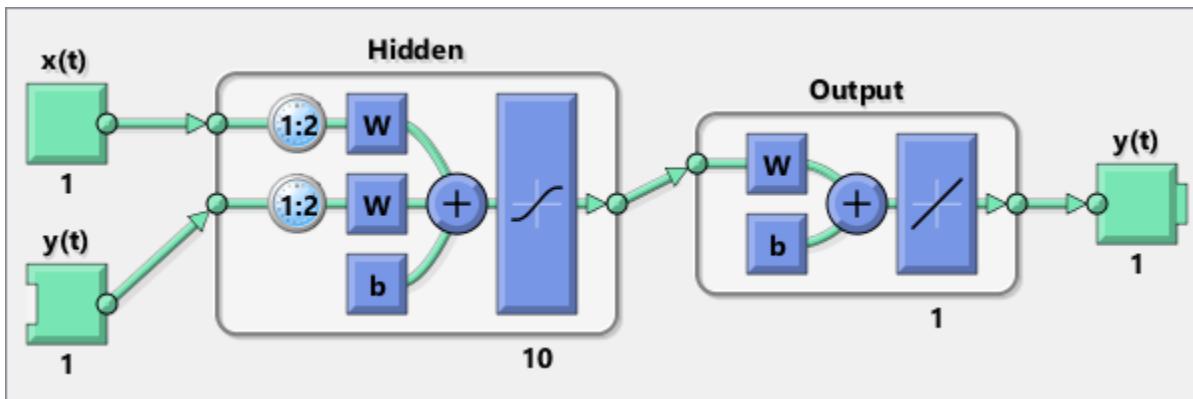
- “在开环模式下设置”（第 22-8 页）
- “基于初始条件进行多步闭环预测”（第 22-8 页）
- “遵循已知序列的多步闭环预测”（第 22-9 页）
- “在闭环仿真后进行开环仿真”（第 22-10 页）

### 在开环模式下设置

具有反馈的动态网络，如 **narxnet** 和 **narnet** 神经网络，可以使用函数 **openloop** 和 **closeloop** 在开环和闭环模式之间变换。闭环网络进行多步预测。换句话说，它们在外部反馈缺失时继续通过使用内部反馈来进行预测。

此处，训练神经网络来对磁悬浮系统建模，并在默认开环模式下进行仿真。

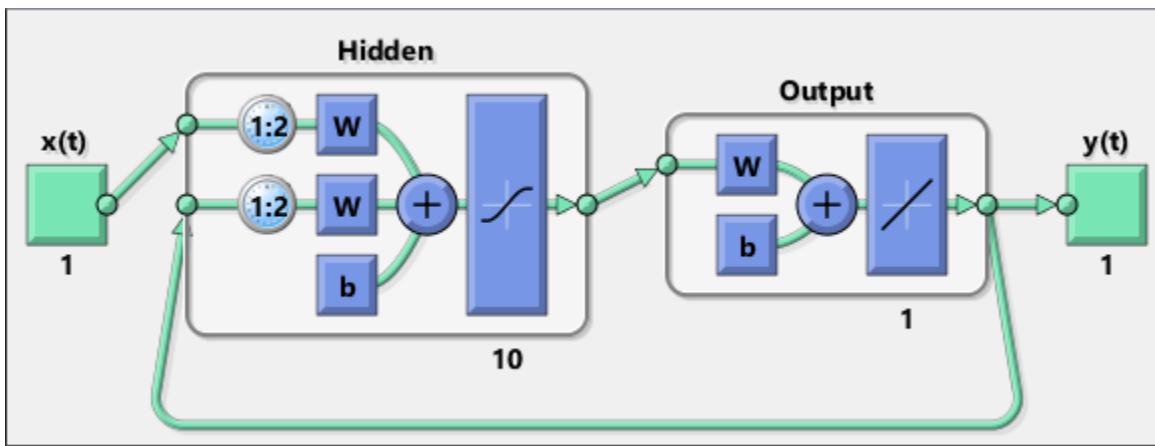
```
[X,T] = maglev_dataset;
net = narxnet(1:2,1:2,10);
[x,xi,ai,t] = preparets(net,X,{},T);
net = train(net,x,t,xi,ai);
y = net(x,xi,ai);
view(net)
```



### 基于初始条件进行多步闭环预测

神经网络也可以仅以闭环形式进行仿真，这样，在给定外部输入序列和初始条件下，神经网络执行的预测数与输入序列具有的时间步数一样多。

```
netc = closeloop(net);
view(netc)
```



此处，训练数据用于定义输入  $x$ ，以及初始输入和层延迟状态  $xi$  和  $ai$ ，但它们可以定义为对任何输入序列和初始状态进行多次预测。

```
[x,xi,ai,t] = preparets(netc,X,[],T);
yc = netc(x,xi,ai);
```

## 遵循已知序列的多步闭环预测

也可以在开环模式下用时序的所有已知值对经过训练的神经网络进行仿真，然后切换到闭环模式继续仿真，对将来进行所需次数的预测。

正如 **openloop** 和 **closeloop** 可用于开环和闭环神经网络之间的变换一样，它们可以转换开环和闭环网络的状态。以下是这些函数的完整接口。

```
[open_net,open_xi,open_ai] = openloop(closed_net,closed_xi,closed_ai);
[closed_net,closed_xi,closed_ai] = closeloop(open_net,open_xi,open_ai);
```

假设有以下情形：您可能有一个 20 个时间步的磁悬浮行为记录，并且您想要向前预测 20 个时间步。

首先，定义输入和目标的前 20 个时间步，表示已知输出由目标  $t$  定义的 20 个时间步。定义输入的接下来的 20 个时间步后，使用网络来预测 20 个输出，在其中使用其每个预测反馈来帮助网络执行下一个预测。

```
x1 = x(1:20);
t1 = t(1:20);
x2 = x(21:40);
```

然后基于此数据对开环神经网络进行仿真。

```
[x,xi,ai,t] = preparets(net,x1,[],t1);
[y1,xf,af] = net(x,xi,ai);
```

现在，网络返回的最终输入和层状态与网络一起转换为闭环形式。开环网络的最终输入状态  $xf$  和层状态  $af$  成为闭环网络的初始输入状态  $xi$  和层状态  $ai$ 。

```
[netc,xi,ai] = closeloop(net,xf,af);
```

通常使用 **preparets** 来定义初始输入和层状态。由于这些已在开环仿真结束时获得，您不需要 **preparets** 继续进行闭环网络的 20 个时间步预测。

```
[y2,xf,af] = netc(x2,xi,ai);
```

请注意，您可以将 `x2` 设置为不同输入序列，以针对您要进行预测的不同时间步数测试不同方案。例如，如果使用 10 个随机输入来预测磁悬浮系统的行为：

```
x2 = num2cell(rand(1,10));
[y2,xf,af] = netc(x2,xi,ai);
```

## 在闭环仿真后进行开环仿真

如果在仿真后网络处于闭环形式，可以从那里以开环形式继续仿真。此处，闭环状态转换回开环状态。  
(您不必将网络转换回开环形式，因为您已有原始开环网络。)

```
[~,xi,ai] = openloop(netc,xf,af);
```

现在，您可以定义外部输入和开环反馈的延拓，并仿真开环网络。

```
x3 = num2cell(rand(2,10));
y3 = net(x3,xi,ai);
```

这样，您就可以在开环和闭环方式之间切换仿真。这方面的一个应用是对传感器进行时序预测，其中最后一个传感器值通常是已知的，允许对下一时间步进行开环预测。但是，在某些情况下，传感器读数不可用或已知是错误的，需要一个闭环预测时间步。预测可以在开环和闭环形式之间交替进行，具体取决于最后一个时间步的传感器读数的可用性。

# 控制系统

---

## 在 Simulink 中设计神经网络预测控制器

### 本节内容

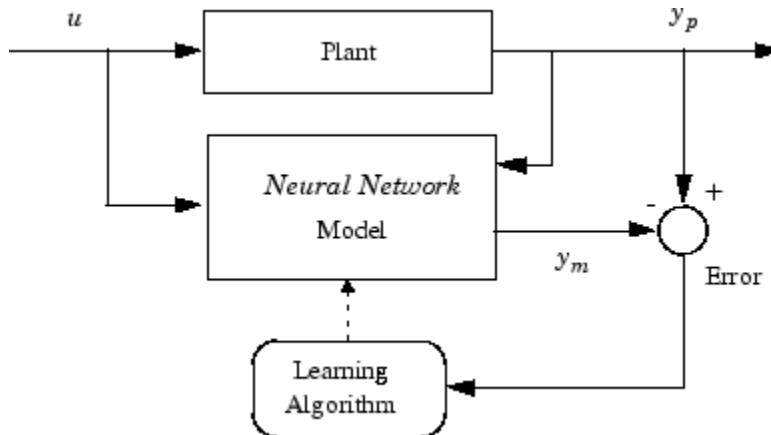
- “系统识别” (第 23-2 页)
- “预测控制” (第 23-3 页)
- “使用神经网络预测控制器模块” (第 23-3 页)

在 Deep Learning Toolbox 软件中实现的神经网络预测控制器使用非线性被控对象的神经网络模型来预测被控对象将来的性能。然后，控制器计算控制输入，该控制输入将在指定的将来时间范围内优化被控对象性能。模型预测控制的第一步是确定神经网络被控对象模型（系统识别）。接下来，控制器使用被控对象模型来预测将来的性能。（请参阅 Model Predictive Control Toolbox™ 文档，了解各种模型预测控制策略在线性系统中的应用的完整内容。）

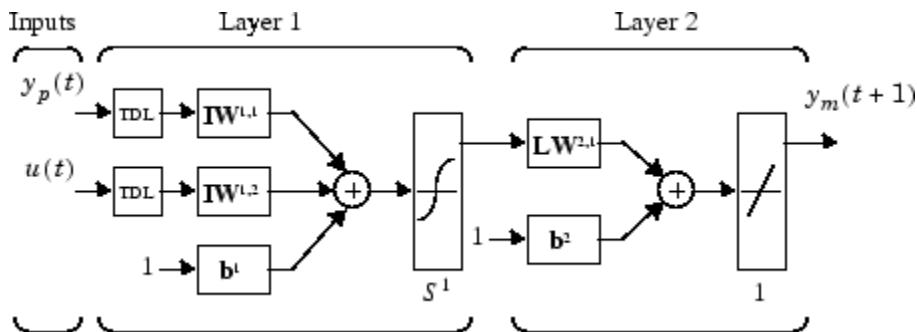
下一节说明系统识别过程。接下来是对优化过程的说明。最后，讨论如何使用在 Simulink® 环境中实现的模型预测控制器模块。

### 系统识别

模型预测控制的第一阶段是训练神经网络来表示被控对象的前向动态。被控对象输出和神经网络输出之间的预测误差用作神经网络训练信号。该过程如下图所示：



神经网络被控对象模型使用先前的输入和先前的被控对象输出来预测被控对象输出的将来值。下图给出了神经网络被控对象模型的结构。



该网络可以在批量模式下使用从被控对象运行中采集的数据进行离线训练。您可以使用在“多层浅层神经网络与反向传播训练”（第 21-2 页）中讨论的任何训练算法进行网络训练。此过程将在以下各节中详细讨论。

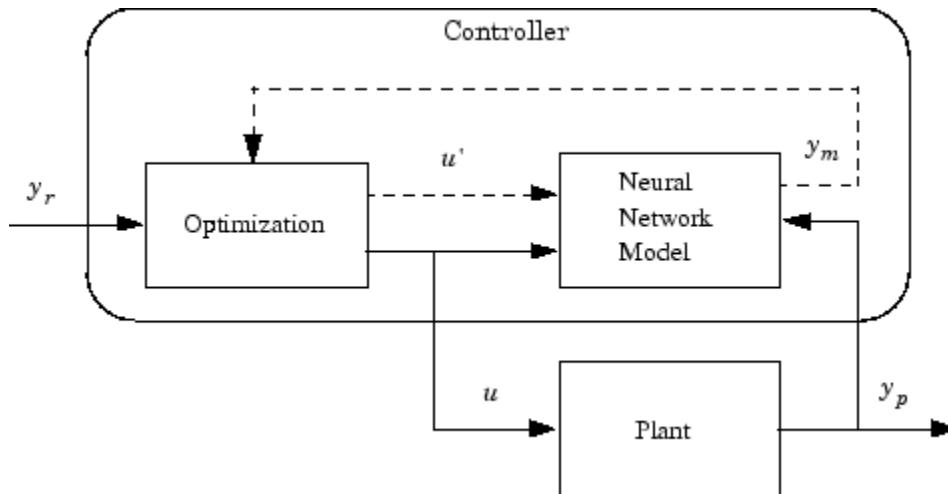
## 预测控制

模型预测控制方法基于滚动时域法 [SoHa96]。神经网络模型预测在指定时域内的被控对象响应。数值优化程序使用预测来确定控制信号，该控制信号最小化在指定时域内的以下性能条件

$$J = \sum_{j=N_1}^{N_2} (y_r(t+j) - y_m(t+j))^2 + \rho \sum_{j=1}^{N_u} (u'(t+j-1) - u'(t+j-2))^2$$

其中， $N_1$ 、 $N_2$  和  $N_u$  定义计算跟踪误差和控制增量的时域。 $u'$  变量是暂定控制信号， $y_r$  是期望响应， $y_m$  是网络模型响应。 $\rho$  值决定控制增量的平方和对性能指数的贡献。

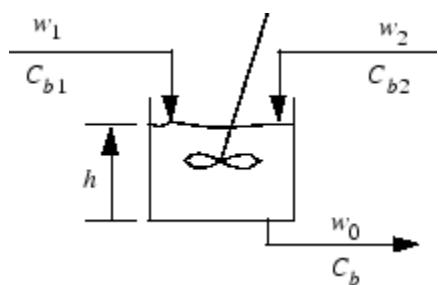
以下模块图说明模型预测控制过程。控制器由神经网络被控对象模型和优化模块组成。优化模块确定最小化  $J$  的  $u'$  的值，然后将最佳  $u$  值输入被控对象。控制器模块在 Simulink 中实现，如下节中所述。



## 使用神经网络预测控制器模块

本节说明如何使用 NN Predictive Controller 模块。第一步是将 Deep Learning Toolbox 模块库中的 NN Predictive Controller 模块复制到 Simulink Editor 中。如果您不确定如何操作，请参阅 Simulink 文档。在以下示例中跳过此步骤。

Deep Learning Toolbox 软件附带示例模型，以说明如何使用预测控制器。此示例使用具有催化作用的连续搅拌釜反应器 (CSTR)。下图显示过程图。



系统的动态模型是

$$\frac{dh(t)}{dt} = w_1(t) + w_2(t) - 0.2\sqrt{h(t)}$$

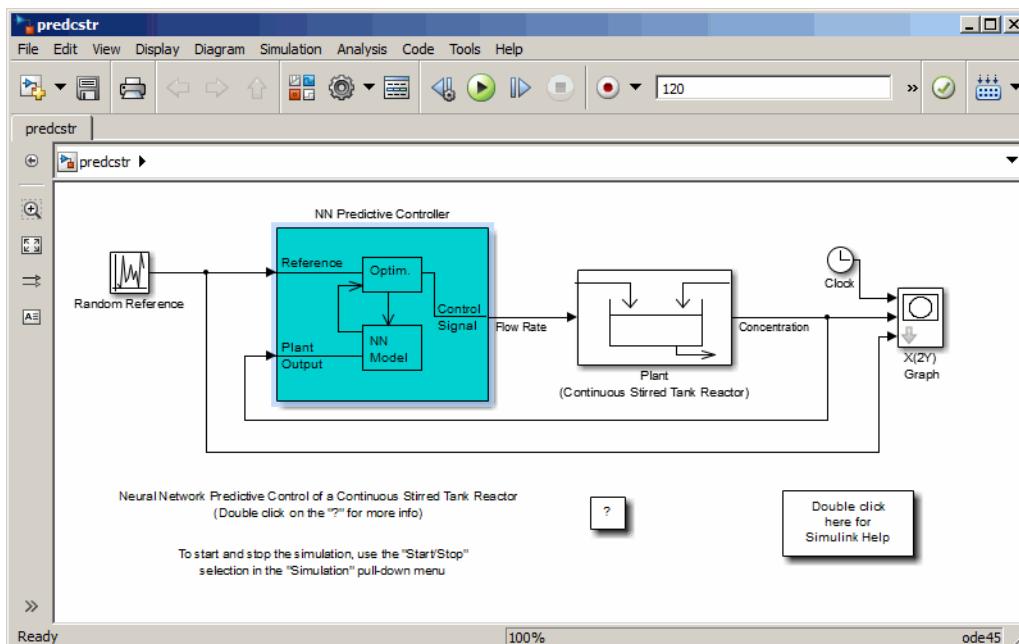
$$\frac{dC_b(t)}{dt} = (C_{b1} - C_b(t))\frac{w_1(t)}{h(t)} + (C_{b2} - C_b(t))\frac{w_2(t)}{h(t)} - \frac{k_1 C_b(t)}{(1 + k_2 C_b(t))^2}$$

其中,  $h(t)$  是液面,  $C_b(t)$  是处理过程输出的产品浓度,  $w_1(t)$  是浓缩进料  $C_{b1}$  的流速,  $w_2(t)$  是稀释进料  $C_{b2}$  的流速。输入浓度设置为  $C_{b1} = 24.9$ ,  $C_{b2} = 0.1$ 。与消耗速率相关联的常量是  $k_1 = 1$  和  $k_2 = 1$ 。

控制器的目标是通过调节流量  $w_1(t)$  来保持产品浓度。为了简化示例, 请将  $w_2(t)$  设置为 0.1。在本试验中, 水箱  $h(t)$  的液位不受控制。

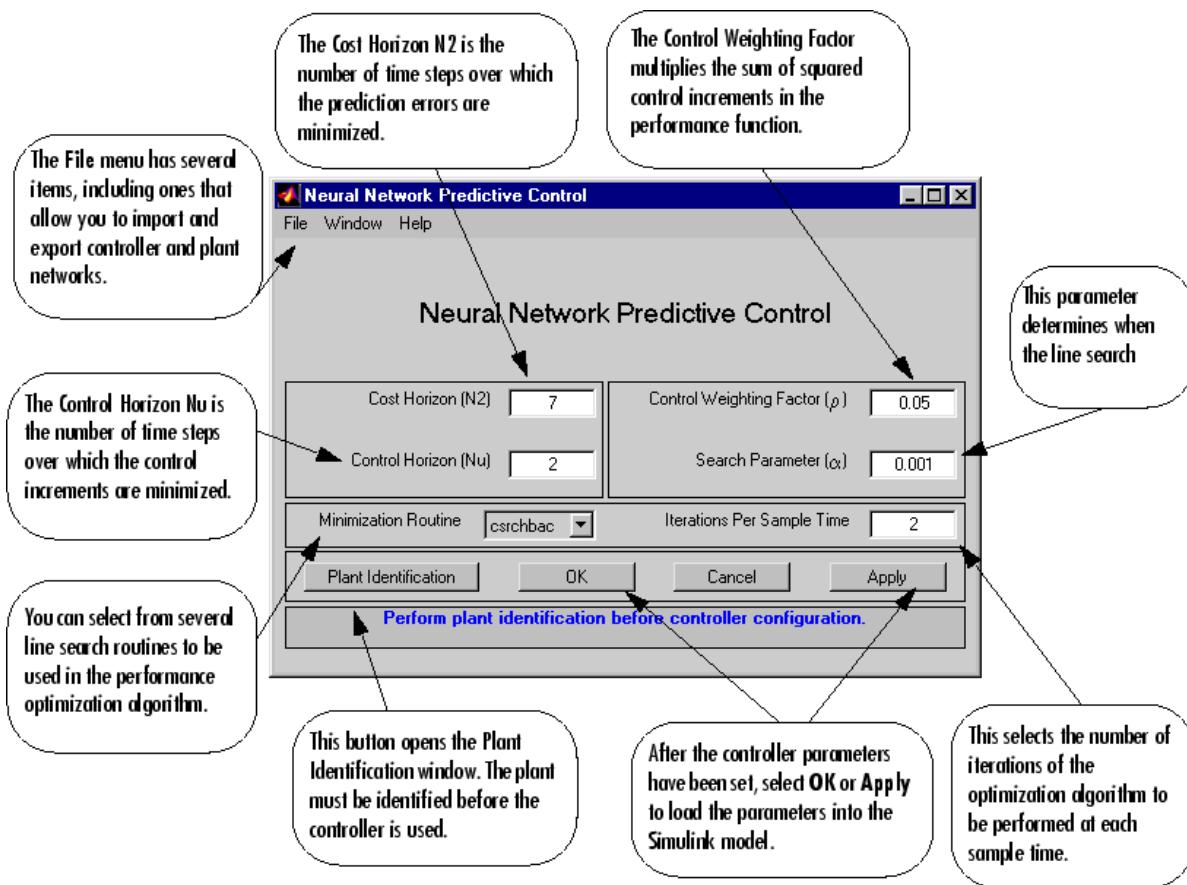
要运行此示例, 请执行下列步骤:

- 1 启动 MATLAB。
- 2 在 MATLAB 命令行窗口中键入 **predcstr**。该命令打开 Simulink Editor 并显示以下模型。

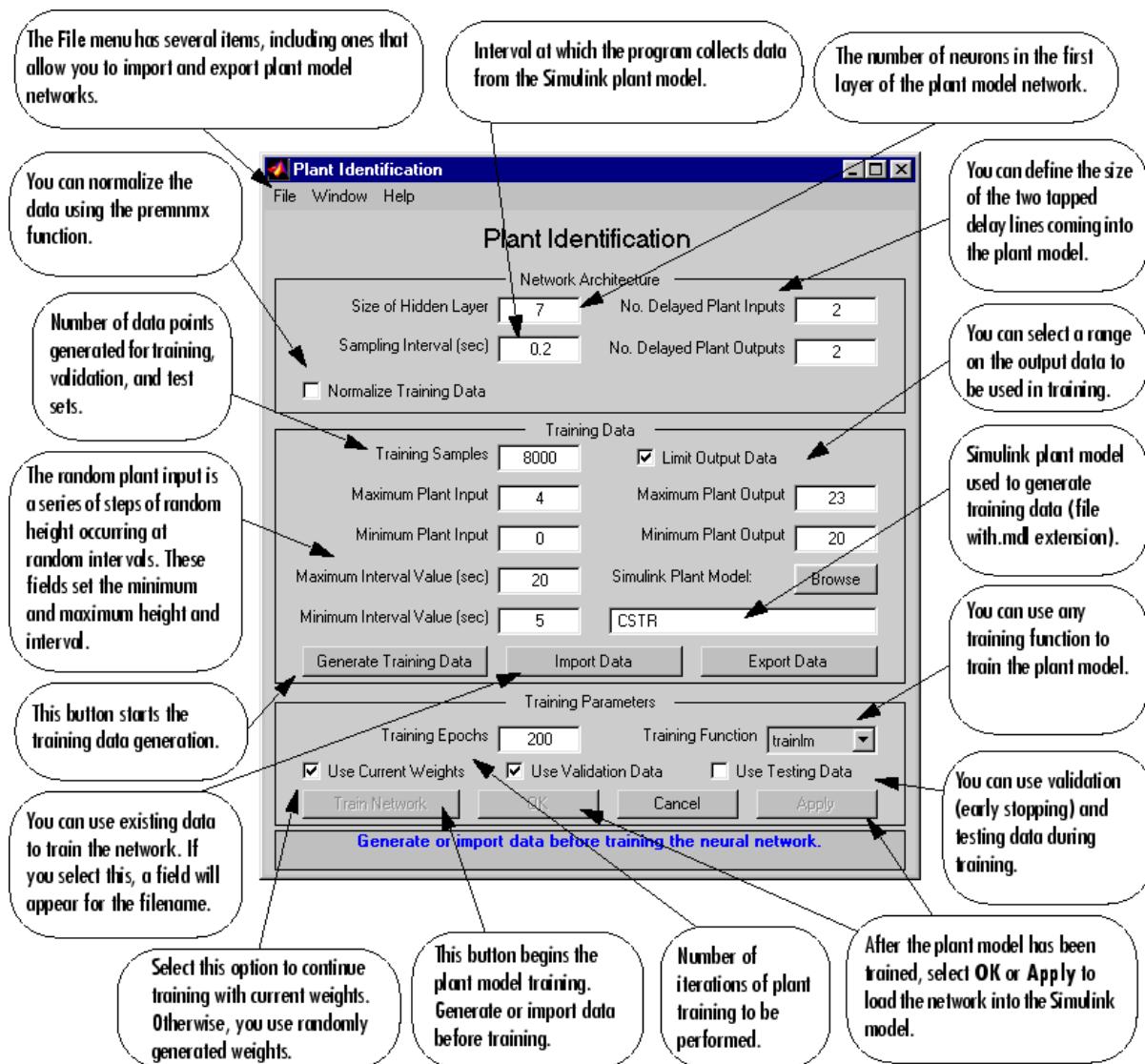


Plant 模块包含 Simulink CSTR 被控对象模型。NN Predictive Controller 模块信号的连接如下:

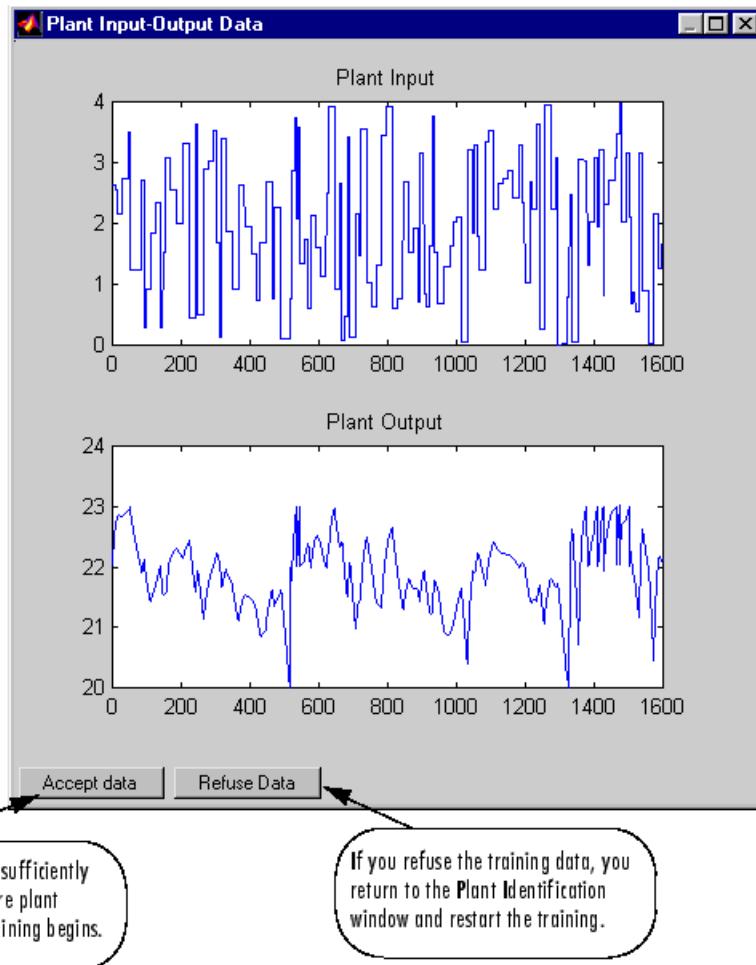
- Control Signal 连接到 Plant 模型的输入。
  - Plant Output 信号连接到 Plant 模块输出。
  - Reference 连接到 Random Reference 信号。
- 3 双击 NN Predictive Controller 模块。这将打开以下用于设计模型预测控制器的窗口。此窗口使您能够更改控制器时域  $N_2$  和  $N_u$ 。 $(N_1$  固定为 1。 $)$  前面所述的加权参数  $\rho$  也在此窗口中定义。参数  $\alpha$  用于控制优化。它决定性能降低多少才算一个成功的优化步。您可以选择优化算法使用哪个线性最小化例程, 还可以决定在每个采样时间内执行优化算法的迭代次数。这些线性最小化例程是基于“多层次神经网络与反向传播训练”(第 21-2 页) 中所述的那些例程进行了稍微修改的版本。



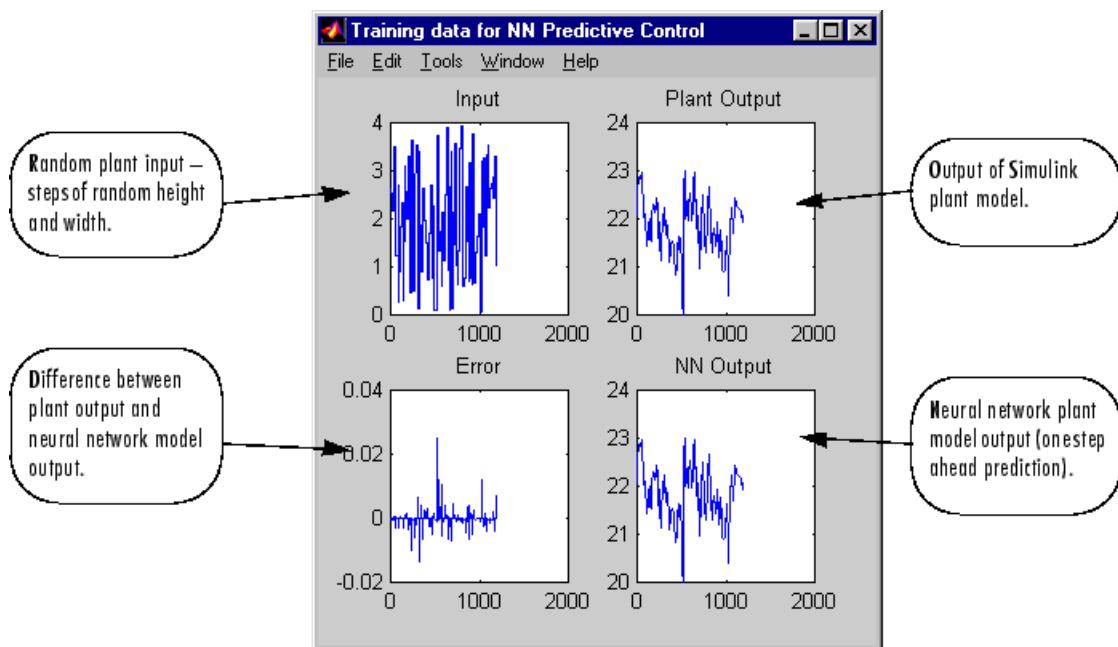
- 4 选择 **Plant Identification**。这将打开以下窗口。您必须开发神经网络被控对象模型，才能使用控制器。被控对象模型预测将来的被控对象输出。优化算法使用这些预测来确定用于优化将来性能的控制输入。如前所示，被控对象模型神经网络有一个隐含层。您可以在此窗口中选择该层的大小、延迟输入和延迟输出的数目以及训练函数。您可以选择“多层浅层神经网络与反向传播训练”（第 21-2 页）中所述的任何训练函数来训练神经网络被控对象模型。



- 5 点击 **Generate Training Data**。该计划通过将一系列随机阶跃输入应用于 Simulink 被控对象模型来生成训练数据。然后，潜在的训练数据将显示在与以下类似的图窗中。

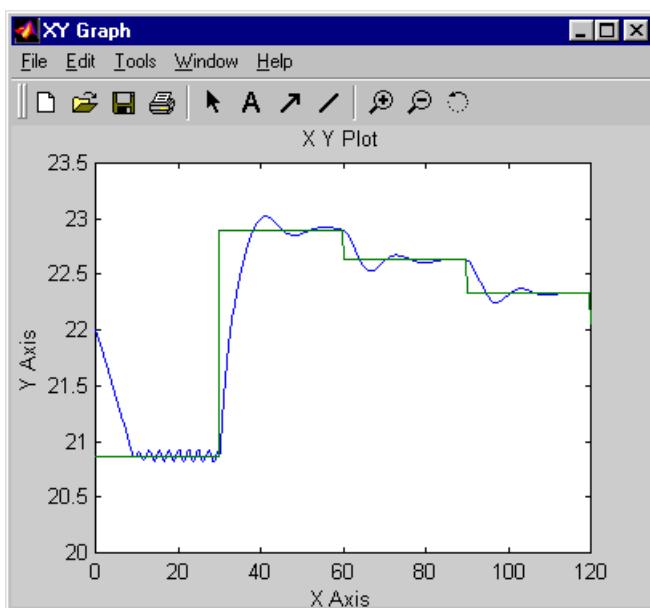


- 6 点击 **Accept Data**, 然后点击 Plant Identification 窗口中的 **Train Network**。被控对象模型训练开始。训练将根据您选择的训练算法（在本例中为 `trainlm`）进行。这是批量训练的简单应用，如“多层次浅层神经网络与反向传播训练”（第 21-2 页）中所述。训练完成后，将显示生成的被控对象模型的响应，如以下图窗中所示。（如果存在验证数据和测试数据，也会分别有单独的图。）



然后，您可以通过再次选择 **Train Network** 继续使用相同的数据集进行训练，您可以选择 **Erase Generated Data** 并生成新数据集，也可以接受当前被控对象模型并开始仿真闭环系统。对于此示例，开始仿真，如以下步骤所示。

- 7 在 Plant Identification 窗口中选择 **OK**。这会将经过训练的神经网络被控对象模型加载到 NN Predictive Controller 模块中。
- 8 在 Neural Network Predictive Control 窗口中选择 **OK**。这会将控制器参数加载到 NN Predictive Controller 模块中。
- 9 返回 Simulink Editor，选择菜单选项 **Simulation > Run** 开始仿真。当仿真运行时，显示被控对象输出和参考信号，如以下图窗中所示。



# 径向基神经网络

---

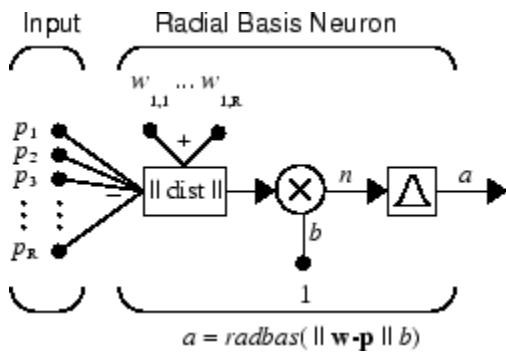
## 径向基神经网络

### 本节内容

- “神经元模型”（第 24-2 页）
- “网络架构”（第 24-3 页）
- “精确设计 (newrbe)”（第 24-3 页）
- “更高效的设计 (newrb)”（第 24-4 页）
- “示例”（第 24-5 页）

### 神经元模型

这是具有 R 个输入的径向基网络。

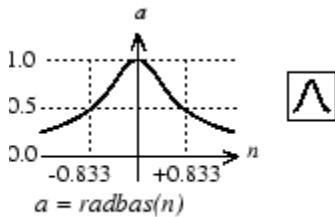


请注意，`radbas` 神经元的净输入的表达式与其他神经元的表达式不同。此处，`radbas` 传递函数的净输入是其权重向量  $w$  和输入向量  $p$  之间的向量距离乘以偏置  $b$ 。（此图窗中的  $\| \text{dist} \|$  框接受输入向量  $p$  和单行输入权重矩阵，并生成两者的点积。）

径向基神经元的传递函数为

$$\text{radbas}(n) = e^{-n^2}$$

以下是 `radbas` 传递函数的图。



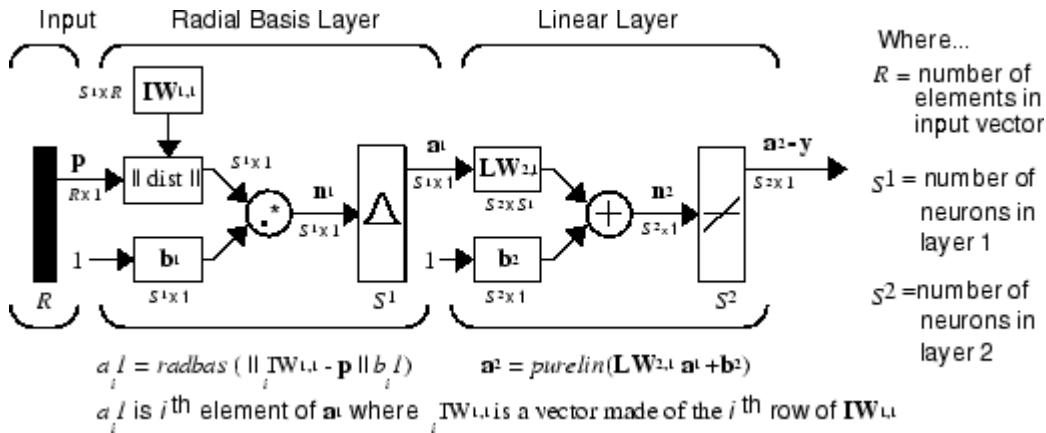
Radial Basis Function

当输入为 0 时，径向基函数具有最大值 1。随着  $w$  和  $p$  之间距离的减小，输出也会增大。因此，径向基神经元充当一个检测器，只要输入  $p$  与其权重向量  $w$  相同，该检测器就生成 1。

偏置  $b$  允许调整 `radbas` 神经元的敏感度。例如，如果神经元的偏置为 0.1，则对于任何输入向量  $p$ ，如果它与其权重向量  $w$  的向量距离为 8.326 (0.8326/b) 处，系统会输出 0.5。

## 网络架构

径向基网络由两个层组成： $S^1$  神经元构成的隐藏径向基层和由  $S^2$  神经元组成的输出线性层。



此图中的  $\| \text{dist} \|$  框接受输入向量  $p$  和输入权重矩阵  $\text{IW}^{1,1}$ ，并生成一个具有  $S_1$  个元素的向量。这些元素是输入向量和由输入权重矩阵的行形成的向量  $\| \text{IW}^{1,1}$  之间的距离。

偏置向量  $b^1$  和  $\| \text{dist} \|$  的输出使用 MATLAB 运算  $*$  进行合并运算，该运算执行逐元素乘法。

前馈网络 **net** 的第一层的输出可通过以下代码获得：

```
a{1} = radbas(netprod(dist(net.IW{1,1},p),net.b{1}))
```

幸运的是，您不必编写这样的代码行。设计该网络的所有细节都内置于设计函数 **newrbe** 和 **newrb** 中，您可以通过 **sim** 获得其输出。

通过跟踪输入向量  $p$  经过网络到输出  $a^2$  的过程，您可以了解该网络的行为。如果向这样的网络提交一个输入向量，径向基层中的每个神经元将根据输入向量与每个神经元的权重向量的接近程度输出一个值。

因此，权重向量与输入向量  $p$  差异很大的径向基神经元具有接近于零的输出。这些值很小的输出对线性输出神经元的影响可以忽略不计。

相反，权重向量接近输入向量  $p$  的径向基神经元生成接近 1 的值。如果神经元的输出为 1，它在第二层的输出权重会将其值传递给第二层中的线性神经元。

事实上，如果只有一个径向基神经元的输出为 1，而所有其他神经元的输出为 0（或非常接近 0），则线性层的输出将是活动神经元的输出权重。不过，这是极端情况。通常几个神经元的输出值总是存在不同程度的差异。

现在详细看看第一层是如何运作的。每个神经元的加权输入是输入向量与其权重向量之间的距离，用 **dist** 计算。每个神经元的净输入是其加权输入与其偏置的逐元素乘积，用 **netprod** 计算。每个神经元的输出是其通过 **radbas** 的净输入。如果一个神经元的权重向量等于输入向量（经过转置），则其加权输入为 0，其净输入为 0，其输出为 1。如果一个神经元的权重向量是 **spread** 到输入向量的距离，其加权输入是 **spread**，其净输入是 **sqrt(-log(.5))**（即 0.8326），因此其输出是 0.5。

## 精确设计 (newrbe)

您可以使用函数 **newrbe** 设计径向基网络。该函数可以生成一个在训练向量上具有零误差的网络。其调用方式如下：

```
net = newrbe(P,T,SPREAD)
```

函数 **newrbe** 接受输入向量 **P** 和目标向量 **T** 的矩阵，以及径向基层的分布常数 **SPREAD**，并返回一个具有权重和偏置的网络，使得当输入为 **P** 时，输出正好为 **T**。

此函数 **newrbe** 创建与 **P** 中输入向量的数目一样多的 **radbas** 神经元，并将第一层的权重设置为 **P'**。因此，存在一个由 **radbas** 神经元组成的层，其中每个神经元充当一个针对不同输入向量的检测器。如果有 **Q** 个输入向量，则会有 **Q** 个神经元。

第一层中的每个偏置设置为  $0.8326/SPREAD$ 。这给出了在加权输入为  $+/- SPREAD$  时跨度为 0.5 的径向基函数。这决定了每个神经元响应的输入空间中区域的宽度。如果 **SPREAD** 是 4，则对于与其权重向量的向量距离在 4 以内的任何输入向量，每个 **radbas** 神经元的响应为 0.5 或更大的值。**SPREAD** 应足够大，以便神经元对输入空间的重叠区域作出强响应。

通过仿真第一层输出  $a^1 (A\{1\})$ ，然后求解以下线性表达式，求得第二层权重  $|W^{2,1}$ （在代码中为 **IW{2,1}**）和偏置  $b^2$ （在代码中为 **b{2}**）：

$$[W\{2,1\} \ b\{2\}] * [A\{1\}; \ ones(1,Q)] = T$$

您知道第二层 (**A{1}**) 和目标 (**T**) 的输入，并且该层是线性的。您可以使用以下代码计算第二层的权重和偏置，以最小化误差平方和。

$$Wb = T/[A\{1\}; \ ones(1,Q)]$$

此处 **Wb** 包含权重和偏置，偏置在最后一列中。如下所述，误差平方和始终为 0。

假设一个问题有 **C** 个约束（输入/目标对组），每个神经元都有 **C + 1** 个变量（**C** 个权重来自 **C** 个 **radbas** 神经元，还有一个偏置）。如果一个线性问题具有 **C** 个约束和 **C** 个以上变量，则该线性问题具有无限数量的零误差解。

因此，**newrbe** 会创建一个在训练向量上具有零误差的网络。唯一需要的条件是确保 **SPREAD** 足够大，以便 **radbas** 神经元的活动输入区域存在足够的重叠，使得几个 **radbas** 神经元在任何给定时刻始终具有相当大的输出。这使得网络函数更平滑，从而对于介于设计中所用输入向量之间的新输入向量具有更强的泛化能力。（然而，**SPREAD** 不应太大，否则会导致每个神经元实际上都在同一个大的输入空间区域内进行响应。）

**newrbe** 的缺点是它生成的网络中的隐藏神经元和输入向量一样多。因此，在需要许多输入向量来正确定义网络时（通常情况下会这样），**newrbe** 不会返回可接受的解。

## 更高效的设计 (**newrb**)

函数 **newrb** 以每次创建一个神经元的迭代方式创建一个径向基网络。神经元添加到网络中，直到误差平方和低于误差目标或达到最大神经元数。对此函数的调用是

```
net = newrb(P,T,GOAL,SPREAD)
```

函数 **newrb** 接受输入向量 **P** 和目标向量 **T** 的矩阵，以及设计参数 **GOAL** 和 **SPREAD**，并返回所需的网络。

**newrb** 的设计方法与 **newrbe** 相似。不同的是，**newrb** 一次创建多个神经元。在每次迭代中，使网络误差降幅最大的输入向量用于创建一个 **radbas** 神经元。检查新网络的误差，如果误差足够低，则结束运行 **newrb**。否则添加下一个神经元。重复此过程，直到满足误差目标或达到最大神经元数量。

与 **newrbe** 一样，分布参数必须足够大，以便 **radbas** 神经元对输入空间的重叠区域作出响应，但不能大到所有神经元都以基本相同的方式作出响应。

为什么不始终使用径向基网络代替标准前馈网络？径向基网络，即使是使用 **newrbe** 高效设计的，其神经元数量也往往比在隐藏层中具有 **tansig** 或 **logsig** 神经元的前馈网络多许多倍。

这是因为，**sigmoid** 神经元可以针对输入空间中的一个大区域提供输出，而 **radbas** 神经元只对输入空间中相对较小的区域作出响应。结果是，输入空间越大（根据输入的数目以及这些输入的变化范围），所需的 **radbas** 神经元就越多。

另一方面，设计径向基网络花费的时间通常远远少于训练 **sigmoid/linear** 网络，有时还只需使用更少的神经元，如以下示例中所示。

## 示例

示例“径向基逼近”（第 30-74 页）说明如何使用径向基网络来拟合函数。在此处，只用五个神经元就完成了问题的求解。

示例“径向基神经元欠叠”（第 30-78 页）和“径向基神经元过叠”（第 30-80 页）研究分布常数如何影响径向基网络的设计过程。

在“径向基神经元欠叠”（第 30-78 页）中，径向基网络设计用于求解与“径向基逼近”（第 30-74 页）中相同的问题。然而，这次使用的分布常数是 0.01。因此，对于距离其权重向量为 0.01 或更大值的任何输入向量，每个径向基神经元返回 0.5 或更低的值。

由于训练输入以 0.1 的间隔出现，因此对于任一给定输入，没有两个径向基神经元具有强输出。

“径向基神经元欠叠”（第 30-78 页）表明，分布常数过小可能会得到不是从设计中所用输入/目标向量泛化产生的解。示例“径向基神经元过叠”（第 30-80 页）展示了相反的问题。如果分布常数足够大，则对于用于设计网络的所有输入，径向基神经元将输出较大的值（接近 1.0）。

如果所有径向基神经元始终输出 1，则提交给网络的任何信息都将丢失。不管输入是什么，第二层始终输出 1。函数 **newrb** 将尝试找到一个网络，但由于在这种情况下出现的数值问题而找不到。

以上示例说明，选择的分布常数应大于相邻输入向量之间的距离（以获得良好的泛化能力），但小于跨整个输入空间的距离。

对于上述问题，这意味着选取的分布常数应大于 0.1（各输入之间的间隔）且小于 2（最左边和最右边输入之间的距离）。



# 自组织和学习向量量化网络

---

## 自组织映射神经网络的聚类

### 本节内容

- “拓扑 (gridtop、hextop、randtop) ” (第 25-3 页)
- “距离函数 (dist、linkdist、mandist、boxdist) ” (第 25-6 页)
- “架构” (第 25-8 页)
- “创建自组织映射神经网络 (selforgmap)” (第 25-8 页)
- “训练 (learnsomb)” (第 25-10 页)
- “示例” (第 25-11 页)

自组织特征映射 (SOFM) 学习根据输入向量在输入空间中的分组方式对输入向量进行分类。它们与竞争层的不同之处在于，自组织映射中的相邻神经元会学习识别输入空间的相邻部分。因此，自组织映射会同时学习训练时所基于的输入向量的分布（如竞争层所做的一样）和拓扑。

SOFM 的层中的神经元最初根据拓扑函数排列在物理位置中。函数 `gridtop`、`hextop` 或 `randtop` 可以将神经元排列成网格、六边形或随机拓扑。神经元之间的距离是使用距离函数根据其位置进行计算的。有四个距离函数：`dist`、`boxdist`、`linkdist` 和 `mandist`。连接距离是最常见的。这些拓扑和距离函数在“拓扑 (gridtop、hextop、randtop) ” (第 25-3 页) 和“距离函数 (dist、linkdist、mandist、boxdist) ” (第 25-6 页) 中进行了介绍。

此处，自组织特征映射网络使用与竞争层相同的过程来识别获胜神经元  $i^*$ 。然而，不是仅更新获胜神经元，而是使用 Kohonen 规则更新获胜神经元的特定邻域  $N_{i^*}(d)$  内的所有神经元。具体来说，按如下方式调整所有这样的神经元  $i \in N_{i^*}(d)$ ：

$$i\mathbf{w}(q) = i\mathbf{w}(q - 1) + \alpha(\mathbf{p}(q) - i\mathbf{w}(q - 1))$$

或

$$i\mathbf{w}(q) = (1 - \alpha)i\mathbf{w}(q - 1) + \alpha\mathbf{p}(q)$$

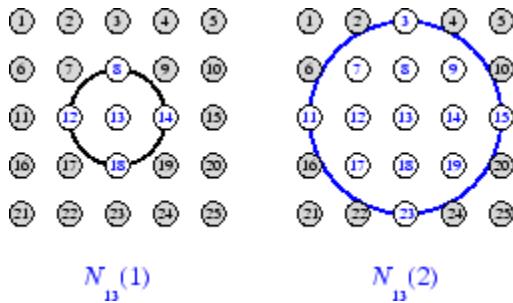
此处的邻域  $N_{i^*}(d)$  包含位于获胜神经元  $i$  的半径  $d$  范围内的所有神经元的索引\*。

$$N_i(d) = \{j, d_{ij} \leq d\}$$

因此，当提交向量  $\mathbf{p}$  时，获胜神经元和其近邻的权重向  $\mathbf{p}$  移动。因此，多次提交输入后，邻近神经元就学习了彼此相似的向量。

SOFM 训练的另一个版本称为批量算法，它会在将整个数据集提交给网络后再更新权重。然后，该算法为每个输入向量确定一个获胜神经元。随后将每个权重向量移至特定位置，即它作为获胜神经元或位于获胜神经元的邻域时的所有对应输入向量的平均位置。

为了说明邻域的概念，请参考下图。左图显示神经元 13 周围半径  $d = 1$  的二维邻域。右图显示半径  $d = 2$  的邻域。



这些邻域可以写为  $N_{13}(1) = \{8, 12, 13, 14, 18\}$  和  $N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}$ 。

SOFM 中的神经元不必以二维模式排列。您可以使用一维排列，也可以使用三维或多维排列。对于一维 SOFM，一个神经元在半径为 1 的范围内只有两个邻点（如果该神经元位于行的末端，则只有一个邻点）。您也可以用不同方式定义距离，例如，使用神经元和邻域的矩形和六边形排列。网络的性能对邻域的确切形状并不敏感。

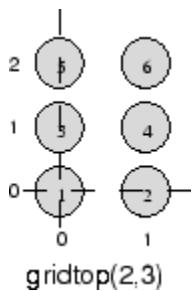
## 拓扑 (gridtop、hextop、randtop)

您可以使用函数 **gridtop**、**hextop** 和 **randtop** 为原始神经元位置指定不同拓扑。

**gridtop** 拓扑以矩形网格中的神经元开始，类似于上图所示。例如，假设您需要一个由 6 个神经元组成的  $2 \times 3$  数组。您可以使用以下代码来实现它：

```
pos = gridtop([2, 3])
pos =
  0   1   0   1   0   1
  0   0   1   1   2   2
```

此处，神经元 1 具有位置 (0,0)，神经元 2 具有位置 (1,0)，神经元 3 具有位置 (0,1) 等。

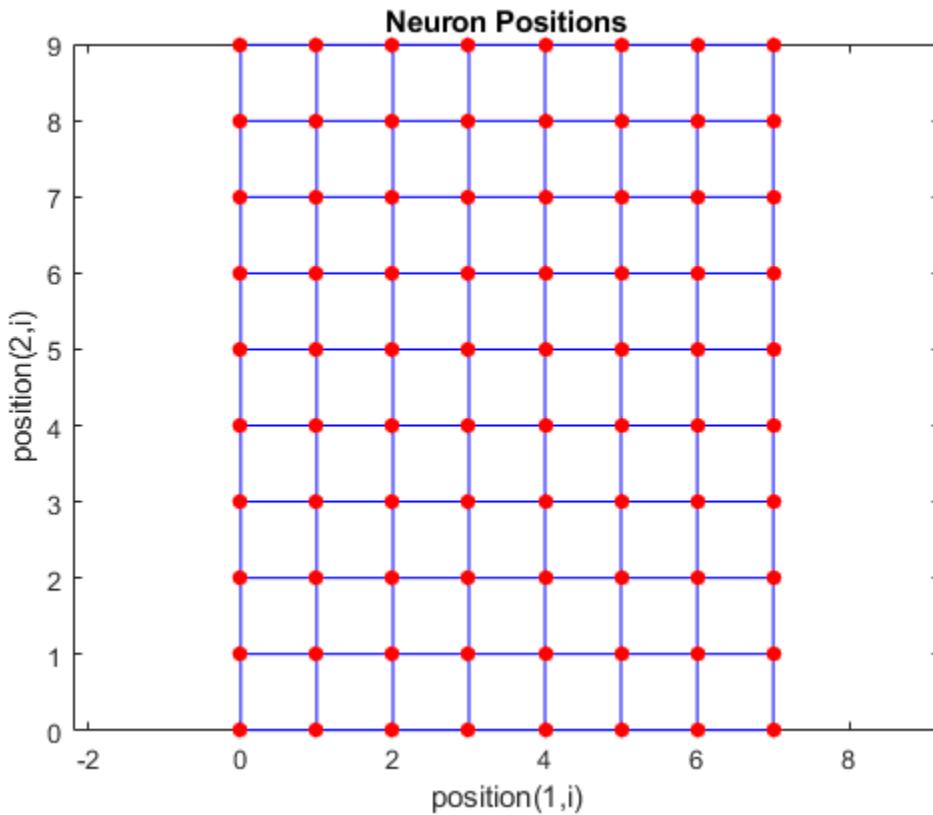


请注意，如果您要求维度大小颠倒的 **gridtop**，您会得到稍微不同的排列：

```
pos = gridtop([3, 2])
pos =
  0   1   2   0   1   2
  0   0   0   1   1   1
```

您可以使用以下代码在 **gridtop** 拓扑中创建一组  $8 \times 10$  神经元：

```
pos = gridtop([8 10]);
plotsom(pos)
```



如图所示，**gridtop** 拓扑中的神经元确实位于一个网格上。

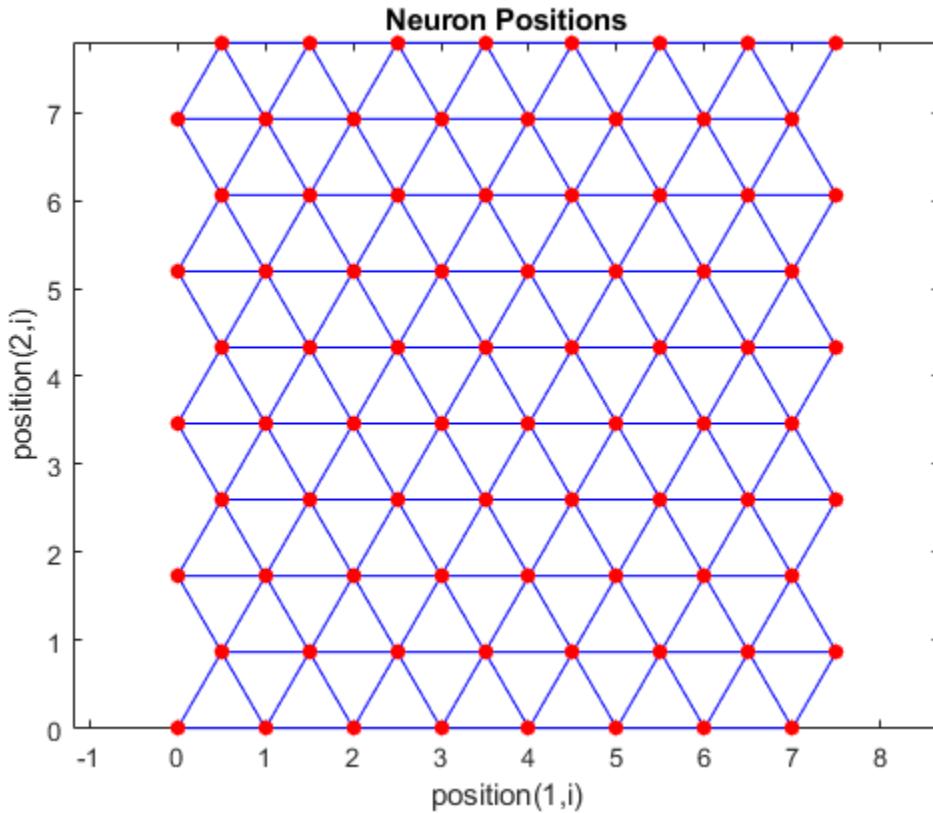
**hextop** 函数创建一组相似的神经元，但它们采用六边形模式。生成的  $2 \times 3$  模式的 **hextop** 神经元如下：

```
pos = hextop([2, 3])
pos =
    0   1.0000   0.5000   1.5000      0   1.0000
    0       0   0.8660   0.8660   1.7321   1.7321
```

请注意，**hextop** 是使用 **selforgmap** 生成的 SOM 网络的默认模式。

您可以使用以下代码以 **hextop** 拓扑创建并绘制一组  $8 \times 10$  神经元：

```
pos = hextop([8 10]);
plotsom(pos)
```



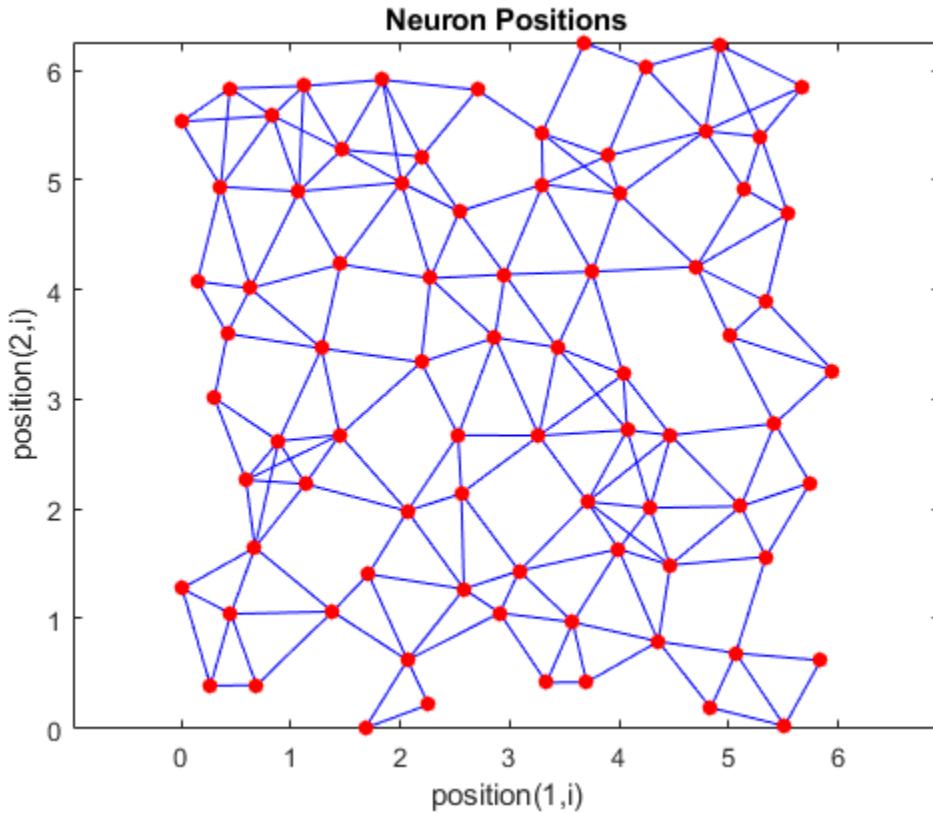
注意六边形排列的神经元的位置。

最后，`randtop` 函数以 N 维随机模式创建神经元。以下代码生成一个随机神经元模式。

```
pos = randtop([2, 3])
pos =
    0   0.7620   0.6268   1.4218   0.0663   0.7862
    0.0925      0   0.4984   0.6007   1.1222   1.4228
```

您可以使用以下代码以 `randtop` 拓扑创建并绘制一组  $8 \times 10$  神经元：

```
pos = randtop([8 10]);
plotsom(pos)
```



有关示例，请参阅这些拓扑函数的帮助。

### 距离函数 (**dist**、**linkdist**、**mandist**、**boxdist**)

在此工具箱中，有四种方法可以计算特定神经元到其邻点的距离。每种计算方法都用一个特殊函数来实现。

**dist** 函数计算从主神经元到其他神经元的欧几里德距离。假设您有三个神经元：

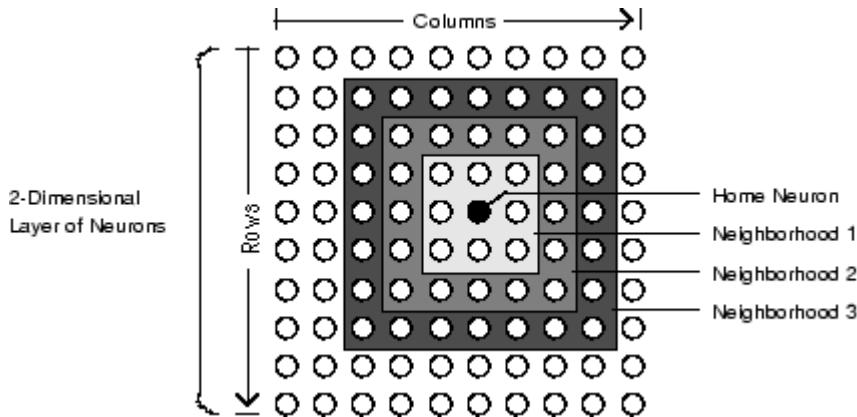
```
pos2 = [0 1 2; 0 1 2]
pos2 =
    0   1   2
    0   1   2
```

您可以使用以下代码找到每个神经元与其他神经元之间的距离：

```
D2 = dist(pos2)
D2 =
    0   1.4142   2.8284
   1.4142       0   1.4142
   2.8284   1.4142       0
```

因此，神经元 1 到自身的距离是 0，从神经元 1 到神经元 2 的距离是 1.4142 等。

下图显示二维 (**gridtop**) 神经元层中的一个主神经元。该主神经元周围有直径越来越大的邻域。直径为 1 的邻域包含该主神经元及其紧挨的邻近神经元。直径为 2 的邻域包含直径为 1 的神经元及其紧挨的邻近神经元。



对于 **dist** 函数，S 神经元层映射的所有邻域都由距离的  $S \times S$  矩阵表示。上面显示的特定距离（在紧挨的邻域中为 1，在邻域 2 中为 2，等等）由函数 **boxdist** 生成。假设在 **gridtop** 配置中有六个神经元。

```
pos = gridtop([2, 3])
pos =
  0   1   0   1   0   1
  0   0   1   1   2   2
```

则框距离为

```
d = boxdist(pos)
d =
  0   1   1   1   2   2
  1   0   1   1   2   2
  1   1   0   1   1   1
  1   1   1   0   1   1
  2   2   1   1   0   1
  2   2   1   1   1   0
```

从神经元 1 到神经元 2、3 和 4 的距离只有 1，因为它们在紧挨邻域中。神经元 1 到 5 和 6 的距离均为 2。从神经元 3 和 4 到所有其他神经元的距离仅为 1。

到一个神经元的连接距离是到达所关注的神经元所必须经过的连接数或步数。因此，如果您使用 **linkdist** 计算到同一组神经元的距离，得到的结果如下

```
dlink =
  0   1   1   2   2   3
  1   0   2   1   3   2
  1   2   0   1   1   2
  2   1   1   0   2   1
  2   3   1   2   0   1
  3   2   2   1   1   0
```

两个向量 **x** 和 **y** 之间的 Manhattan 距离的计算方式如下

```
D = sum(abs(x-y))
```

因此，如果您有

```
W1 = [1 2; 3 4; 5 6]
W1 =
  1   2
```

3	4
5	6

和

```
P1 = [1;1]
P1 =
1
1
```

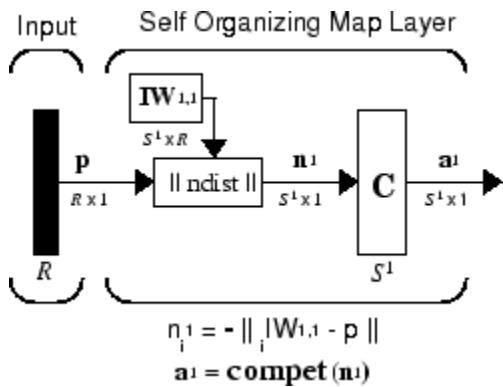
则您得到距离

```
Z1 = mandist(W1,P1)
Z1 =
1
5
9
```

用 `mandist` 计算的距离确实符合上面给出的数学表达式。

## 架构

此 SOFM 的架构如下所示。



此架构类似于竞争网络的架构，不同之处是此处不使用偏置。对于与  $i^*$  (获胜神经元) 对应的输出元素  $a^1_{i^*}$ ，竞争传递函数会输出 1。 $a^1$  中的所有其他输出元素均为 0。

然而，如上所述，现在靠近获胜神经元的神经元随获胜神经元一起更新。您可以从不同神经元拓扑中进行选择。同样，您可以从各种距离表达式中进行选择，来计算靠近获胜神经元的神经元。

## 创建自组织映射神经网络 (selforgmap)

您可以使用函数 `selforgmap` 创建一个新 SOM 网络。该函数定义在两个学习阶段中使用的变量：

- 排序阶段学习率
- 排序阶段步骤
- 调整阶段学习率
- 调整阶段领域距离

这些值用于训练和自适应。

请参考以下示例。

假设您要创建一个包含由两个元素组成的输入向量的网络，并且要在六边形  $2 \times 3$  网络中包含六个神经元。获得该网络的代码如下：

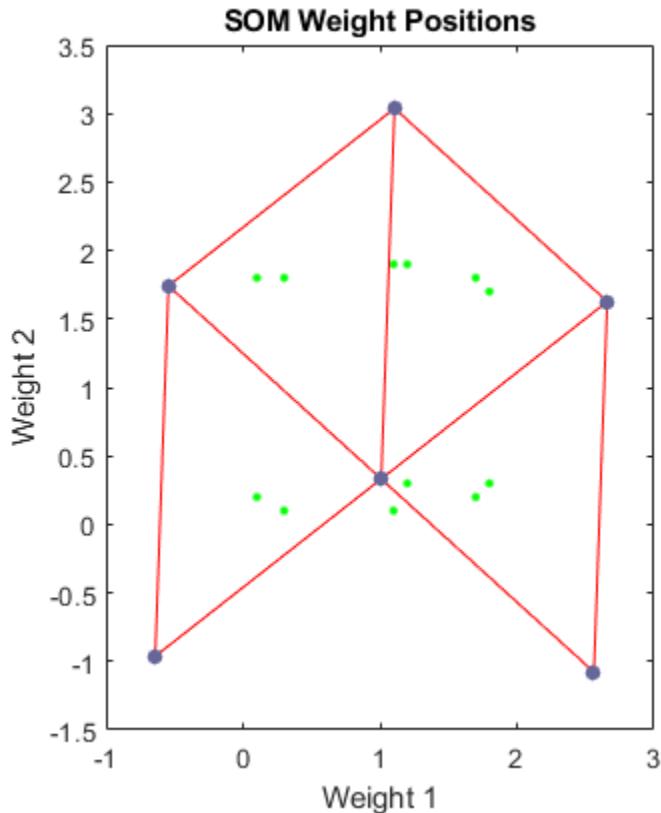
```
net = selforgmap([2 3]);
```

假设要用于训练的向量是：

```
P = [.1 .3 1.2 1.1 1.8 1.7 .1 .3 1.2 1.1 1.8 1.7;...  
0.2 0.1 0.3 0.1 0.3 0.2 1.8 1.8 1.9 1.9 1.7 1.8];
```

您可以使用以下代码配置网络以输入数据并绘制所有这些数据：

```
net = configure(net,P);  
plotsompos(net,P)
```



绿点是训练向量。`selforgmap` 的初始化将初始权重分布到整个输入空间。请注意，它们最初离训练向量有一定距离。

当对网络进行仿真时，计算每个神经元的权重向量和输入向量之间的负距离 (`negdist`) 以获得加权输入。加权输入也是净输入 (`netsum`)。净输入相互竞争 (`compet`)，因此只有具有最大正值净输入的神经元才会输出 1。

## 训练 (learnsomb)

自组织特征映射中的默认学习在批量模式 (trainbu) 下进行。自组织映射的权重学习函数是 **learnsomb**。

首先，网络为每个输入向量确定获胜神经元。随后将每个权重向量移至特定位置，即它作为获胜神经元或位于获胜神经元的邻域时的所有对应输入向量的平均位置。定义邻域大小的距离在训练过程中的两个阶段会更改。

### 排序阶段

该阶段持续给定的步数。邻域距离从给定的初始距离开始，并减小到调整邻域距离 (1.0)。随着邻域距离在此阶段逐步减小，网络的神经元通常在输入空间中以与它们自己的物理排序相同的拓扑进行排序。

### 调整阶段

此阶段持续训练或自适应阶段的其余部分。邻域大小已降至 1 以下，因此只有获胜神经元对每个样本进行学习。

现在看看这些网络中常用的一些特定值。

根据 **learnsomb** 学习参数进行学习，此处显示其默认值。

学习参数	默认值	目的
LP.init_neighborhood	3	初始邻域大小
LP.steps	100	排序阶段步骤

邻域大小 NS 在两个阶段中进行更改：排序阶段和调整阶段。

排序阶段持续的步骤与 LP.steps 一样多。在此阶段，算法将 ND 从初始邻域大小 LP.init\_neighborhood 向下调整为 1。在此阶段，神经元权重在输入空间中自行排序，且顺序应与相关联的神经元的位置一致。

在调整阶段，ND 小于 1。在此阶段，权重应该在输入空间中相对均匀地分布，同时保持在排序阶段建立的拓扑顺序。

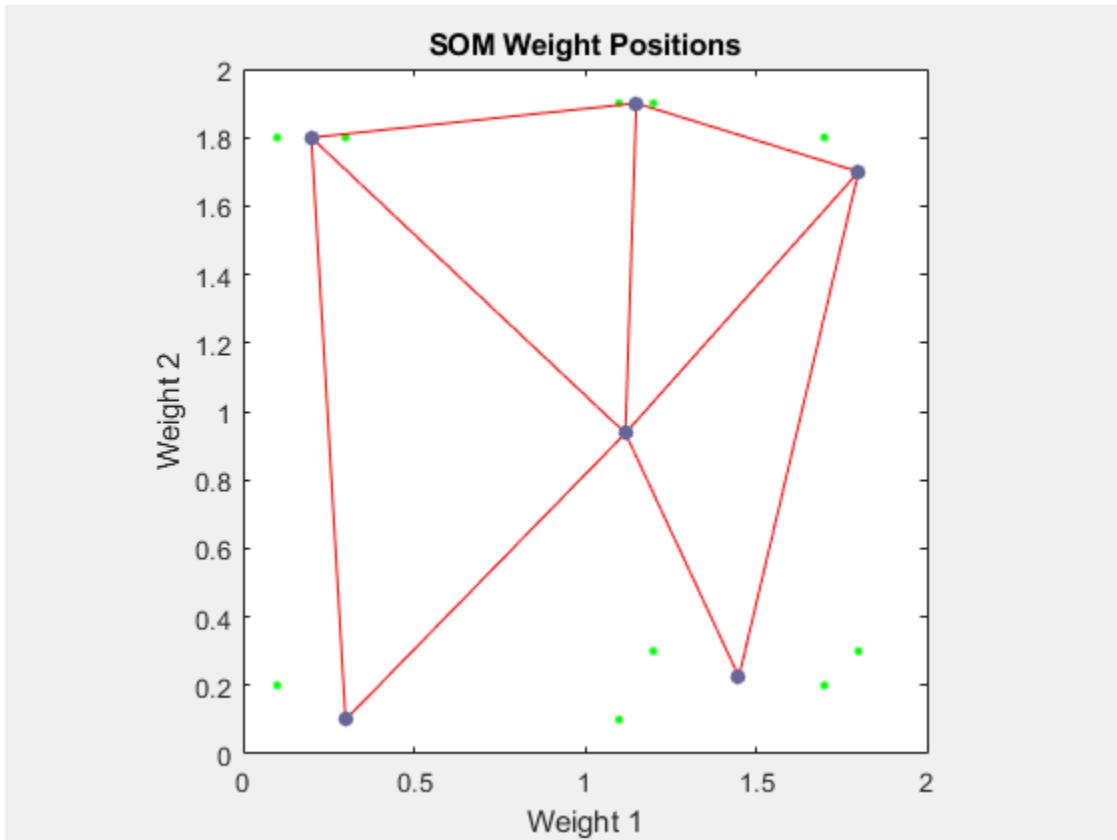
因此，神经元的权重向量最初全部一起以较大步幅向输入空间中出现输入向量的区域前进。然后，当邻域大小减小到 1 时，映射趋于根据提交的输入向量对自身进行拓扑排序。一旦邻域大小为 1，网络应很好地完成排序。训练继续进行，以便给神经元时间来在输入向量中均匀分布。

与竞争层一样，如果输入向量以均匀概率出现在输入空间的一部分中，自组织映射的神经元将按照它们之间近似相等的距离来对自身排序。如果各输入向量在整个输入空间中以不同频率出现，则特征映射层倾向于将神经元分配给与该处输入向量的频率成比例的区域。

因此，特征图在学习对其输入进行分类的同时，也学习其输入的拓扑和分布。

您可以使用以下代码对网络进行 1000 轮训练

```
net.trainParam.epochs = 1000;
net = train(net,P);
plotsompos(net,P)
```



您可以看到神经元已开始向不同训练组移动。需要额外的训练来使神经元更接近不同组。

如前所述，在神经元更新其权重方面，自组织映射不同于传统的竞争学习。特征图不是只更新获胜神经元，而是更新获胜神经元及其邻点的权重。结果是相邻的神经元倾向于具有相似的权重向量，并对相似的输入向量作出响应。

## 示例

下面简要介绍两个示例。您也可以尝试类似的示例“一维自组织映射”（第 30-69 页）和“二维自组织映射”（第 30-71 页）。

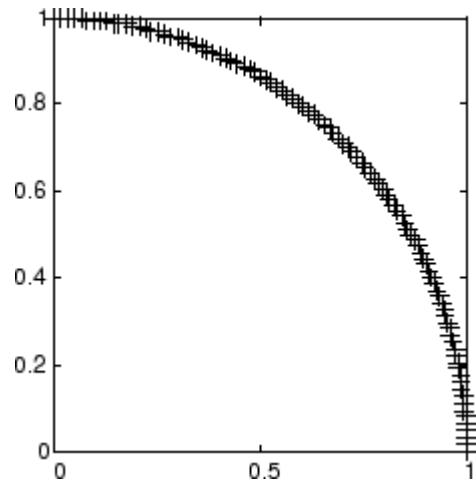
### 一维自组织映射

假设 100 个二元素单位输入向量均匀分布在  $0^\circ$  和  $90^\circ$  之间。

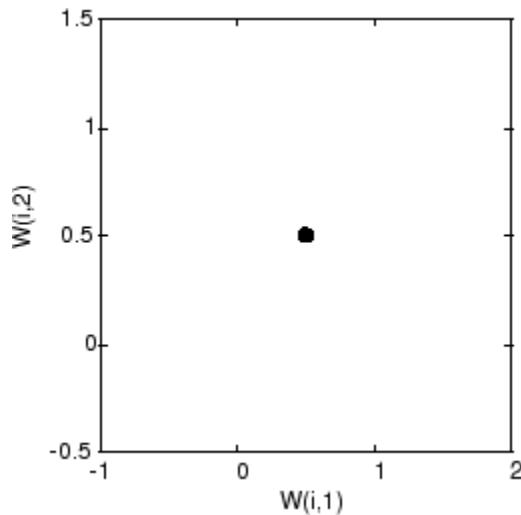
```
angles = 0:0.5*pi/99:0.5*pi;
```

以下是数据图。

```
P = [sin(angles); cos(angles)];
```

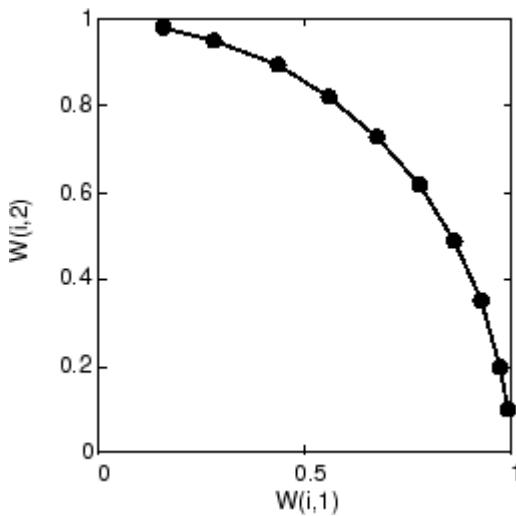


自组织映射定义为一个包含 10 个神经元的一维层。该图将根据上述输入向量进行训练。最初，这些神经元位于图形的中心。



当然，由于所有权重向量都从输入向量空间的中间开始，您现在看到的只是一个圆。

随着训练的开始，权重向量一起向输入向量移动。随着邻域大小的减小，它们也会进行排序。最后，该层会调整其权重，使得每个神经元对输入向量占据的输入空间区域作出强响应。相邻神经元权重向量的位置也反映输入向量的拓扑。



请注意，自组织映射是用随机顺序的输入向量来训练的，因此从相同的初始向量开始并不能保证获得相同的训练结果。

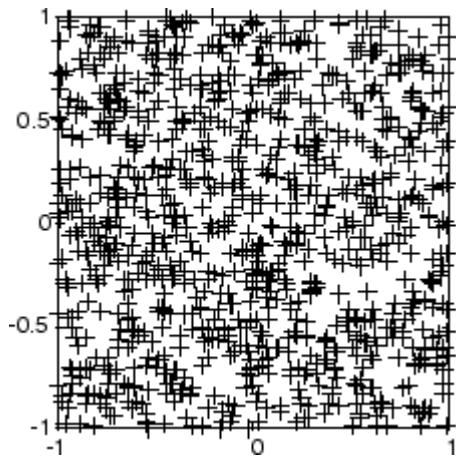
## 二维自组织映射

此示例说明如何训练二维自组织映射。

首先，用以下代码创建一些随机输入数据：

```
P = rands(2,1000);
```

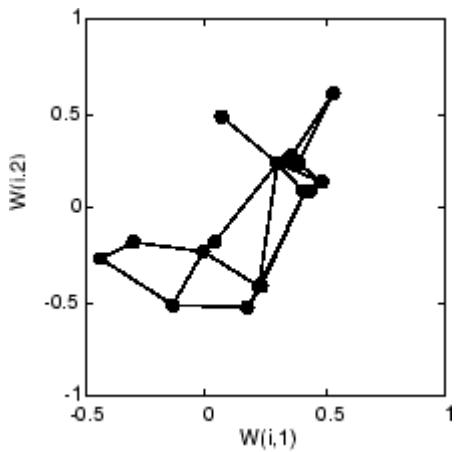
这是以下 1000 个输入向量的图。



使用一个包含 30 个神经元的  $5 \times 6$  二维图对这些输入向量进行分类。该二维图由  $5 \times 6$  神经元组成，根据 Manhattan 距离邻域函数 **mandist** 计算距离。

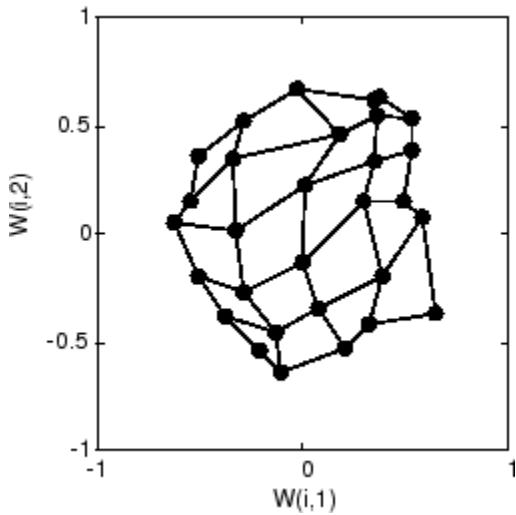
然后，通过 5000 个数据提交周期来训练映射，每 20 个周期显示一次。

这是 40 个周期后的自组织映射。



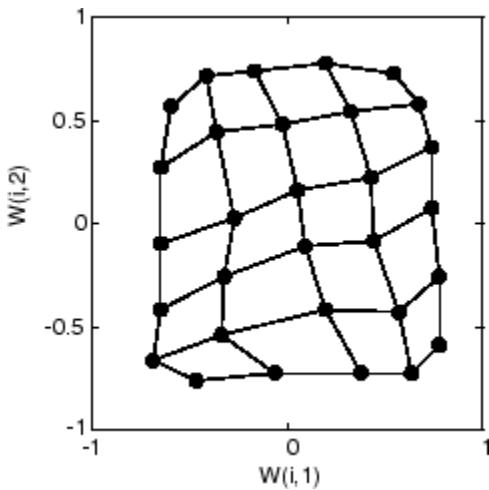
用圆表示的权重向量几乎是随机放置的。然而，即使只经过 40 个数据提交周期，由线连接的相邻神经元也有了紧密靠近的权重向量。

以下是 120 个周期后的图。

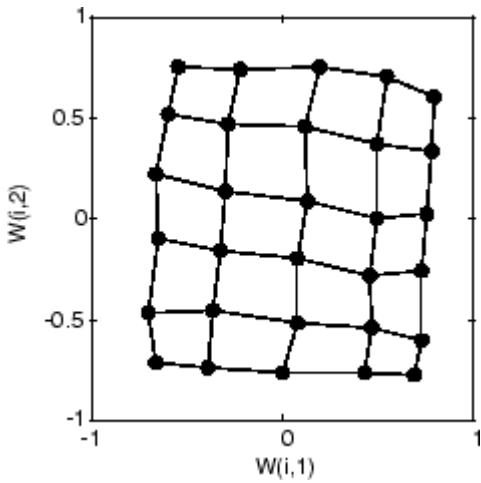


120 个周期后，图开始根据输入空间的拓扑来组织自身，这将限制输入向量。

以下是在 500 个周期后在输入空间中更均匀分布的图。



最后，以下是在 5000 个周期后在输入空间中分布相当均匀的图。此外，神经元的间距非常均匀，反映在此问题中输入向量均匀分布。



因此，二维自组织映射已学习了其输入空间的拓扑。

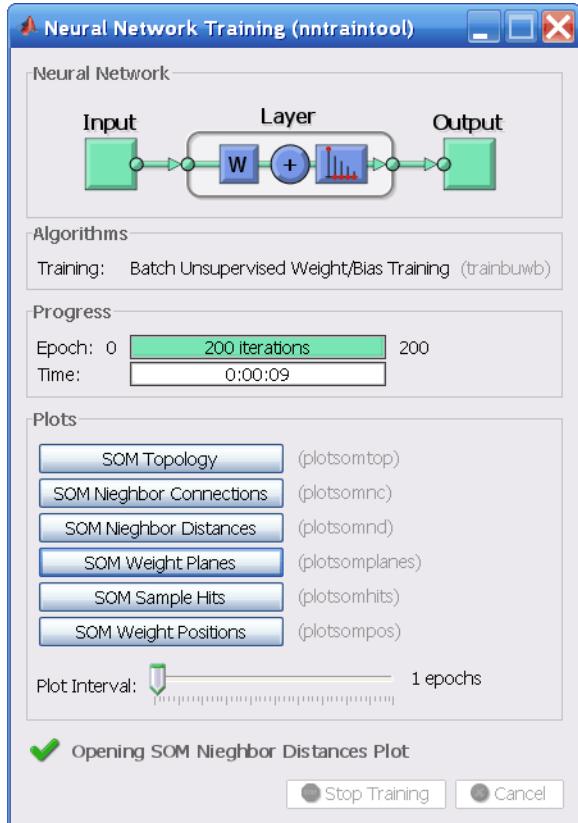
必须注意，虽然自组织映射不需要很长时间来组织自身即可使相邻神经元识别相似的输入，但映射最终根据输入向量的分布来排列自身可能需要很长时间。

### 用批量算法进行训练

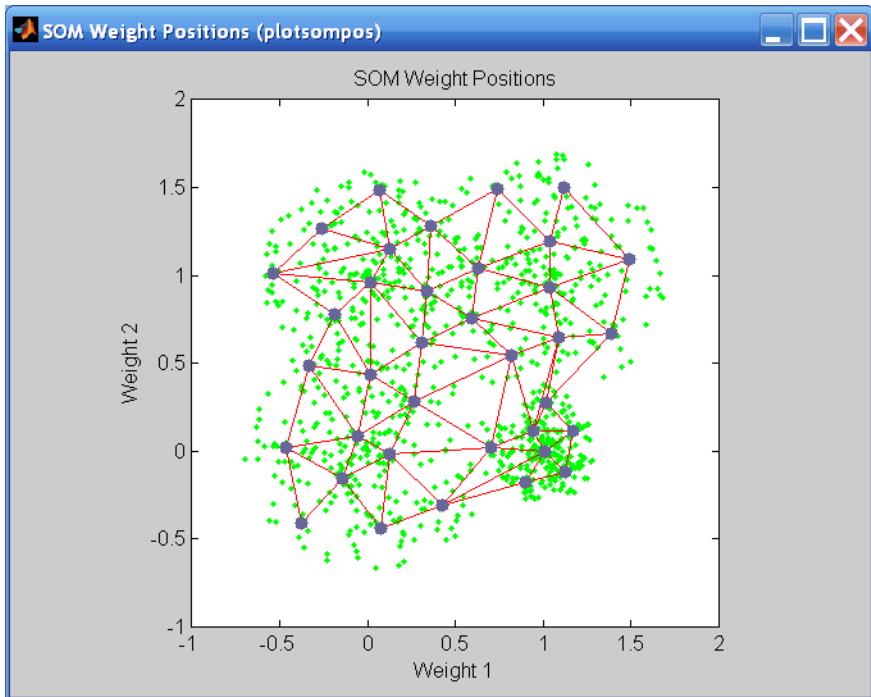
批量训练算法通常比增量算法快得多，并且是 SOFM 训练的默认算法。您可以使用以下命令对简单数据集试验该算法：

```
x = simplecluster_dataset
net = selforgmap([6 6]);
net = train(net,x);
```

此命令序列创建并训练一个由 36 个神经元组成的  $6 \times 6$  二维图。在训练过程中，出现以下图窗。



您可以从该窗口访问几个有用的可视化。如果您点击 **SOM Weight Positions**，将出现以下图窗，其中显示数据点和权重向量的位置。如该图窗所示，批量算法仅经过 200 次迭代，图就在输入空间中得到很好的分布。

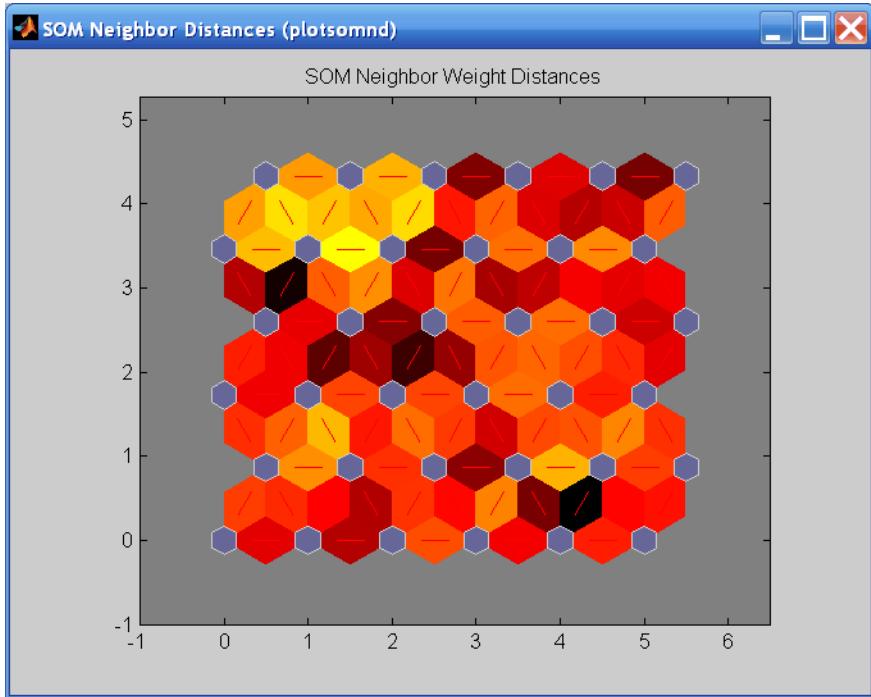


当输入空间是高维空间时，您无法同时可视化所有权重。在本例中，请点击 **SOM Neighbor Distances**。以下图窗显示相邻神经元之间的距离。

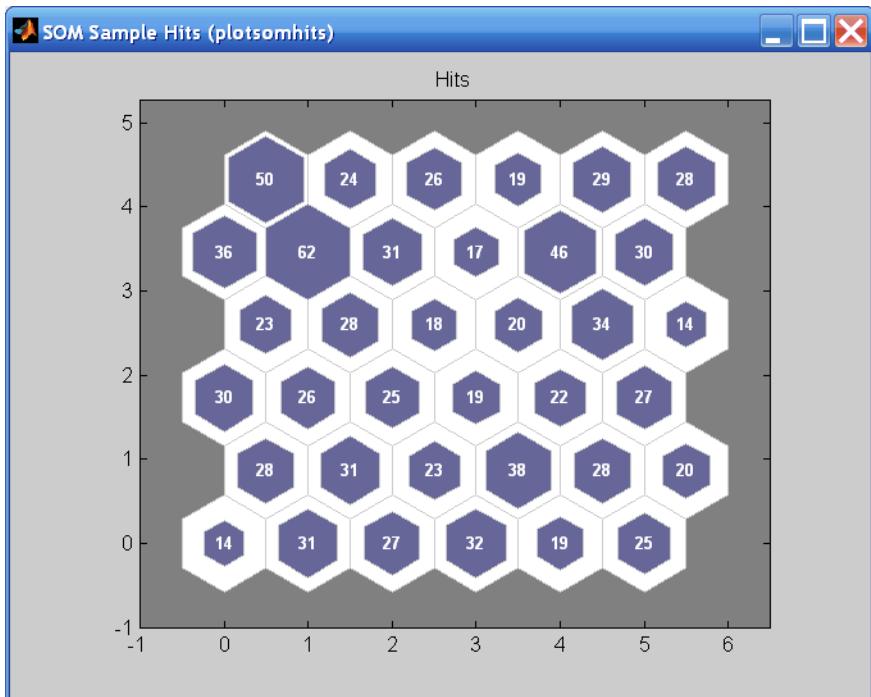
该图窗使用以下颜色编码：

- 蓝色六边形表示神经元。
- 红线连接相邻的神经元。
- 包含红线的区域中的颜色指示神经元之间的距离。
- 颜色越深，表示距离越大。
- 颜色越浅，表示距离越小。

一组浅色线段出现在左上角区域，由一些颜色较暗的线段界定。这种组合方式表明网络已将数据聚类为两个组。这两组可以在前面的权重位置图窗中看到。该图窗的右下角区域包含一小组紧密聚类的数据点。在此区域中，对应的权重彼此更靠近，这在邻点距离图窗中由更浅的颜色表示。当这个小区域中的权重连接到更大的区域时，距离更大，如邻点距离图窗中颜色较深的条带所示。邻点距离图窗右下角区域的线段比左上角区域的线段颜色更深。这种颜色差异表明该区域中的数据点相距较远。此距离在权重位置图中得到确认。

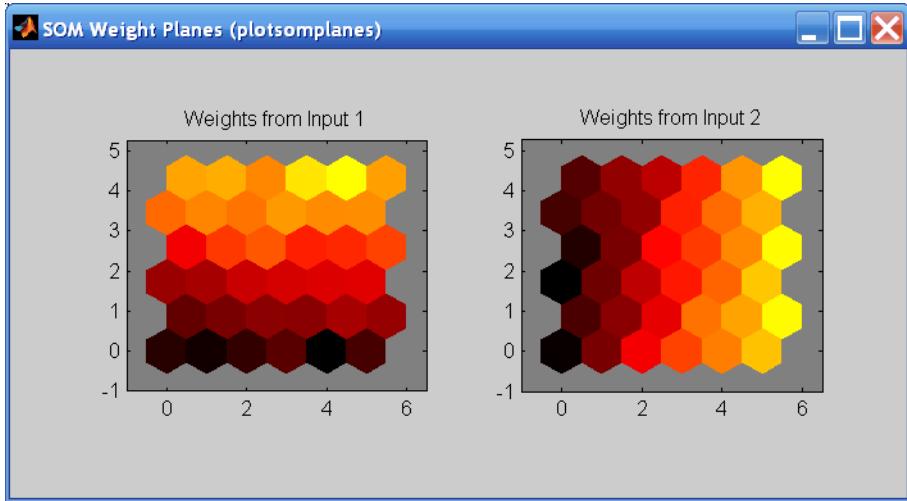


另一个有用的图窗可以显示每个神经元关联多少个数据点。点击 **SOM Sample Hits** 可查看以下图窗。最佳情形是数据在神经元之间分布相当均匀。在此示例中，数据在左上神经元中稍微密集一些，但总体分布相当均匀。



您也可以使用权重平面图窗来可视化权重本身。在训练窗口中点击 **SOM Weight Planes** 以获得下一个图窗。输入向量的每个元素有一个对应的权重平面（在本例中为两个）。它们是将每个输入连接到每个神经

元的权重的可视化表示。（较浅和较深的颜色分别表示较大和较小的权重。）如果两个输入的连接模式非常相似，则可以假设这两个输入高度相关。在这种情况下，输入 1 的连接与输入 2 的连接非常不同。



您还可以从命令行生成前面的所有图窗。尝试下列绘图命令：**plotsomhits**、**plotsomnc**、**plotsomnd**、**plotsomplanes**、**plotsompos** 和 **plotsomtop**。



# 自适应滤波器和自适应训练

---



# 高级主题

---

- “使用并行和 GPU 计算的浅层神经网络” (第 27-2 页)
- “优化神经网络训练速度和内存” (第 27-9 页)
- “选择多层神经网络训练函数” (第 27-13 页)
- “提高浅层神经网络泛化能力，避免过拟合” (第 27-24 页)

## 使用并行和 GPU 计算的浅层神经网络

### 本节内容

- “并行机制模式”（第 27-2 页）
- “分布式计算”（第 27-2 页）
- “单 GPU 计算”（第 27-4 页）
- “分布式 GPU 计算”（第 27-6 页）
- “并行时序”（第 27-7 页）
- “并行可用性、回退和反馈”（第 27-7 页）

**注意** 对于深度学习，自动支持并行和 GPU 计算。您可以使用 `trainNetwork` 函数来训练卷积神经网络 (CNN、ConvNet) 或长短期记忆网络 (LSTM 或 BiLSTM 网络)，并使用 `trainingOptions` 选择执行环境 (CPU、GPU、多 GPU 和并行)。

并行训练或在 GPU 上的训练需要 Parallel Computing Toolbox。有关使用 GPU 和以并行方式进行深度学习的详细信息，请参阅“在并行运行的 CPU、GPU 上和云上使用大数据进行深度学习”（第 1-6 页）。

### 并行机制模式

神经网络本质上是并行算法。多核 CPU、图形处理单元 (GPU) 以及具有多个 CPU 和 GPU 的计算机群集都可以利用这种并行机制。

Parallel Computing Toolbox 在与 Deep Learning Toolbox 结合使用时支持神经网络训练和仿真利用各种并行机制模式。

例如，以下是标准单线程训练和仿真会话：

```
[x, t] = bodyfat_dataset;
net1 = feedforwardnet(10);
net2 = train(net1, x, t);
y = net2(x);
```

您可以在此会话中并行执行的两个步骤是：调用 `train` 和隐式调用 `sim`（其中网络 `net2` 作为函数被调用）。

在 Deep Learning Toolbox 中，您可以将任何数据（如前面示例代码中的 `x` 和 `t`）划分到各采样中。如果 `x` 和 `t` 各只包含一个样本，则不存在并行关系。但是，如果 `x` 和 `t` 包含成百上千或成千上万个样本，则并行机制既可以提高速度，又可以求解更大的问题。

### 分布式计算

通过使用 MATLAB Parallel Server™，Parallel Computing Toolbox 支持神经网络训练和仿真在单台 PC 上的多个 CPU 内核上运行，或在一个网络的多台计算机的多个 CPU 上运行。

使用多个内核可以加快计算速度。如果求解问题所使用的数据集太大而无法放入一台计算机的内存中，则可以使用多台计算机来求解问题。问题大小仅受限于所有计算机上可用的内存总量。

要管理群集配置，请通过 MATLAB 主页选项卡上的环境菜单中的并行 > Manage Cluster Profiles 使用 Cluster Profile Manager。

要使用默认群集配置文件（通常是本地 CPU 内核）打开 MATLAB 工作进程池，请使用以下命令：

```
pool = parpool
```

```
Starting parallel pool (parpool) using the 'local' profile ... connected to 4 workers.
```

当 **parpool** 运行时，它会显示池中可用的工作进程数量。确定工作进程数量的另一种方法是查询池：

```
pool.NumWorkers
```

```
4
```

现在您可以在所有工作进程中按样本拆分数据的情况下训练和仿真神经网络。为此，请将 **train** 和 **sim** 参数 '**useParallel**' 设置为 '**yes**'。

```
net2 = train(net1,x,t,'useParallel','yes')
y = net2(x,'useParallel','yes')
```

使用 '**showResources**' 参数来验证计算是否跨多个工作进程运行。

```
net2 = train(net1,x,t,'useParallel','yes','showResources','yes');
y = net2(x,'useParallel','yes','showResources','yes');
```

MATLAB 指示使用了哪些资源。例如：

```
Computing Resources:
Parallel Workers
Worker 1 on MyComputer, MEX on PCWIN64
Worker 2 on MyComputer, MEX on PCWIN64
Worker 3 on MyComputer, MEX on PCWIN64
Worker 4 on MyComputer, MEX on PCWIN64
```

当调用 **train** 和 **sim** 时，它们会在训练和仿真之前将输入矩阵或元胞数组数据划分为分布式合成值。当 **sim** 计算出合成值时，会先将此输出转换回相同的矩阵或元胞数组形式，再将其返回。

但是，如果出现以下情况，您可能需要手动执行此数据划分：

- 问题大小对主机来说太大。通过按顺序手动定义合成值的元素，可定义更大的问题。
- 众所周知，有些工作进程所在的计算机比其他计算机速度更快或内存更多。在分配数据时，您可以使每个工作进程处理不同数量的样本。这称为负载平衡。

以下代码按顺序创建一系列随机数据集，并将它们保存到各个单独文件中：

```
pool = gcp;
for i=1:pool.NumWorkers
    x = rand(2,1000);
    save(['inputs' num2str(i)],'x');
    t = x(1,:).*x(2,:)+2*(x(1,:)+x(2,:));
    save(['targets' num2str(i)],'t');
    clear x t
end
```

由于数据是按顺序定义的，因此您定义的总数据集可以大于可放入主机内存中的数据量。计算机内存一次只能容纳该数据集的一部分。

现在，您可以按顺序在多个并行工作进程上加载数据集，并基于合成数据训练和仿真网络。当使用合成数据调用 **train** 或 **sim** 时，'**useParallel**' 参数会自动设置为 '**yes**'。在使用合成数据时，请在训练前使用 **configure** 函数手动配置网络的输入和输出，以匹配数据集之一。

```

xc = Composite;
tc = Composite;
for i=1:pool.NumWorkers
    data = load(['inputs' num2str(i)],'x');
    xc{i} = data.x;
    data = load(['targets' num2str(i)],'t');
    tc{i} = data.t;
    clear data
end
net2 = configure(net1,xc{1},tc{1});
net2 = train(net2,xc,tc);
yc = net2(xc);

```

要转换 `sim` 返回的合成输出，您可以访问它的每个元素，如果担心内存限制，可以单独访问。

```

for i=1:pool.NumWorkers
    yi = yc{i}
end

```

如果您不担心内存限制，请将合成值合并为一个本地值。

```
y = {yc{:}};
```

当进行负载平衡时，会发生相同的过程，但不是每个数据集都有相同数量的样本（在前面的示例中为 1000 个），而是可以调整样本数以最好地利用工作进程主机的内存和速度差异。

不要求每个工作进程都有数据。如果合成值的元素 `i` 未定义，则计算中将不使用工作进程 `i`。

## 单 GPU 计算

随着每代新产品的发展，GPU 卡的内核数量、内存大小和速度效率都在快速增长。视频游戏长期受益于改进的 GPU 性能。现在，这些卡足够灵活，可以执行一般的数值计算任务，例如训练神经网络。

有关最新 GPU 要求的信息，请参阅 Parallel Computing Toolbox 的网页；或者查询 MATLAB 以确定您的 PC 是否有支持的 GPU。此函数返回系统中 GPU 的数量：

```

count = gpuDeviceCount
count =
1

```

如果结果是一个或多个，您可以通过索引查询每个 GPU 的特征。这包括它的名称、多处理器的数量、每个多处理器的 `SIMDWidth` 以及总内存。

```

gpu1 = gpuDevice(1)
gpu1 =

```

CUDADevice with properties:

```

Name: 'GeForce GTX 470'
Index: 1
ComputeCapability: '2.0'
SupportsDouble: 1
DriverVersion: 4.1000
MaxThreadsPerBlock: 1024

```

```

MaxShmemPerBlock: 49152
MaxThreadBlockSize: [1024 1024 64]
    MaxGridSize: [65535 65535 1]
        SIMDWidth: 32
    TotalMemory: 1.3422e+09
AvailableMemory: 1.1056e+09
MultiprocessorCount: 14
ClockRateKHz: 1215000
ComputeMode: 'Default'
GPUOverlapsTransfers: 1
KernelExecutionTimeout: 1
    CanMapHostMemory: 1
DeviceSupported: 1
    DeviceSelected: 1

```

利用 GPU 的最简单方法是指定在参数 'useGPU' 设置为 'yes' ('no' 为默认值) 的情况下调用 **train** 和 **sim**。

```

net2 = train(net1,x,t,'useGPU','yes')
y = net2(x,'useGPU','yes')

```

如果 **net1** 具有默认的训练函数 **trainlm**, 您会看到一条警告, 指出 GPU 计算不支持 Jacobian 矩阵训练, 仅支持梯度训练。因此, 训练函数自动更改为梯度训练函数 **trainscg**。为避免出现该通知, 您可以在训练前指定该函数:

```
net1.trainFcn = 'trainscg';
```

要验证训练和仿真是否在 GPU 设备上进行, 请要求显示计算机资源:

```

net2 = train(net1,x,t,'useGPU','yes','showResources','yes')
y = net2(x,'useGPU','yes','showResources','yes')

```

以上每行代码都输出以下资源摘要:

```

Computing Resources:
GPU device #1, GeForce GTX 470

```

当任一输入参数是 **gpuArray** 时, 许多 MATLAB 函数会自动在 GPU 上执行。通常情况下, 您可以使用 **gpuArray** 和 **gather** 函数在 GPU 之间来回移动数组。然而, 为了在 GPU 上高效进行神经网络计算, 需要转置矩阵和填充列以使每列中的第一个元素在 GPU 内存中正确对齐。Deep Learning Toolbox 提供一个名为 **nndata2gpu** 的特殊函数, 它可将数组移至 GPU 上并对其进行适当的组织:

```

xg = nndata2gpu(x);
tg = nndata2gpu(t);

```

现在, 您可以使用已在 GPU 上转换的数据来训练和仿真网络, 而不必指定 'useGPU' 参数。然后使用补充函数 **gpu2nndata** 转换生成的 GPU 数组并将其返回给 MATLAB。

在使用 **gpuArray** 数据进行训练之前, 必须使用 **configure** 函数, 用常规的 MATLAB 矩阵手动配置网络的输入和输出:

```

net2 = configure(net1,x,t); % Configure with MATLAB arrays
net2 = train(net2,xg,tg); % Execute on GPU with NNET formatted gpuArrays
yg = net2(xg); % Execute on GPU
y = gpu2nndata(yg); % Transfer array to local workspace

```

在 GPU 和其他可能需要部署神经网络的硬件上, 通常情况下指数函数 **exp** 不是用硬件实现的, 而是用软件库实现的。这可能减慢使用 **tansig** **sigmoid** 传递函数的神经网络的运行速度。另外还有 **Elliot sigmoid** 函数, 其表达式不包括对任何更高阶函数的调用:

(equation)  $a = n / (1 + \text{abs}(n))$

在训练前，网络的 tansig 层可转换为 elliotSig 层，如下所示：

```
for i=1:net.numLayers
    if strcmp(net.layers{i}.transferFcn,'tansig')
        net.layers{i}.transferFcn = 'elliotSig';
    end
end
```

现在，训练和仿真可能在 GPU 上更快，部署硬件更简单。

## 分布式 GPU 计算

分布式计算和 GPU 计算可以结合使用，以在单台计算机上的多个 CPU 和/或 GPU 上运行计算，或使用 MATLAB Parallel Server 在群集上的多个 CPU 和/或 GPU 上运行计算。

要使用这种计算方式，最简单的方法是通过由您使用的群集配置文件确定的并行池，指定 `train` 和 `sim` 来执行此操作。在这种情况下，特别推荐使用 '`showResources`' 选项，以验证是否采用预期的硬件：

```
net2 = train(net1,x,t,'useParallel','yes','useGPU','yes','showResources','yes')
y = net2(x,'useParallel','yes','useGPU','yes','showResources','yes')
```

这些代码行使用并行池中所有可用的工作进程。每个独占 GPU 都有一个对应的工作进程使用此 GPU，而其他工作进程作为 CPU 工作。在某些情况下，只使用 GPU 可能会更快。例如，如果一台计算机有三个 GPU，每个 GPU 有四个工作进程，由 GPU 加速的三个工作进程可能会被第四个 CPU 工作进程限制速度。在这些情况下，您可以指定 `train` 和 `sim` 仅使用具有独占 GPU 的工作进程。

```
net2 = train(net1,x,t,'useParallel','yes','useGPU','only','showResources','yes')
y = net2(x,'useParallel','yes','useGPU','only','showResources','yes')
```

与简单的分布式计算一样，分布式 GPU 计算可以受益于手动创建的合成值。通过自己定义合成值，您可以指示要使用哪些工作进程、为每个工作进程分配多少样本以及哪些工作进程使用 GPU。

例如，如果您有四个工作进程，而只有三个 GPU，则您可以为 GPU 工作进程定义更大的数据集。此处，对每个合成元素使用不同样本加载来创建一个随机数据集：

```
numSamples = [1000 1000 1000 300];
xc = Composite;
tc = Composite;
for i=1:4
    xi = rand(2,numSamples(i));
    ti = xi(1,:).^2 + 3*xi(2,:);
    xc{i} = xi;
    tc{i} = ti;
end
```

您现在可以指定 `train` 和 `sim` 使用三个可用的 GPU：

```
net2 = configure(net1,xc{1},tc{1});
net2 = train(net2,xc,tc,'useGPU','yes','showResources','yes');
yc = net2(xc,'showResources','yes');
```

为了确保前三个工作进程使用 GPU，手动将每个工作进程的合成元素转换为 `gpuArray`。每个工作进程在并行执行的 `spmd` 模块中执行此变换。

```
spmd
if labindex <= 3
```

```

xc = nnData2GPU(xc);
tc = nnData2GPU(tc);
end
end

```

现在数据指定何时使用 GPU，因此您不需要告诉 `train` 和 `sim` 这样做。

```

net2 = configure(net1, xc{1}, tc{1});
net2 = train(net2, xc, tc, 'showResources', 'yes');
yc = net2(xc, 'showResources', 'yes');

```

确保每个 GPU 只由一个工作进程使用，这样计算效率最高。如果多个工作进程将 `gpuArray` 数据分配到同一个 GPU 上，计算仍然会工作，但速度会慢一些，因为 GPU 将按顺序处理多个工作进程的数据。

## 并行时序

对于时序网络，只对 `x` 和 `t` 使用元胞数组值，并根据需要可选地包括初始输入延迟状态 `xi` 和初始层延迟状态 `ai`。

```

net2 = train(net1, x, t, xi, ai, 'useGPU', 'yes')
y = net2(x, xi, ai, 'useParallel', 'yes', 'useGPU', 'yes')

net2 = train(net1, x, t, xi, ai, 'useParallel', 'yes')
y = net2(x, xi, ai, 'useParallel', 'yes', 'useGPU', 'only')

net2 = train(net1, x, t, xi, ai, 'useParallel', 'yes', 'useGPU', 'only')
y = net2(x, xi, ai, 'useParallel', 'yes', 'useGPU', 'only')

```

请注意，并行发生在样本之间，或在时序跨不同序列的情况下发生。然而，如果网络只有输入延迟，没有层延迟，则延迟的输入可以预先计算，以便在计算时，时间步变为不同样本并且可以并行化。

`timedelaynet` 以及 `narxnet` 和 `narnet` 的开环版本等网络即属这种情况。如果网络有层延迟，则为了计算目的，时间就不能“扁平化”，因此单个序列数据无法并行化。`layrecnet` 以及 `narxnet` 和 `narnet` 的闭环版本等网络即属这种情况。但是，如果数据由多个序列组成，则可以在不同序列之间并行化。

## 并行可用性、回退和反馈

如前所述，您可以查询 MATLAB 以发现当前可用的并行资源。

要查看主机上有哪些 GPU 可用，请执行以下代码：

```

gpuCount = gpuDeviceCount
for i=1:gpuCount
    gpuDevice(i)
end

```

要查看当前并行池中运行的工作进程数量，请执行以下代码：

```
poolSize = pool.NumWorkers
```

要查看在使用 MATLAB Parallel Server 的 PC 群集上运行的并行池中可用的 GPU，请执行以下代码：

```

spmd
    worker.index = labIndex;
    worker.name = system('hostname');
    worker.gpuCount = gpuDeviceCount;
try

```

```
worker.gpuInfo = gpuDevice;
catch
    worker.gpuInfo = [];
end
worker
end
```

当 'useParallel' 或 'useGPU' 设置为 'yes' 但并行或 GPU 工作进程不可用时，遵循如下约定：当请求资源时，如果资源可用，就会使用它们。执行的计算不带误差，即使实际上带有误差。从请求的资源回退到实际资源的过程如下：

- 如果 'useParallel' 为 'yes' 但 Parallel Computing Toolbox 不可用，或并行池未打开，则计算返回到单线程 MATLAB。
- 如果 'useGPU' 为 'yes' 但当前 MATLAB 会话的 gpuDevice 未分配或不受支持，则计算将返回到 CPU 上进行。
- 如果 'useParallel' 和 'useGPU' 为 'yes'，则每个拥有独占 GPU 的工作进程都使用该 GPU，而其他工作进程返回到 CPU 上。
- 如果 'useParallel' 为 'yes' 且 'useGPU' 为 'only'，则使用具有独占 GPU 的工作进程。不使用其他工作进程，除非任何工作进程都没有 GPU。在没有 CPU 的情况下，所有工作进程都使用 CPU。

当不确定实际使用的是什么硬件时，请检查 gpuDeviceCount、gpuDevice 和 pool.NumWorkers 以确保所需的硬件可用，并调用 train 和 sim 且将 'showResources' 设置为 'yes' 以验证实际使用了哪些资源。

# 优化神经网络训练速度和内存

## 本节内容

- “内存减少” (第 27-9 页)
- “快速 Elliot sigmoid” (第 27-9 页)

## 内存减少

根据特定神经网络，仿真和梯度计算可以在 MATLAB 或 MEX 中进行。MEX 的内存效率更高，但 MATLAB 可以用时间来换取更高的内存效率。

要确定使用的是 MATLAB 还是 MEX，请使用 'showResources' 选项，如以下一般语法所示：

```
net2 = train(net1,x,t,'showResources','yes')
```

如果正在使用 MATLAB 并且存在内存限制，则可对 N 个数据子集中的每个子集依次执行 N 次计算，以换取将所需的临时存储量减少至原来的  $1/N$ 。

```
net2 = train(net1,x,t,'reduction',N);
```

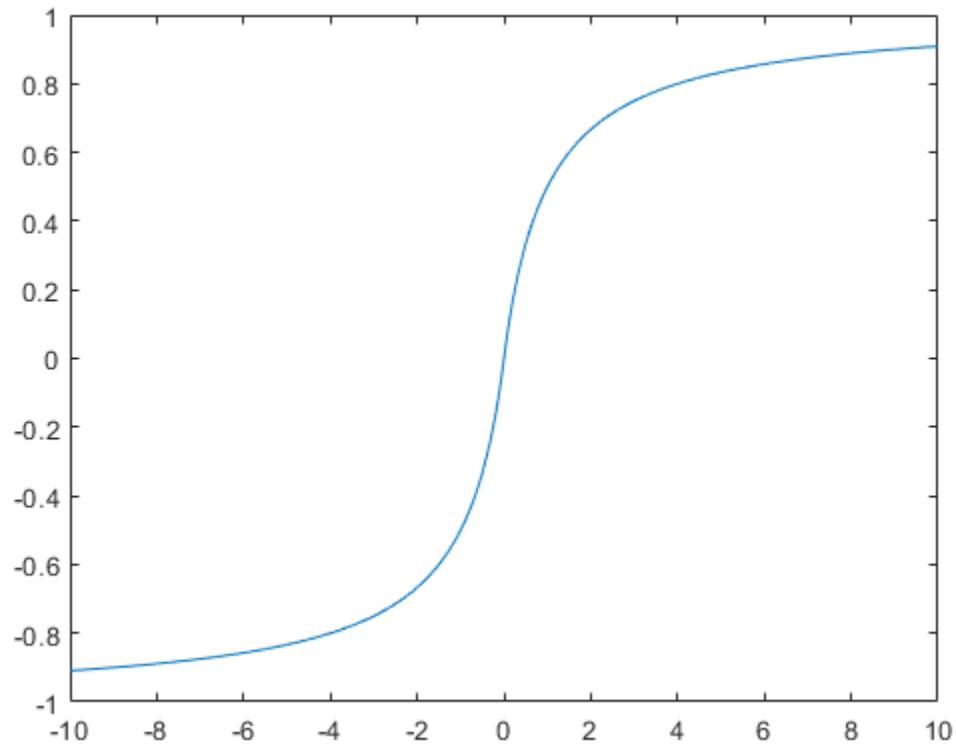
这称为降内存。

## 快速 Elliot sigmoid

一些简单的计算硬件可能不直接支持指数函数，而软件实现可能很慢。Elliot sigmoid `elliotsig` 函数的作用与对称 sigmoid `tansig` 函数相同，但避免了指数函数。

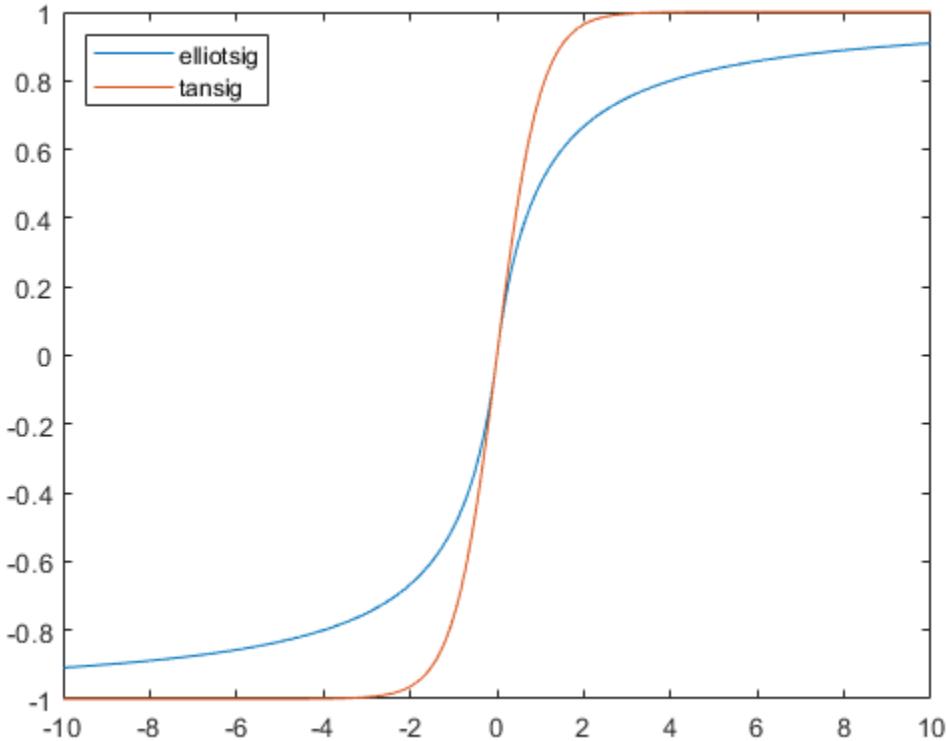
以下是 Elliot sigmoid 的绘图：

```
n = -10:0.01:10;
a = elliotsig(n);
plot(n,a)
```



接下来，将 elliotsig 与 tansig 进行比较。

```
a2 = tansig(n);
h = plot(n,a,n,a2);
legend(h,'elliotsig','tansig','Location','NorthWest')
```



要使用 `elliotsig` 而不是 `tansig` 来训练神经网络，请变换网络的传递函数：

```
[x,t] = bodyfat_dataset;
net = feedforwardnet;
view(net)
net.layers{1}.transferFcn = 'elliotsig';
view(net)
net = train(net,x,t);
y = net(x)
```

此处比较 `elliotsig` 和 `tansig` 的执行时间。`elliotsig` 在测试系统上大约快四倍。

```
n = rand(5000,5000);
tic,for i=1:100,a=tansig(n); end, tansigTime = toc;
tic,for i=1:100,a=elliotsig(n); end, elliotTime = toc;
speedup = tansigTime / elliotTime

speedup =
```

4.1406

然而，虽然使用 `elliotsig` 进行仿真更快，但由于两个传递函数的形状不同，训练速度不一定会更快。此处，为 `tansig` 和 `elliotsig` 分别训练 10 个网络，但即使使用同一网络基于同一问题进行训练，训练时间也有很大差异。

```
[x,t] = bodyfat_dataset;
tansigNet = feedforwardnet;
tansigNet.trainParam.showWindow = false;
```

```
elliotNet = tansigNet;
elliotNet.layers{1}.transferFcn = 'elliotsig';
for i=1:10, tic, net = train(tansigNet,x,t); tansigTime = toc, end
for i=1:10, tic, net = train(elliotNet,x,t), elliotTime = toc, end
```

## 选择多层神经网络训练函数

本节内容
“SIN 数据集” (第 27-13 页)
“PARITY 数据集” (第 27-15 页)
“ENGINE 数据集” (第 27-16 页)
“CANCER 数据集” (第 27-18 页)
“CHOLESTEROL 数据集” (第 27-19 页)
“DIABETES 数据集” (第 27-21 页)
“总结” (第 27-22 页)

对于给定的问题，很难知道哪种训练算法最快。它取决于许多因素，包括问题的复杂性、训练集中数据点的数量、网络中权重和偏置的数量、误差目标，以及网络用于模式识别（判别分析）还是函数逼近（回归）。本节比较各种训练算法。前馈网络针对六个不同问题进行训练。其中三个问题属于模式识别类别，另外三个问题属于函数逼近类别。其中两个问题是简单的“玩具”问题，而另外四个是“现实世界”问题。使用具有各种不同架构和复杂性的网络，并将网络训练到各种不同准确度水平。

下表列出了用于测试的算法及其缩写。

缩写	算法	说明
LM	<b>trainlm</b>	Levenberg-Marquardt
BFG	<b>trainbfg</b>	BFGS 拟牛顿
RP	<b>trainrp</b>	弹性反向传播
SCG	<b>trainscg</b>	量化共轭梯度
CGB	<b>traincgb</b>	带 Powell/Beale 重启的共轭梯度
CGF	<b>traincfg</b>	Fletcher-Powell 共轭梯度
CGP	<b>traincgp</b>	Polak-Ribiére 共轭梯度
OSS	<b>trainoss</b>	单步正割
GDX	<b>traingdx</b>	可变学习率反向传播

下表列出了六个基准问题以及所使用的网络的一些特征、训练过程和计算机。

问题标题	问题类型	网络结构	误差目标	计算机
SIN	函数逼近	1-5-1	0.002	Sun Sparc 2
PARITY	模式识别	3-10-10-1	0.001	Sun Sparc 2
ENGINE	函数逼近	2-30-2	0.005	Sun Enterprise 4000
CANCER	模式识别	9-5-5-2	0.012	Sun Sparc 2
CHOLESTEROL	函数逼近	21-15-3	0.027	Sun Sparc 20
DIABETES	模式识别	8-15-15-2	0.05	Sun Sparc 20

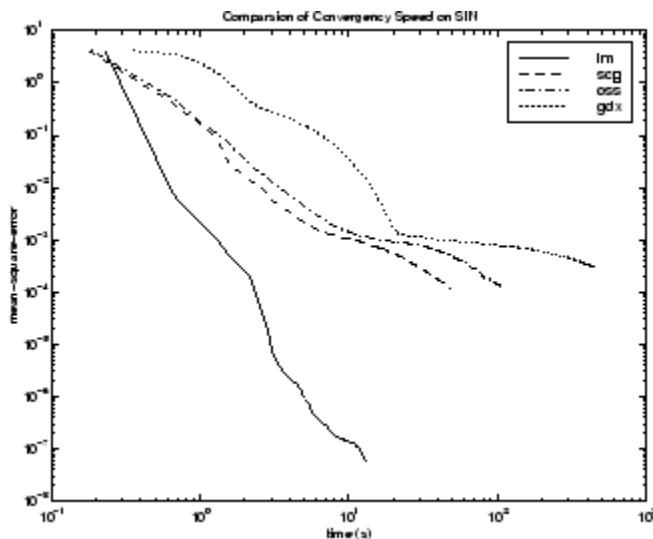
### SIN 数据集

第一个基准数据集是简单的函数逼近问题。一个 1-5-1 网络（在隐含层具有 `tansig` 传递函数，在输出层具有线性传递函数）用于逼近正弦波的单个周期。下表摘要显示了使用九种不同训练算法训练网络的结果。

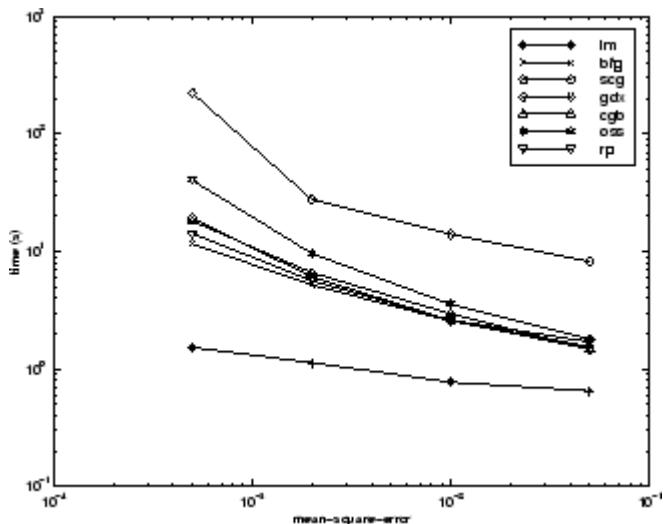
果。表中的每个条目表示 30 次不同尝试，每次尝试使用不同的随机初始权重。在每种情况下，网络会持续训练，直到平方误差小于 0.002。对于此问题，最快的算法是 Levenberg-Marquardt 算法。平均来说，它的速度是仅次于它的算法的四倍以上。这是 LM 算法最适合的问题类型 - 函数逼近问题，其中网络的权重数少于 100 个，逼近必须非常准确。

算法	均值时间 (秒)	比率	最长时间 (秒)	最大时间 (秒)	标准 (秒)
LM	1.14	1.00	0.65	1.83	0.38
BFG	5.22	4.58	3.17	14.38	2.08
RP	5.67	4.97	2.66	17.24	3.72
SCG	6.09	5.34	3.18	23.64	3.81
CGB	6.61	5.80	2.99	23.65	3.67
CGF	7.86	6.89	3.57	31.23	4.76
CGP	8.24	7.23	4.07	32.32	5.03
OSS	9.64	8.46	3.97	59.63	9.79
GDX	27.69	24.29	17.21	258.15	43.65

各种算法的性能可能会受到逼近准确度要求的影响。下图显示了这一点，它是几种代表性算法的均方误差对执行时间（30 次尝试取平均值）的图。在此处，您可以看到，随着时间的推移，LM 算法的误差比所示的其他算法减少得更快。



下图进一步说明各算法之间的关系，它是收敛所需的时间对均方误差收敛目标的图。此处您可以看到，随着误差目标的减小，LM 算法提供的改进变得更加显著。一些算法随着误差目标的降低而表现得更好 (LM 和 BFG)，而其他算法随着误差目标的降低而表现得更差 (OSS 和 GDX)。

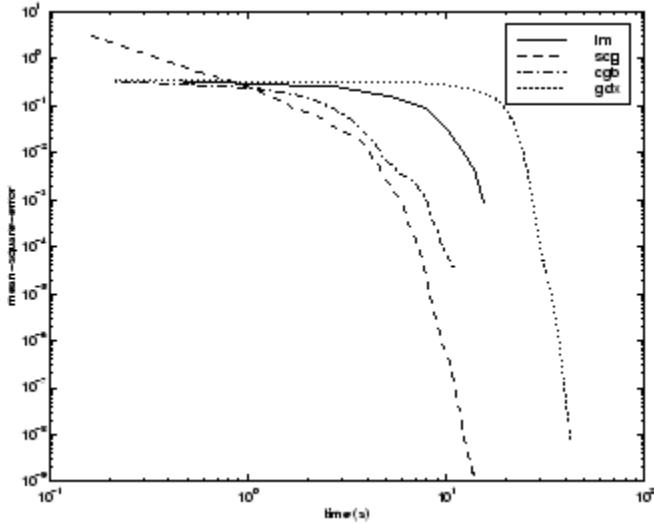


## PARITY 数据集

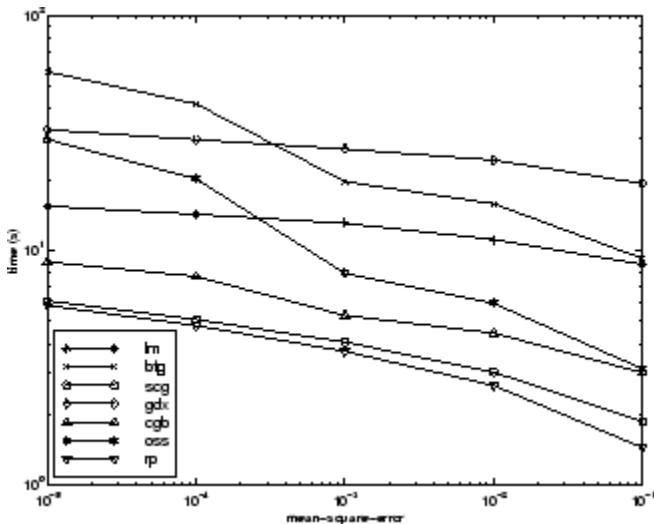
第二个基准测试问题是简单的模式识别问题 - 检测一个 3 位数的奇偶性。如果输入模式中的 1 的数量是奇数，则网络应输出 1；否则，它应输出 -1。用于求解此问题的网络是一个 3-10-10-1 网络，它的每个层中都有 tansig 神经元。下表摘要显示了用九种不同算法训练该网络的结果。表中的每个条目表示 30 次不同尝试，每次尝试使用不同的随机初始权重。在每种情况下，网络会持续训练，直到平方误差小于 0.001。对于此问题，最快的算法是弹性反向传播算法，尽管共轭梯度算法（特别是量化共轭梯度算法）几乎同样快。请注意，LM 算法在此问题上表现不佳。一般来说，LM 算法在模式识别问题上的表现不如在函数逼近问题上的表现好。LM 算法是为近似线性的最小二乘问题设计的。由于模式识别问题中的输出神经元通常是饱和的，因此您不会在线性区域中操作。

算法	均值时间 (秒)	比率	最长时间 (秒)	最大时间 (秒)	标准 (秒)
RP	3.73	1.00	2.35	6.89	1.26
SCG	4.09	1.10	2.36	7.48	1.56
CGP	5.13	1.38	3.50	8.73	1.05
CGB	5.30	1.42	3.91	11.59	1.35
CGF	6.62	1.77	3.96	28.05	4.32
OSS	8.00	2.14	5.06	14.41	1.92
LM	13.07	3.50	6.48	23.78	4.96
BFG	19.68	5.28	14.19	26.64	2.85
GDX	27.07	7.26	25.21	28.52	0.86

与函数逼近问题一样，各种算法的性能会受到网络所需准确度的影响。下图显示了这一点，它是一些典型算法的均方误差对执行时间的图。LM 算法在某一点后会快速收敛，但收敛的时间晚于其他算法。



下图进一步说明各算法之间的关系，它是收敛所需的时间对均方误差收敛目标的图。您可以再次看到，随着误差目标的减小，一些算法的性能下降（OSS 和 BFG）。



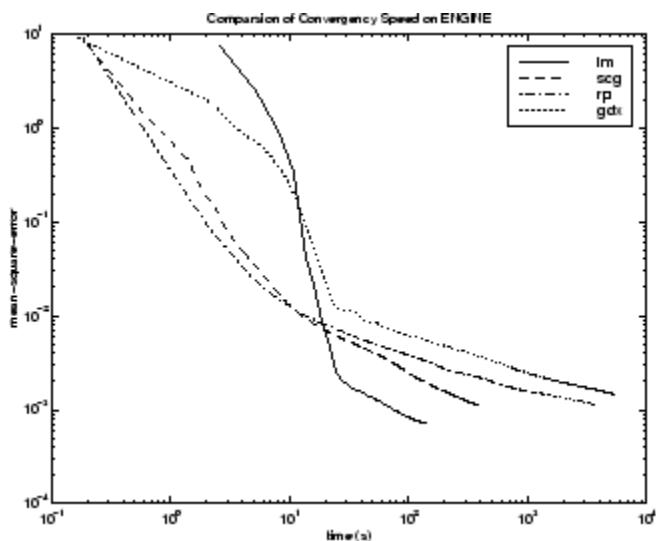
## ENGINE 数据集

第三个基准问题是一个现实的函数逼近（或非线性回归）问题。这些数据是从某发动机运行中获得的。网络的输入是发动机转速和供油水平，网络的输出是扭矩和排放水平。用于求解此问题的网络是一个 2-30-2 网络，它的隐藏层中有 tansig 神经元，输出层中有线性神经元。下表摘要显示了用九种不同算法训练该网络的结果。表中的每个条目表示 30 次不同尝试（由于时间限制，RP 和 GDX 各有 10 次尝试），每次尝试使用不同的随机初始权重。在每种情况下，网络会持续训练，直到平方误差小于 0.005。对于此问题，最快的算法是 LM 算法，其次是 BFGS 拟牛顿算法和共轭梯度算法。尽管这是函数逼近问题，但 LM 算法并不像在 SIN 数据集上那样优势明显。在本例中，网络中权重和偏置的数量远远大于在 SIN 问题中使用的权重和偏置的数量（152 对 16），并且 LM 算法的优势随着网络参数个数的增加而减弱。

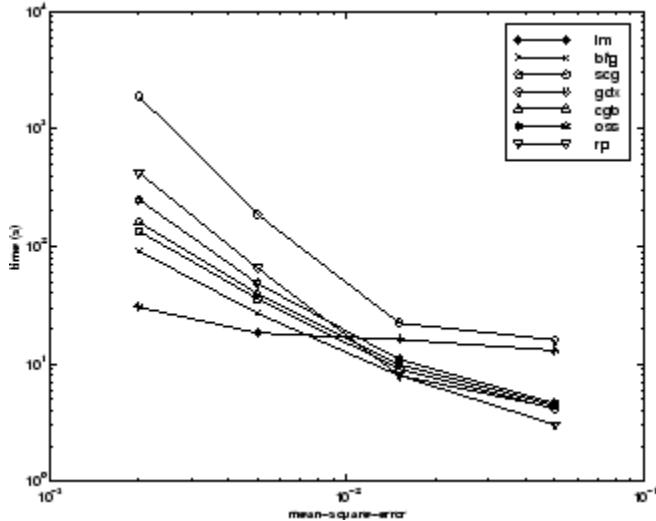
算法	均值时间 (秒)	比率	最小时间 (秒)	最大时间 (秒)	标准 (秒)
LM	18.45	1.00	12.01	30.03	4.27

算法	均值时间 (秒)	比率	最小时间 (秒)	最大时间 (秒)	标准 (秒)
BFG	27.12	1.47	16.42	47.36	5.95
SCG	36.02	1.95	19.39	52.45	7.78
CGF	37.93	2.06	18.89	50.34	6.12
CGB	39.93	2.16	23.33	55.42	7.50
CGP	44.30	2.40	24.99	71.55	9.89
OSS	48.71	2.64	23.51	80.90	12.33
RP	65.91	3.57	31.83	134.31	34.24
GDX	188.50	10.22	81.59	279.90	66.67

以下是一些典型算法的均方误差对执行时间的图。随着时间的推移，LM 算法的性能相对于其他算法逐渐改善。



下图进一步说明各算法之间的关系，它是收敛所需的时间对均方误差收敛目标的图。您可以再次看到，随着误差目标的减小，一些算法的性能下降（GDX 和 RP），而 LM 算法却不断改善。



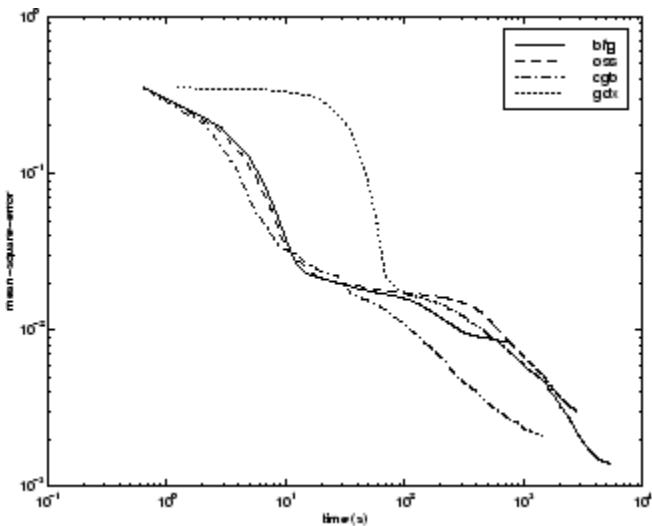
## CANCER 数据集

第四个基准问题是一个现实的模式识别（或非线性判别分析）问题。网络的目标是根据显微镜检查收集的细胞特征将肿瘤分为良性或恶性。输入属性包括团块厚度、细胞大小和细胞形状的均匀性、边粘附量和裸核的频率。这些数据来自麦迪逊威斯康星大学医院的威廉·沃尔伯格博士。用于求解此问题的网络是一个 9-5-5-2 网络，它的所有层中都有 tansig 神经元。下表摘要显示了用九种不同算法训练该网络的结果。表中的每个条目表示 30 次不同尝试，每次尝试使用不同的随机初始权重。在每种情况下，网络会持续训练，直到平方误差小于 0.012。一些算法的若干次运行未收敛，因此只使用每个算法的前 75% 的运行次数来获得统计数据。

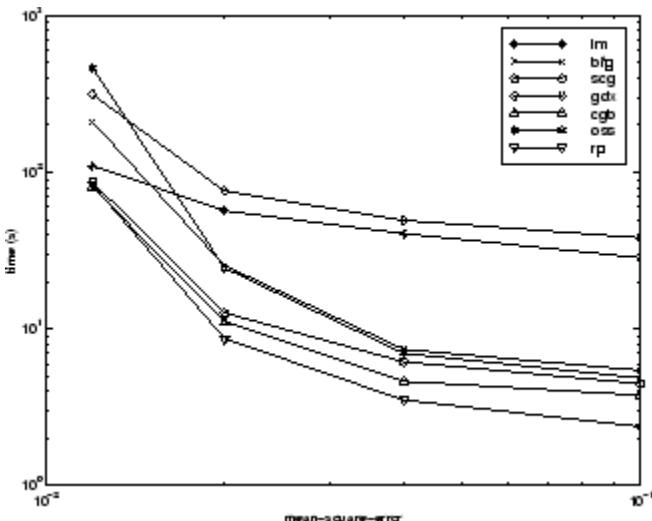
共轭梯度法和弹性反向传播都能快速收敛，而 LM 算法的收敛也相当快。与在 PARITY 数据集上一样，LM 算法在模式识别问题上的表现不如在函数逼近问题上的表现好。

算法	均值时间 (秒)	比率	最小时间 (秒)	最大时间 (秒)	标准 (秒)
CGB	80.27	1.00	55.07	102.31	13.17
RP	83.41	1.04	59.51	109.39	13.44
SCG	86.58	1.08	41.21	112.19	18.25
CGP	87.70	1.09	56.35	116.37	18.03
CGF	110.05	1.37	63.33	171.53	30.13
LM	110.33	1.37	58.94	201.07	38.20
BFG	209.60	2.61	118.92	318.18	58.44
GDX	313.22	3.90	166.48	446.43	75.44
OSS	463.87	5.78	250.62	599.99	97.35

以下是一些典型算法的均方误差对执行时间的图。对于此问题，性能的差异没有在以前的问题中那样表现明显。



下图进一步说明各算法之间的关系，它是收敛所需的时间对均方误差收敛目标的图。您可以再次看到，随着误差目标的减小，一些算法的性能下降（OSS 和 BFG），而 LM 算法却不断改善。在任何问题上，随着误差目标的降低，LM 算法的性能通常都会相对于其他算法逐渐改善。



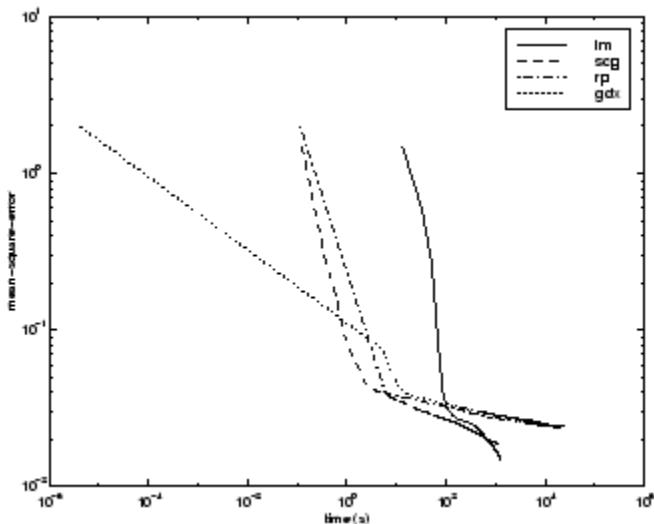
## CHOLESTEROL 数据集

第五个基准问题是一个现实的函数逼近（或非线性回归）问题。网络的目标是根据对 21 种频谱分量的测量来预测胆固醇水平（ldl、hdl 和 vldl）。数据来自俄克拉荷马州立大学化学系的 Neil Purdie 博士 [PuLu92]。用于求解此问题的网络是一个 21-15-3 网络，它的隐藏层中有 tansig 神经元，输出层中有线性神经元。下表摘要显示了用九种不同算法训练该网络的结果。表中的每个条目表示 20 次不同尝试（RP 和 GDX 各有 10 次尝试），每次尝试使用不同的随机初始权重。在每种情况下，网络会持续训练，直到平方误差小于 0.027。

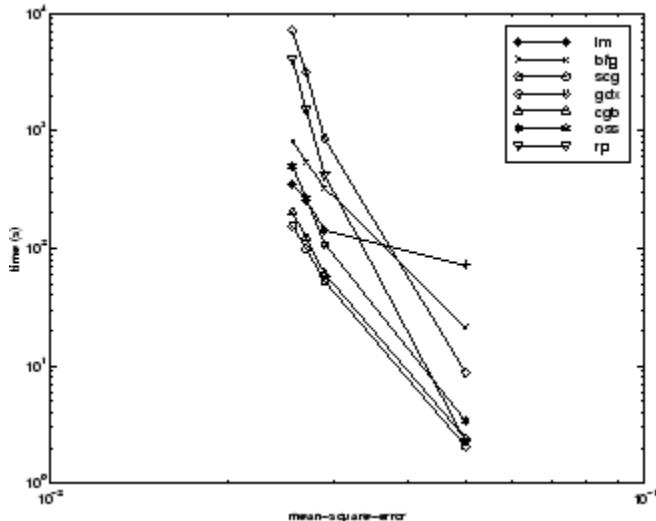
量化共轭梯度算法在此问题上性能最佳，尽管所有共轭梯度算法的性能都很好。LM 算法在此函数逼近问题上的表现不如在另外两个问题上的表现好。这是因为网络中权重和偏置的数量再次增大（378 对 152 对 16）。随着参数个数的增加，LM 算法所需的计算量呈几何增长。

算法	均值时间 (秒)	比率	最长时间 (秒)	最大时间 (秒)	标准 (秒)
SCG	99.73	1.00	83.10	113.40	9.93
CGP	121.54	1.22	101.76	162.49	16.34
CGB	124.06	1.2	107.64	146.90	14.62
CGF	136.04	1.36	106.46	167.28	17.67
LM	261.50	2.62	103.52	398.45	102.06
OSS	268.55	2.69	197.84	372.99	56.79
BFG	550.92	5.52	471.61	676.39	46.59
RP	1519.00	15.23	581.17	2256.10	557.34
GDX	3169.50	31.78	2514.90	4168.20	610.52

以下是一些典型算法的均方误差对执行时间的图。对于此问题，您可以看到，相对于其他算法，LM 算法能够将均方误差降低到更低的水平。SCG 和 RP 算法具有最快的初始收敛。



下图进一步说明各算法之间的关系，它是收敛所需的时间对均方误差收敛目标的图。您可以看到，随着误差目标的降低，相对于其他算法，LM 和 BFG 算法的性能不断改善。



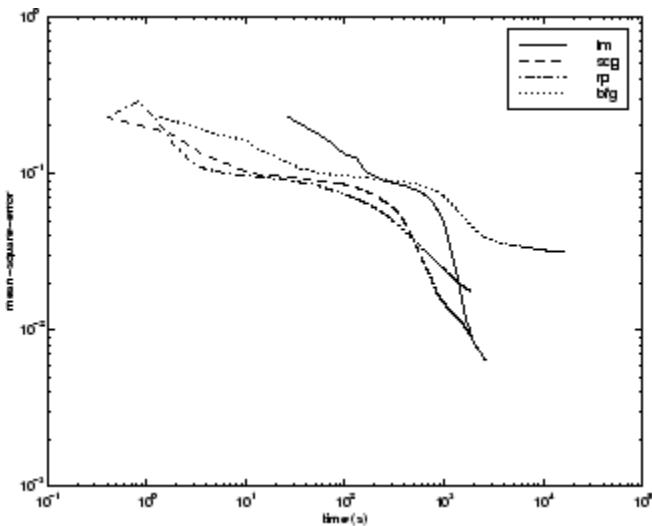
## DIABETES 数据集

第六个基准问题是一个模式识别问题。网络的目标是根据个人数据（年龄、怀孕次数）和医学检查结果（如血压、体重指数、葡萄糖耐量试验结果等）来诊断个人是否患有糖尿病。这些数据来自加州大学欧文分校的机器学习数据库。用于求解此问题的网络是一个 8-15-15-2 网络，它的所有层中都有 tansig 神经元。下表摘要显示了用九种不同算法训练该网络的结果。表中的每个条目表示 10 次不同尝试，每次尝试使用不同的随机初始权重。在每种情况下，网络会持续训练，直到平方误差小于 0.05。

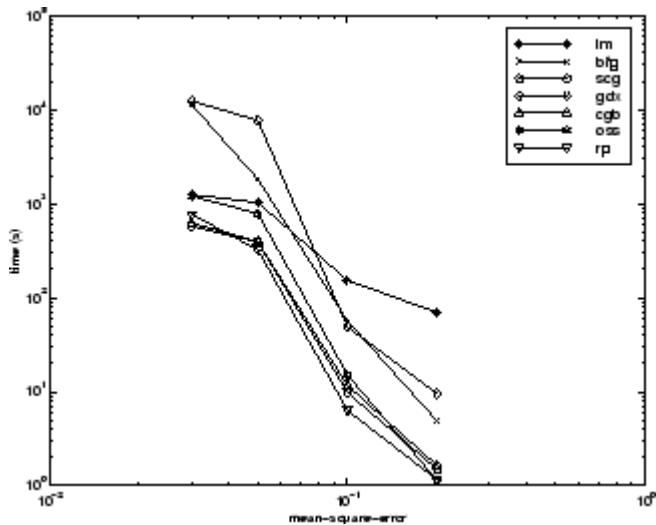
共轭梯度法和弹性反向传播都能快速收敛。此问题的结果与所考虑的其他模式识别问题是一致的。RP 算法在所有模式识别问题上都效果良好。这是合理的，因为该算法旨在克服用 sigmoid 函数进行训练所带来的困难，当远离中心点执行运算时，sigmoid 函数具有非常小的斜率。对于模式识别问题，您要在输出层中使用 sigmoid 传递函数，并且要网络在 sigmoid 函数的尾部运行。

算法	均值时间 (秒)	比率	最长时间 (秒)	最大时间 (秒)	标准 (秒)
RP	323.90	1.00	187.43	576.90	111.37
SCG	390.53	1.21	267.99	487.17	75.07
CGB	394.67	1.22	312.25	558.21	85.38
CGP	415.90	1.28	320.62	614.62	94.77
OSS	784.00	2.42	706.89	936.52	76.37
CGF	784.50	2.42	629.42	1082.20	144.63
LM	1028.10	3.17	802.01	1269.50	166.31
BFG	1821.00	5.62	1415.80	3254.50	546.36
GDX	7687.00	23.73	5169.20	10350.00	2015.00

以下是一些典型算法的均方误差对执行时间的图。与其他问题一样，您可以看到 SCG 和 RP 具有快速的初始收敛，而 LM 算法能够提供较小的最终误差。



下图进一步说明各算法之间的关系，它是收敛所需的时间对均方误差收敛目标的图。在本例中，您可以看到随着误差目标的降低，BFG 算法的性能下降，而 LM 算法的性能不断改善。RP 算法表现最好，除了在最小的误差目标上 SCG 表现更好。



## 总结

从上述试验中可以推断出几个算法特征。一般情况下，在函数逼近问题上，对于包含多达几百个权重的网络，Levenberg-Marquardt 算法具有最快的收敛。如果需要非常准确的训练，这种优势尤其明显。在许多情况下，`trainlm` 能够获得比任何其他被测算法更低的均方误差。然而，随着网络中权重数量的增加，`trainlm` 的优势会减小。此外，`trainlm` 在模式识别问题上的表现相对较差。`trainlm` 的存储要求高于被测的其他算法。

`trainrp` 函数是在模式识别问题上最快的算法。然而，它在函数逼近问题上表现不佳。随着误差目标的降低，其性能也会下降。与考虑的其他算法相比，该算法的内存要求相对较小。

共轭梯度算法，特别是 `trainscg`，似乎在各种各样的问题上都表现良好，特别是对于具有大量权重个数的网络。SCG 算法在函数逼近问题上几乎与 LM 算法一样快（对于大型网络更快），在模式识别问题上几乎

与 **trainrp** 一样快。当误差减小时，其性能不会像 **trainrp** 性能下降得那样快。共轭梯度算法具有相对适中的内存要求。

**trainbfg** 的表现与 **trainlm** 相似。它不需要像 **trainlm** 那样多的存储空间，但所需的计算量确实会随着网络规模的增大而呈几何增长，因为每次迭代都必须计算等效逆矩阵。

可变学习率算法 **traingdx** 通常比其他方法慢得多，其存储要求与 **trainrp** 大致相同，但它仍可用于求解某些问题。在某些情况下，收敛速度越慢越好。例如，当使用早停法时，如果使用收敛过快的算法，可能会产生不一致的结果。您可能会过冲在验证集上误差最小化的点。

## 提高浅层神经网络泛化能力，避免过拟合

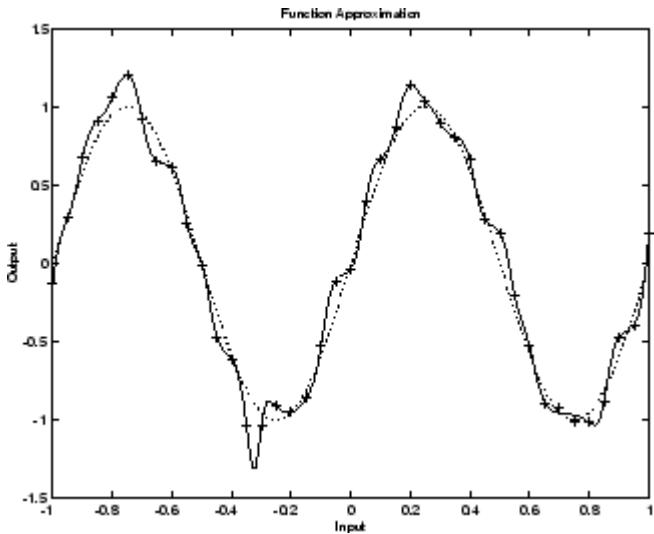
### 本节内容

- “重新训练神经网络”（第 27-25 页）
- “多个神经网络”（第 27-25 页）
- “早停法”（第 27-26 页）
- “索引数据划分 (divideind)”（第 27-27 页）
- “随机数据划分 (dividerand)”（第 27-27 页）
- “分块数据划分 (divideblock)”（第 27-27 页）
- “交错数据划分 (divideint)”（第 27-27 页）
- “正则化”（第 27-28 页）
- “早停法和正则化的摘要与讨论”（第 27-29 页）
- “训练后分析 (regression)”（第 27-30 页）

**提示** 要了解如何为深度学习网络设置参数，请参阅“设置参数并训练卷积神经网络”（第 1-35 页）。

神经网络训练中出现的问题之一称为过拟合。网络在训练集上的误差达到一个非常小的值、但在提交给网络的新数据上误差很大。网络记忆了训练示例，但尚未学会泛化应用于新情况。

下图显示一个 1-20-1 神经网络的响应，该网络已经过训练，用于逼近含噪正弦函数。虚线表示基础正弦函数，+ 符号表示含噪测量值，实线表示神经网络响应。很明显，此网络对数据进行了过拟合，将无法很好地泛化。



提高网络泛化能力的一种方法是使用足够大的网络来提供足够的拟合度。您使用的网络越大，网络可以创建的函数就越复杂。如果您使用的网络足够小，它将没有能力对数据进行过拟合。运行神经网络设计示例 **nnd11gn** [HDB96]，研究缩小网络规模如何防止过拟合。

遗憾的是，很难预知特定应用的合适网络规模。Deep Learning Toolbox 软件还实现了另外两种提高泛化能力的方法：正则化和早停法。接下来的几节说明这两种方法及其实现例程。

请注意，如果网络中的参数个数远远少于训练集中的总点数，则过拟合极少发生或根本不会发生。如果您可以轻松地采集更多数据并增加训练集的大小，则不必担心以下方法会防止过拟合。本节的其余部分仅适用于您要充分利用有限数据源的情况。

## 重新训练神经网络

通常情况下，每个反向传播训练会话都从不同的初始权重和偏置开始，并以不同方式将数据划分为训练集、验证集和测试集。对于同一个问题，这些不同条件会导致相差很大的解。

最好训练几个网络，以确保找到具有良好泛化能力的网络。

在此处，加载数据集并将其划分为两部分：90% 用于设计网络，10% 用于测试所有网络。

```
[x, t] = bodyfat_dataset;
Q = size(x, 2);
Q1 = floor(Q * 0.90);
Q2 = Q - Q1;
ind = randperm(Q);
ind1 = ind(1:Q1);
ind2 = ind(Q1 + (1:Q2));
x1 = x(:, ind1);
t1 = t(:, ind1);
x2 = x(:, ind2);
t2 = t(:, ind2);
```

接下来，选择一种网络架构并基于数据集的第一部分进行十次训练，并求出各网络架构在数据集第二部分上的均方误差。

```
net = feedforwardnet(10);
numNN = 10;
NN = cell(1, numNN);
perfs = zeros(1, numNN);
for i = 1:numNN
    fprintf('Training %d/%d\n', i, numNN);
    NN{i} = train(net, x1, t1);
    y2 = NN{i}(x2);
    perfs(i) = mse(net, t2, y2);
end
```

每个网络都将从不同初始权重和偏置开始进行训练，并将第一个数据集以不同比例划分为训练集、验证集和测试集。请注意，测试集适合度量每个对应网络的泛化能力，但不适合所有网络，因为作为一个网络的测试集的数据可能会被其他神经网络用于训练或验证。这也是为什么原始数据集被划分为两部分，这是为了确保保留完全独立的测试集。

性能最差的神经网络是泛化用于数据集第二部分时效果最佳的网络。

## 多个神经网络

提高泛化能力的另一个简单方法是训练多个神经网络并对其输出取平均值，尤其是对于含噪数据或小型数据集。

例如，此处针对一个小问题训练 10 个神经网络，并将其均方误差与平均值的均方误差进行比较。

首先，加载数据集并将其划分为设计集和测试集。

```
[x, t] = bodyfat_dataset;
Q = size(x, 2);
Q1 = floor(Q * 0.90);
Q2 = Q - Q1;
ind = randperm(Q);
ind1 = ind(1:Q1);
ind2 = ind(Q1 + (1:Q2));
x1 = x(:, ind1);
t1 = t(:, ind1);
x2 = x(:, ind2);
t2 = t(:, ind2);
```

然后，训练十个神经网络。

```
net = feedforwardnet(10);
numNN = 10;
nets = cell(1, numNN);
for i = 1:numNN
    fprintf('Training %d/%d\n', i, numNN)
    nets{i} = train(net, x1, t1);
end
```

接下来，基于第二个数据集对每个网络进行测试，得到单一网络性能和平均输出性能。

```
perfs = zeros(1, numNN);
y2Total = 0;
for i = 1:numNN
    neti = nets{i};
    y2 = neti(x2);
    perfs(i) = mse(neti, t2, y2);
    y2Total = y2Total + y2;
end
perfs
y2AverageOutput = y2Total / numNN;
perfAveragedOutputs = mse(nets{1}, t2, y2AverageOutput)
```

平均输出的均方误差可能低于大多数单一网络的性能，但未必是全部。它可能会更好地泛化应用于其他新数据。

对于一些高难度问题，可以训练 100 个网络，并对任一输入取其输出的平均值。这尤其适合小型含噪数据集与贝叶斯正则化训练函数 `trainbr` 结合使用的情况，如下所述。

## 早停法

提高泛化能力的默认方法称为早停法。该方法自动提供给所有的有监督网络创建函数，包括反向传播网络创建函数，如 `feedforwardnet`。

在此方法中，可用数据划分为三个子集。第一个子集是训练集，用于计算梯度和更新网络权重及偏置。第二个子集是验证集。在训练过程中会监控基于验证集的误差。验证误差通常在训练的初始阶段减小，训练集误差也是如此。然而，当网络开始过拟合数据时，基于验证集的误差通常开始增大。当验证误差在指定次数的迭代 (`net.trainParam.max_fail`) 中增加时，训练停止，并返回验证误差最小时的权重和偏置。

训练期间不使用测试集误差，但测试集误差用于比较不同模型。在训练过程中绘制测试集误差也很有用。如果测试集误差与验证集误差达到最小值所需的迭代次数显著不同，这可能表示数据集的划分不佳。

可使用四个函数将数据划分为训练集、验证集和测试集。它们是 `dividerand`（默认值）、`divideblock`、`divideint` 和 `divideind`。您可以使用以下属性访问或更改网络的划分函数：

```
net.divideFcn
```

其中每个函数都采用自定义其行为的参数。这些值会被存储，且可以通过以下网络属性进行更改：

```
net.divideParam
```

## 索引数据划分 (divideind)

创建一个简单的测试问题。对于完整的数据集，以 0.01 的步长生成有 201 个输入点的含噪正弦波，输入点范围为 -1 至 1：

```
p = [-1:0.01:1];
t = sin(2*pi*p)+0.1*randn(size(p));
```

按索引划分数据，以便将连续样本依次分配给训练集、验证集和测试集：

```
trainInd = 1:3:201;
valInd = 2:3:201;
testInd = 3:3:201;
[trainP,valP,testP] = divideind(p,trainInd,valInd,testInd);
[trainT,valT,testT] = divideind(t,trainInd,valInd,testInd);
```

## 随机数据划分 (dividerand)

您可以随机划分输入数据，以便将 60% 的样本分配给训练集，20% 分配给验证集，20% 分配给测试集，如下所示：

```
[trainP,valP,testP,trainInd,valInd,testInd] = dividerand(p);
```

该函数不仅划分输入数据，还返回索引，以便您可以使用 `divideind` 相应地划分目标数据：

```
[trainT,valT,testT] = divideind(t,trainInd,valInd,testInd);
```

## 分块数据划分 (divideblock)

您还可以随机划分输入数据，以便将前 60% 的样本分配给训练集，接下来的 20% 分配给验证集，最后 20% 分配给测试集，如下所示：

```
[trainP,valP,testP,trainInd,valInd,testInd] = divideblock(p);
```

使用 `divideind` 相应地划分目标数据：

```
[trainT,valT,testT] = divideind(t,trainInd,valInd,testInd);
```

## 交错数据划分 (divideint)

划分输入数据的另一种方法是根据百分比将样本循环分配给训练集、验证集和测试集。您可以按照交错方式将 60% 的样本分配给训练集、20% 分配给验证集、20% 分配给测试集，如下所示：

```
[trainP,valP,testP,trainInd,valInd,testInd] = divideint(p);
```

使用 `divideind` 相应地划分目标数据。

```
[trainT,valT,testT] = divideind(t,trainInd,valInd,testInd);
```

## 正则化

另一种提高泛化能力的方法称为正则化。这涉及到修正性能函数，通常选择性能函数来作为基于训练集的网络误差的平方和。下一节解释如何修正性能函数，其后的一节说明自动设置最优性能函数以实现最佳泛化的例程。

### 修正的性能函数

用于训练前馈神经网络的典型性能函数是网络误差的均值平方和。

$$F = mse = \frac{1}{N} \sum_{i=1}^N (e_i)^2 = \frac{1}{N} \sum_{i=1}^N (t_i - \alpha_i)^2$$

如果您通过添加由网络权重和偏置的平方和均值组成的项  $msereg = \gamma * msw + (1 - \gamma) * mse$  来修正性能函数，则可能会提高泛化能力；其中  $\gamma$  是性能比，并且

$$msw = \frac{1}{n} \sum_{j=1}^n w_j^2$$

使用此性能函数会减小网络的权重和偏置，这将迫使网络响应更平滑，降低过拟合的可能性。

以下代码重新初始化以前的网络，并使用 BFGS 算法和正则化性能函数对其进行重新训练。此处，性能比设置为 0.5，这使得均方误差和均方权重具有相等的权重。

```
[x,t] = simplefit_dataset;
net = feedforwardnet(10,'trainbfg');
net.divideFcn = '';
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-5;
net.performParam.regularization = 0.5;
net = train(net,x,t);
```

正则化的问题是很难确定性能比参数的最优值。如果将此参数设置得太大，可能会出现过拟合。如果该比率太小，则网络无法充分拟合训练数据。下一节说明自动设置正则化参数的例程。

### 自动正则化 (trainbr)

该方法以自动方式确定最佳的正则化参数。此过程使用的一种方法是 David MacKay 提出的贝叶斯框架 [MacK92]。在此框架中，网络的权重和偏置假设为服从指定分布的随机变量。正则化参数与这些分布的未知方差相关。然后，您可以使用统计方法来估计这些参数。

贝叶斯正则化的详细讨论超出了本用户指南的范围。有关结合使用贝叶斯正则化和 Levenberg-Marquardt 训练的详细说明，请参阅 [FoHa97]。

贝叶斯正则化已通过函数 `trainbr` 实现。以下代码说明如何使用此函数来训练 1-20-1 网络，以逼近“提高浅层神经网络泛化能力，避免过拟合”（第 27-24 页）的图中所示的含噪正弦波。（通过设置 `net.divideFcn` 取消数据划分，从而独立于早停法分析 `trainbr` 的效果。）

```
x = -1:0.05:1;
t = sin(2*pi*x) + 0.1*randn(size(x));
net = feedforwardnet(20,'trainbr');
net = train(net,x,t);
```

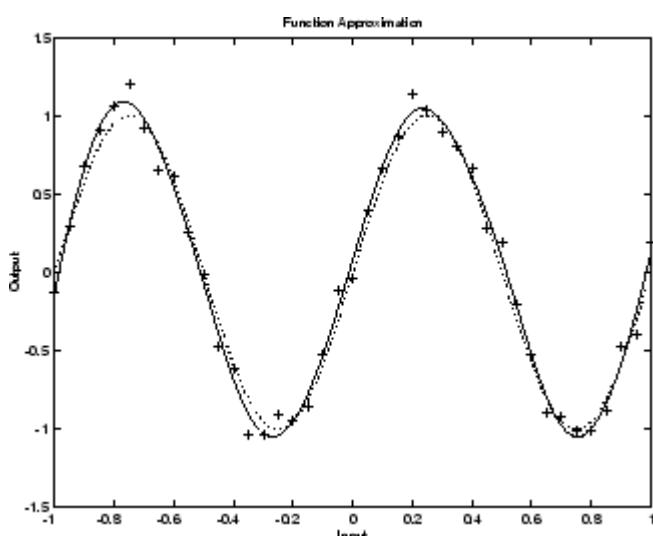
该算法的一个特点是，它可以度量网络实际上在使用多少个网络参数（权重和偏置）。在本例中，最终经过训练的网络使用 1-20-1 网络中的 61 个总权重和偏置中的大约 12 个参数（由打印输出中的 #Par 指

示)。不管网络中的参数个数变得多大，此实际参数个数应保持大致相同。(此论述假设网络已训练足够的迭代次数以确保收敛。)

**trainbr** 算法通常在网络输入和目标缩放到大致在 [-1,1] 范围内时效果最好。此处的测试问题是这种情况。如果您的输入和目标不在此范围内，可以使用函数 **mapminmax** 或 **mapstd** 来执行缩放，如“选择神经网络输入输出处理函数”(第 21-6 页) 中所述。默认情况下，使用 **feedforwardnet** 创建的网络包含 **mapminmax** 作为输入和输出处理函数。

下图显示经过训练的网络的响应。与上一个图(图中 1-20-1 网络过拟合数据)相比，您可以看到网络响应非常接近基础正弦函数(虚线)，因此，网络对新输入具有很强的泛化能力。您可以尝试更大的网络，但网络响应永远不会过拟合数据。这就消除了确定最优网络大小所需的猜测工作。

当使用 **trainbr** 时，务必让算法运行到实际参数个数收敛为止。训练可能会停止并显示“Maximum MU reached”消息，这是典型情形，明确表示算法已真正收敛。如果误差平方和(SSE)和权重平方和(SSW)在几次迭代中相对恒定，则也可以视为算法已收敛。出现这种情况时，您可能需要点击训练窗口中的 **Stop Training** 按钮。



## 早停法和正则化的摘要与讨论

如果应用得当，早停法和正则化可以确保网络泛化。

对于早停法，您必须小心，不要使用收敛太快的算法。如果您使用快速算法(如 **trainlm**)，请合理设置训练参数，使收敛相对较慢。例如，将 **mu** 设置为相对较大的值(如 1)，并将 **mu\_dec** 和 **mu\_inc** 分别设置为接近 1 的值(如 0.8 和 1.5)。训练函数 **trainscg** 和 **trainbr** 通常在早停法中运行良好。

使用早停法时，验证集的选择也很重要。验证集应能代表训练集中的所有点。

当使用贝叶斯正则化时，务必将网络训练到收敛为止。当网络收敛时，误差平方和、权重平方和以及有效参数个数应达到恒定值。

使用早停法和正则化时，最好从几个不同初始条件开始训练网络。在某些情况下，其中一种方法可能失败。通过测试几种不同初始条件，您可以确认稳健网络性能。

当数据集很小并且您正在训练函数逼近网络时，贝叶斯正则化可提供比早停法更好的泛化性能。这是因为贝叶斯正则化不要求验证数据集与训练数据集分离；它使用所有数据。

为了深入了解算法的性能，我们在几个基准数据集上测试了早停法和贝叶斯正则化，如下表所示。

数据集标题	点数	网络	说明
BALL	67	2-10-1	小球位置测量中的双传感器标定
SINE (5% N)	41	1-15-1	高斯噪声水平为 5% 的单周期正弦波
SINE (2% N)	41	1-15-1	高斯噪声水平为 2% 的单周期正弦波
ENGINE (ALL)	1199	2-30-2	发动机传感器 - 完整数据集
ENGINE (1/4)	300	2-30-2	发动机传感器 - 数据集的 1/4
CHOLEST (ALL)	264	5-15-3	胆固醇测量 - 完整数据集
CHOLEST (1/2)	132	5-15-3	胆固醇测量 - 1/2 数据集

这些数据集有不同的大小以及不同数量的输入和目标。使用其中两个数据集，使用所有数据对网络进行一次训练，然后仅使用一部分数据重新训练。这说明当数据集较小时，贝叶斯正则化的优势变得更加明显。除了 SINE 数据集以外，所有数据集都是从物理系统获得的。这两个数据集是通过在正弦波的单个周期中添加不同水平的噪声而有意创建的。算法在这两个数据集上的性能说明了噪声的影响。

下表摘要显示了在七个测试集上早停法和贝叶斯正则化的性能。（`trainscg` 算法用于早停法测试。其他算法提供类似的性能。）

### 均方测试集误差

方法	Ball	Engine (All)	Engine (1/4)	Choles (All)	Choles (1/2)	Sine (5% N)	Sine (2% N)
ES	1.2e-1	1.3e-2	1.9e-2	1.2e-1	1.4e-1	1.7e-1	1.3e-1
BR	1.3e-3	2.6e-3	4.7e-3	1.2e-1	9.3e-2	3.0e-2	6.3e-3
ES/BR	92	5	4	1	1.5	5.7	21

您可以看到，在大多数情况下，贝叶斯正则化的性能优于早停法。当数据集很小时，或当数据集中几乎没有噪声时，性能的改善最为明显。例如 BALL 数据集，它是从噪声很小的传感器中获得的。

尽管贝叶斯正则化的泛化性能通常优于早停法，但并非始终如此。此外，在工具箱中实现的贝叶斯正则化形式在模式识别问题上的性能不如在函数逼近问题上。这是因为当网络输出饱和时，在 Levenberg-Marquardt 算法中使用的对 Hessian 矩阵的逼近不如在模式识别问题中准确。贝叶斯正则化方法的另一个缺点是，它通常比早停法花费更长的时间才能收敛。

## 训练后分析 (regression)

经过训练的网络的性能在一定程度上可以通过基于训练集、验证集和测试集的误差来度量，但更详细地调查网络响应通常是有用的。一种选择是执行网络响应和对应目标之间的回归分析。例程 `regression` 就是为执行此分析而设计的。

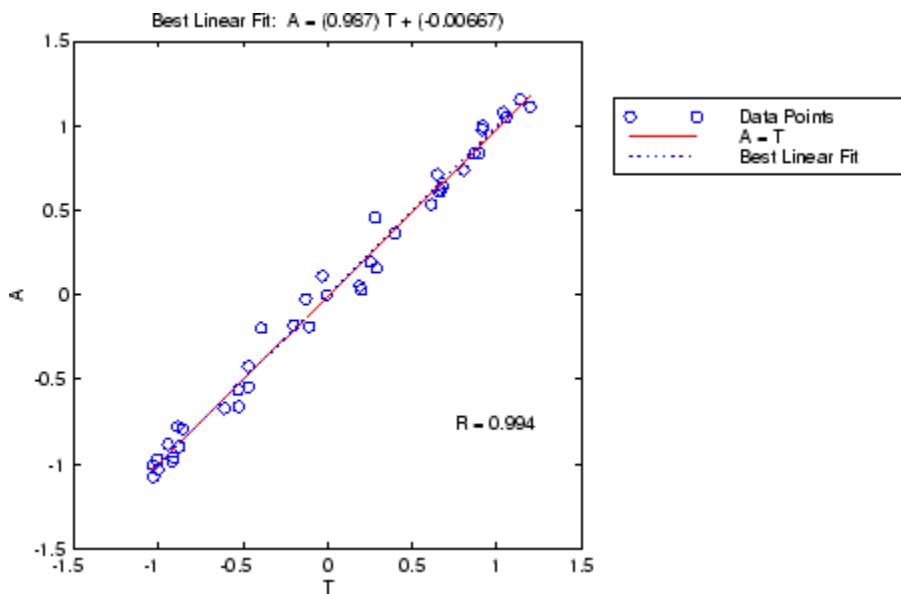
以下命令说明如何对经过训练的网络执行回归分析。

```
x = [-1:.05:1];
t = sin(2*pi*x)+0.1*randn(size(x));
net = feedforwardnet(10);
net = train(net,x,t);
y = net(x);
[r,m,b] = regression(t,y)
```

```
r =
0.9935
m =
0.9874
b =
-0.0067
```

将网络输出和对应的目标传递给 **regression**。它返回三个参数。前两个参数 **m** 和 **b** 对应于使目标与网络输出具有最佳线性回归的斜率和  $y$  截距。如果存在完美拟合（输出完全等于目标），则斜率为 1， $y$  截距为 0。在此示例中，您可以看到数值非常接近它们。**regression** 返回的第三个变量是输出和目标之间的相关系数（R 值）。它用于衡量目标解释输出变化的程度。如果此数值等于 1，则目标和输出之间就有完美的相关性。在此示例中，该数值非常接近 1，这表明拟合良好。

下图显示 **regression** 提供的图形输出。它是网络输出（用空心圆表示）对目标的图。最佳线性拟合由虚线表示。实线表示完美拟合（输出等于目标）。在此示例中，由于拟合非常好，很难区分最佳线性拟合线和完美拟合线。





# 历史神经网络

---

## 感知器神经网络

### 本节内容

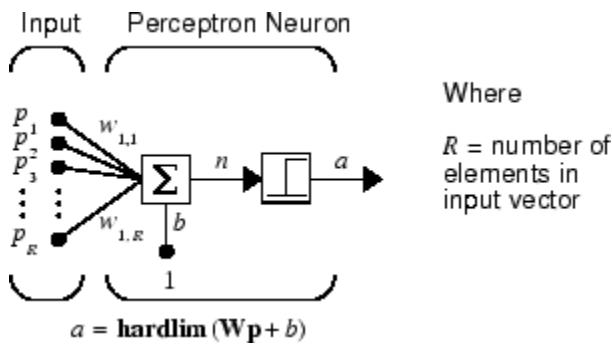
- “神经元模型”（第 28-2 页）
- “感知器架构”（第 28-3 页）
- “创建感知器”（第 28-4 页）
- “感知器学习规则 (learnp)”（第 28-5 页）
- “训练 (train)”（第 28-7 页）
- “限制和注意事项”（第 28-10 页）

Rosenblatt [Rose61] 创建了感知器的许多变体。其中最简单的一个是单层网络，其权重和偏置可以训练，以在向其提交对应的输入向量时生成正确的目标向量。使用的训练方法称为感知器学习规则。感知器引起了人们极大的兴趣，因为它能够从其训练向量中泛化，从最初随机分布的连接中学习。感知器特别适用于模式分类中的简单问题。对于它们求解的问题来说，它们是快速可靠的网络。此外，了解感知器的运行，能够为您了解更复杂的网络奠定良好的基础。

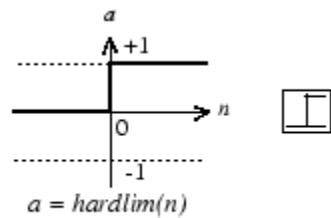
鉴于篇幅所限，本节仅对感知器进行了简短讨论。有关更多更深入的讨论，请参阅 [HDB1996] 的第 4 章 “Perceptron Learning Rule”，其中讨论了使用多层感知器求解单层无法解决的高难度问题。

### 神经元模型

使用硬限制传递函数 hardlim 的感知器神经元如下所示。



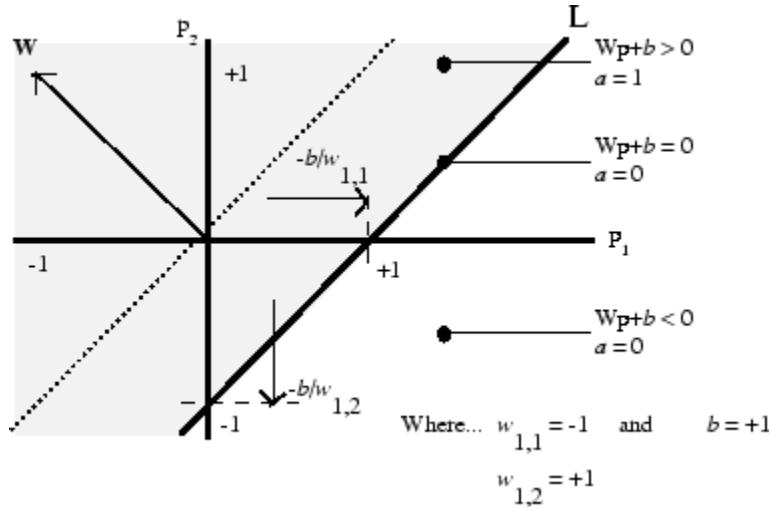
每个外部输入都用适当的权重  $w_{1j}$  进行加权，将加权输入之和发送到硬限制传递函数，该函数还通过偏置将输入 1 传递给它。硬限制传递函数返回 0 或 1，如下所示。



Hard-Limit Transfer Function

如果传递函数的净输入等于或大于 0，则感知器神经元生成 1；否则生成 0。

硬限制传递函数通过将输入空间划分为两个区域，使感知器能够对输入向量进行分类。具体来说，如果净输入  $n$  小于 0，输出将为 0；如果净输入  $n$  等于或大于 0，输出将为 1。下图显示了一个双输入硬限制神经元的输入空间，权重  $w_{1,1} = -1$ ,  $w_{1,2} = 1$  且偏置  $b = 1$ 。



两个分类区域由

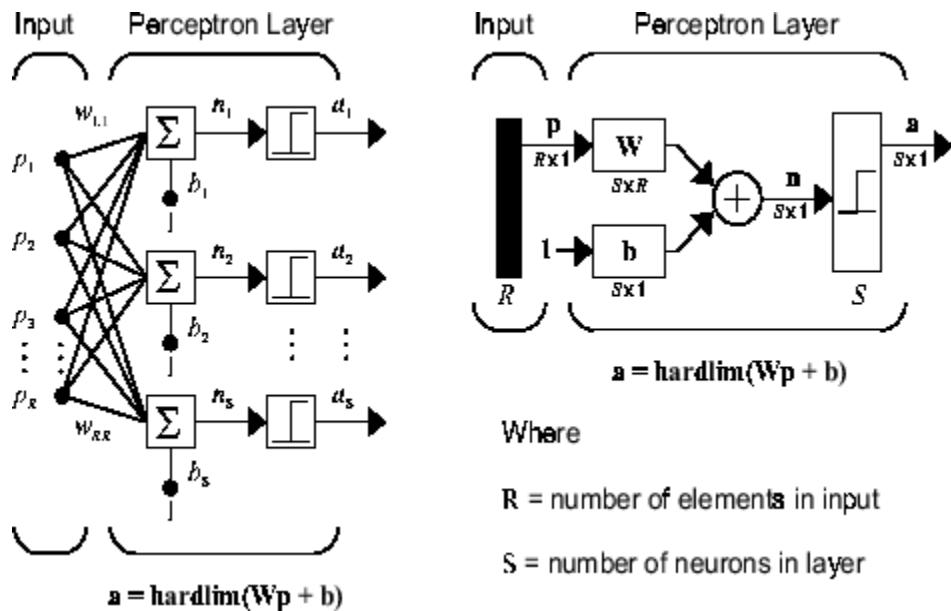
$\mathbf{Wp} + b = 0$  处的决策边界线  $L$  构成。这条线垂直于权重矩阵  $\mathbf{W}$  并根据偏置  $b$  偏移。线  $L$  上方和左侧的输入向量将生成大于 0 的净输入，因此，使硬限制神经元输出 1。线  $L$  下方和右侧的输入向量使神经元输出 0。您可以选择权重和偏置值来定向和移动分隔线，以便根据需要对输入空间进行分类。

没有偏置的硬限制神经元始终有一条分类线穿过原点。增加偏置可以让神经元求解两组输入向量不在原点不同侧的问题。该偏置允许决策边界偏离原点，如上图所示。

您可以运行示例程序 **nnd4db**。使用该示例程序，您可以移动决策边界，选择新输入进行分类，并了解如何通过反复应用学习规则生成一个确实能够对输入向量进行适当分类的网络。

## 感知器架构

感知器网络由单层  $S$  感知器神经元组成，这些神经元通过一组权重连接到  $R$  输入  $w_{ij}$ ，有如下所示的两种形式。与之前一样，网络索引  $i$  和  $j$  指示  $w_{ij}$  是从第  $j$  个输入到第  $i$  个神经元的连接的强度。



这种简单描述的感知器学习规则只能训练一层。因此，此处只考虑单层网络。这种约束限制了感知器可以执行的计算。感知器能够求解的问题类型在“限制和注意事项”（第 28-10 页）中讨论。

## 创建感知器

您可以使用以下代码创建一个感知器：

```
net = perceptron;
net = configure(net,P,T);
```

其中输入参数如下：

- $P$  是由  $Q$  个输入向量组成的  $R \times Q$  矩阵，每个输入向量包含  $R$  个元素。
- $T$  是由  $Q$  个目标向量组成的  $S \times Q$  矩阵，每个目标向量包含  $S$  个元素。

通常，`hardlim` 函数用于感知器，因此该函数是默认值。

以下命令创建一个感知器网络，该网络有一个单元素输入向量（值为 0 和 2）和一个神经元（输出可以是 0 或 1）：

```
P = [0 2];
T = [0 1];
net = perceptron;
net = configure(net,P,T);
```

您可以通过执行以下命令来查看创建了什么网络：

```
inputweights = net.inputweights{1,1}
```

它生成

```
inputweights =
  delays: 0
  initFcn: 'initzero'
  learn: true
```

```

learnFcn: 'learnp'
learnParam: (none)
    size: [1 1]
weightFcn: 'dotprod'
weightParam: (none)
userdata: (your custom info)

```

默认学习函数是 **learnp**, 在“感知器学习规则 (learnp)”(第 28-5 页)中讨论该函数。**hardlim** 传递函数的净输入是 **dotprod**, 它生成输入向量和权重矩阵的乘积, 并与偏置相加以计算净输入。

默认初始化函数 **initzero** 用于将权重的初始值设置为零。

类似地,

```
biases = net.biases{1}
```

可得

```

biases =
initFcn: 'initzero'
learn: 1
learnFcn: 'learnp'
learnParam: []
size: 1
userdata: [1x1 struct]

```

您可以看到, 偏置的默认初始化值也是 0。

## 感知器学习规则 (learnp)

感知器是基于期望行为的示例来训练的。期望的行为可以通过一组输入、输出对组来汇总:

$$\mathbf{p}_1 \mathbf{t}_1, \mathbf{p}_2 \mathbf{t}_2, \dots, \mathbf{p}_Q \mathbf{t}_Q$$

其中 **p** 是网络的输入, **t** 是对应的正确 (目标) 输出。目标是减少误差 **e**, 表示为 **t - a** 之差, 即神经元响应 **a** 和目标向量 **t** 之间的差。感知器学习规则 **learnp** 在给定输入向量 **p** 和关联误差 **e** 的情况下, 计算期望的感知器权重和偏置变化。目标向量 **t** 必须包含 0 或 1 值, 因为感知器 (具有 **hardlim** 传递函数) 只能输出这些值。

每次执行 **learnp** 时, 感知器生成正确输出的可能性都会提高。如果存在一个解, 则证明感知器规则在有限次迭代中收敛于解。

如果不使用偏置, **learnp** 的求解方式是仅更改权重向量 **w**, 以指向要分类为 1 的输入向量并远离要分类为 0 的向量。这将产生垂直于 **w** 并对输入向量进行适当分类的决策边界。

一旦提交输入向量 **p** 并且网络响应 **a** 计算出来, 单个神经元可能出现三种情况:

**第 1 种情况。** 如果提交输入向量并且神经元的输出正确 ( $\mathbf{a} = \mathbf{t}$  且  $\mathbf{e} = \mathbf{t} - \mathbf{a} = 0$ ) , 则权重向量 **w** 不变。

**第 2 种情况。** 如果神经元输出为 0 但应为 1 ( $\mathbf{a} = 0$  且  $\mathbf{t} = 1$ , 而且  $\mathbf{e} = \mathbf{t} - \mathbf{a} = 1$ ) , 则将输入向量 **p** 与权重向量 **w** 相加。这使得权重向量更接近输入向量, 从而增加了输入向量在将来分类为 1 的几率。

**第 3 种情况。** 如果神经元输出为 1 但应为 0 ( $\mathbf{a} = 1$  且  $\mathbf{t} = 0$ , 而且  $\mathbf{e} = \mathbf{t} - \mathbf{a} = -1$ ) , 则从权重向量 **w** 中减去输入向量 **p**。这使得权重向量更偏离输入向量, 从而增加了输入向量在将来分类为 0 的几率。

可以使用误差  $\mathbf{e} = \mathbf{t} - \mathbf{a}$  对权重向量所做的更改  $\Delta\mathbf{w}$  更简洁地编写感知器学习规则:

**第1种情况。**如果  $e = 0$ , 则更改  $\Delta w$  等于 0。

**第2种情况。**如果  $e = 1$ , 则更改  $\Delta w$  等于  $p^T$ 。

**第3种情况。**如果  $e = -1$ , 则更改  $\Delta w$  等于  $-p^T$ 。

这三种情况都可以用一个表达式来表述:

$$\Delta w = (t - \alpha)p^T = ep^T$$

注意到偏置只是输入始终为 1 的权重, 可以得到神经元偏置变化的表达式:

$$\Delta b = (t - \alpha)(1) = e$$

对于一层神经元的情况, 可得到

$$\Delta W = (t - a)(p)^T = e(p)^T$$

和

$$\Delta b = (t - a) = e$$

感知器学习规则可以总结如下:

$$W^{new} = W^{old} + ep^T$$

和

$$b^{new} = b^{old} + e$$

其中  $e = t - a$ 。

现在尝试一个简单示例。从具有一个输入向量的单个神经元开始, 该输入向量只包含两个元素。

```
net = perceptron;
net = configure(net,[0;0],0);
```

为了简化问题, 将偏置设置为等于 0, 将权重设置为 1 和 -0.8:

```
net.b{1} = [0];
w = [1 -0.8];
net.IW{1,1} = w;
```

输入目标对组由下式给出

```
p = [1; 2];
t = [1];
```

您可以使用下式计算输出和误差

```
a = net(p)
a =
0
e = t-a
e =
1
```

并使用函数 `learnp` 求出权重的变化。

```
dw = learnp(w,p,[],[],[],[],e,[],[],[],[],[])
dw =
 1   2
```

然后，通过以下方式获得新权重：

```
w = w + dw
w =
 2.0000  1.2000
```

可以重复进行求新权重（和偏置）的过程，直到没有误差为止。前面提到过，对于所有可以由感知器求解的问题，感知器学习规则可保证在有限步数内收敛。其中包括所有线性可分的分类问题。在这种情况下，要分类的对象可以用一条线分离。

您可能想尝试示例 `nnd4pr`。它允许您选择新输入向量，并应用学习规则对它们进行分类。

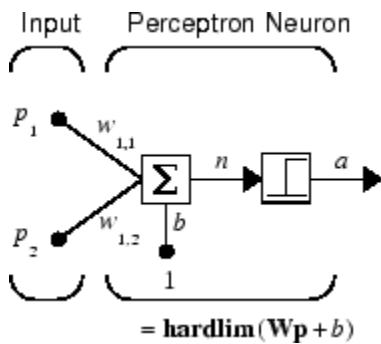
## 训练 (train)

如果重复使用 `sim` 和 `learnp` 向感知器提交输入，并根据误差更改感知器的权重和偏置，则感知器最终会找到求解问题的权重和偏置值，前提是感知器可以求解该问题。每一次经历所有训练输入和目标向量称为一次遍历。

函数 `train` 执行这样的循环计算。在每次遍历中，函数 `train` 都通过指定的输入序列，在提交输入时针对序列中的每个输入向量计算输出和误差并进行网络调整。

请注意，`train` 无法保证生成的网络能够实现预期目标。您必须通过计算每个输入向量的网络输出来检查 **W** 和 **b** 的新值，以查看是否达到所有目标。如果网络不能成功执行，您可以再次调用 `train` 对其进一步训练，使用新的权重和偏置进行更多遍训练，您也可以分析该问题，看看它是否适合用感知器求解。“限制和注意事项”（第 28-10 页）中讨论了感知器网络无法求解的问题。

我们通过一个简单问题来说明训练过程。假设有一个单神经元感知器，其单个向量输入具有两个元素：



此网络以及您将要求解的问题非常简单，您甚至可以通过手算来完成求解。下面讨论的问题引述了 [HDB1996] 中的内容。

假设您有以下分类问题，并希望用单向量输入、二元素感知器网络来求解它。

$$\left\{ \mathbf{p}_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, t_1 = 0 \right\} \left\{ \mathbf{p}_2 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, t_2 = 1 \right\} \left\{ \mathbf{p}_3 = \begin{bmatrix} -2 \\ 2 \end{bmatrix}, t_3 = 0 \right\} \left\{ \mathbf{p}_4 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, t_4 = 1 \right\}$$

使用初始权重和偏置。通过在变量后面的圆括号中使用数字来表示该计算的每步的变量。因此，上面的初始值是  $\mathbf{W}(0)$  和  $b(0)$ 。

$$\mathbf{W}(0) = [0 \ 0] \quad b(0) = 0$$

首先使用初始权重和偏置计算第一个输入向量  $\mathbf{p}_1$  的感知器输出  $a$ 。

$$\begin{aligned} a &= \text{hardlim}(\mathbf{W}(0)\mathbf{p}_1 + b(0)) \\ &= \text{hardlim}\left([0 \ 0]\begin{bmatrix} 2 \\ 2 \end{bmatrix} + 0\right) = \text{hardlim}(0) = 1 \end{aligned}$$

输出  $a$  与目标值  $t_1$  不相等，因此使用感知器规则根据误差找出权重和偏置的增量变化。

$$\begin{aligned} e &= t_1 - a = 0 - 1 = -1 \\ \Delta\mathbf{W} &= e\mathbf{p}_1^T = (-1)[2 \ 2] = [-2 \ -2] \\ \Delta b &= e = -1 = -1 \end{aligned}$$

您可以使用感知器更新规则计算新的权重和偏置。

$$\begin{aligned} \mathbf{W}^{new} &= \mathbf{W}^{old} + e\mathbf{p}^T = [0 \ 0] + [-2 \ -2] = [-2 \ -2] = \mathbf{W}(1) \\ b^{new} &= b^{old} + e = 0 + (-1) = -1 = b(1) \end{aligned}$$

现在提交下一个输入向量  $\mathbf{p}_2$ 。输出的计算如下。

$$\begin{aligned} a &= \text{hardlim}(\mathbf{W}(1)\mathbf{p}_2 + b(1)) \\ &= \text{hardlim}\left([-2 \ -2]\begin{bmatrix} 1 \\ -2 \end{bmatrix} - 1\right) = \text{hardlim}(1) = 1 \end{aligned}$$

在这种情况下，目标是 1，所以误差为零。因此，权重或偏置没有变化，即  $\mathbf{W}(2) = \mathbf{W}(1) = [-2 \ -2]$  且  $b(2) = b(1) = -1$ 。

您可以继续以这种方式提交  $\mathbf{p}_3$ ，计算输出和误差，并更改权重和偏置等。对所有四个输入进行一次遍历后，您将得到值  $\mathbf{W}(4) = [-3 \ -1]$  和  $b(4) = 0$ 。要确定是否获得了满意解，请对所有输入向量进行一次遍历，看看它们是否都生成了所需的目标值。第四个输入并未生成所需的目标值，但算法在第六次提交输入时是收敛的。最终值是

$$\mathbf{W}(6) = [-2 \ -3] \text{ 和 } b(6) = 1.$$

这是手算的结果。现在，如何使用 `train` 函数来实现呢？

以下代码定义一个感知器。

```
net = perceptron;
```

假设有以下单一输入

```
p = [2; 2];
```

目标为

```
t = [0];
```

将 `epochs` 设置为 1，这样 `train` 只需遍历一次输入向量（本例中只有一个输入向量）。

```
net.trainParam.epochs = 1;
net = train(net,p,t);
```

新的权重和偏置是

```
w = net.iw{1,1}, b = net.b{1}
w =
-2 -2
b =
-1
```

因此，初始权重和偏置为 0，在只使用第一个向量进行训练后，其值为  $[-2 -2]$  和  $-1$ ，与手算结果相符。

现在应用第二个输入向量  $p_2$ 。在更改权重和偏置前，输出始终为 1，不过现在目标为 1，因此误差为 0，变化为零。您可以从上一个结果开始，按照这种方式反复应用一个新的输入向量。您可以使用 `train` 自动完成这项工作。

每轮都应用 `train`，对由所有四个输入向量组成的序列进行一次遍历。从网络定义开始。

```
net = perceptron;
net.trainParam.epochs = 1;
```

输入向量和目标是

```
p = [[2;2] [1;-2] [-2;2] [-1;1]]
t = [0 1 0 1]
```

现在使用以下代码训练网络

```
net = train(net,p,t);
```

新的权重和偏置是

```
w = net.iw{1,1}, b = net.b{1}
w =
-3 -1
b =
0
```

这与您之前手算的结果相同。

最后，使用每个输入来对经过训练的网络进行仿真。

```
a = net(p)
a =
0 0 1 1
```

输出还不等于目标，所以需要通过多次遍历来训练网络。尝试进行多轮训练。该运行在两轮训练后的均值绝对误差性能为 0：

```
net.trainParam.epochs = 1000;
net = train(net,p,t);
```

这样，在第三轮训练中提交输入时，网络已完成训练。（通过手算可知，网络在提交第六个输入向量时收敛。这种情况发生在第二轮训练中，但要在第三轮才能检测到网络收敛。）最终的权重和偏置是

```
w = net.iw{1,1}, b = net.b{1}
w =
```

```

-2 -3
b =
1

```

各种输入的仿真输出和误差为

```

a = net(p)
a =
    0      1      0      1
error = a-t
error =
    0      0      0      0

```

您确认训练过程十分成功。该网络收敛并为四个输入向量生成正确的目标输出。

使用 **perceptron** 创建的网络的默认训练函数是 **trainc**。（您可以通过执行 **net.trainFcn** 来查找此函数。）该训练函数以其纯形式应用感知器学习规则，即依次单独应用各输入向量，并在每次提交一个输入向量后对权重和偏置进行更正。因此，使用 **train** 进行的感知器训练将在有限步数内收敛，除非提交的问题无法用简单的感知器求解。

其他网络也可以通过各种方式使用 **train** 函数。键入 **help train** 可了解关于此基本函数的更多信息。

您可能想尝试各种示例程序。例如，“用双输入感知器分类”（第 30-101 页）说明简单感知器的分类与训练。

## 限制和注意事项

应该使用 **adapt** 来训练感知器网络，该函数按一次一个的方式向网络提交输入向量，并根据每次提交的结果来更正网络。以这种方式使用 **adapt** 可以保证任何线性可分问题都能在提交有限次数的训练数据后得以求解。

如前几页所述，也可以用函数 **train** 来训练感知器。通常情况下，当使用 **train** 训练感知器时，它会将输入批量提交给网络，并基于各次更正的总和来更正网络。遗憾的是，目前没有证据表明这种训练算法可以使感知器训练收敛。因此，不推荐使用 **train** 来训练感知器。

感知器网络有若干限制。首先，由于硬限制传递函数，感知器的输出值只能取两个值之一（0 或 1）。其次，感知器只能对线性可分的向量集进行分类。如果能绘制出一条直线或一个平面将输入向量正确分类，则输入向量是线性可分的。如果向量不是线性可分的，学习将永远无法到达对所有向量都正确分类的程度。然而，事实证明，如果向量是线性可分的，自适应训练的感知器始终能在有限时间内求得解。您可能想尝试“线性不可分向量”（第 30-118 页）。它显示尝试对非线性可分的输入向量分类的难度。

但公平来说，具有多个感知器的网络可用于求解更高难度的问题。例如，假设有一组向量（包含四个向量），您想将其分成不同组，可以绘制两条线将它们分开。我们可以找到一个双神经元网络，通过它的两个决策边界将输入分为四个类别。有关感知器的更多讨论和更复杂的感知器问题，请参阅 [HDB1996]。

## 离群值和归一化感知器规则

离群值输入向量的长度远远大于或小于其他输入向量，因此离群值的存在可能造成训练时间过长。应用感知器学习规则涉及到根据误差在当前权重和偏置基础上加减输入向量。因此，具有较大元素的输入向量可能导致权重和偏置的更改，而明显较小的输入向量需要很长时间才能克服这些更改。您可能想尝试“离群值输入向量”（第 30-106 页），查看离群值对训练的影响。

稍微更改感知器学习规则，即可让训练时间对极大或极小的离群值输入向量不太敏感。

以下是更新权重的原始规则：

$$\Delta \mathbf{w} = (t - \alpha) \mathbf{p}^T = e \mathbf{p}^T$$

如上所示，输入向量  $\mathbf{p}$  越大，它对权重向量  $\mathbf{w}$  的影响就越大。因此，如果一个输入向量明显大于其他输入向量，则较小的输入向量必须多次提交才能产生影响。

解决方法是将规则归一化，使每个输入向量对权重的影响有相同的量级：

$$\Delta \mathbf{w} = (t - \alpha) \frac{\mathbf{p}^T}{\|\mathbf{p}\|} = e \frac{\mathbf{p}^T}{\|\mathbf{p}\|}$$

通过函数 `learnpn` 来实现归一化感知器规则，该函数的调用方式与 `learnp` 完全相同。归一化感知器规则函数 `learnpn` 的执行时间稍微有所增加，但如果存在离群值输入向量，则可大大减少轮数。您可以尝试“归一化感知器规则”（第 30-112 页），了解归一化的训练规则的工作原理。



# 神经网络对象引用

---

## 神经网络对象属性

本节内容
“常规” (第 29-2 页)
“架构” (第 29-2 页)
“子对象结构体” (第 29-4 页)
“函数” (第 29-6 页)
“权重和偏置值” (第 29-8 页)

下列属性定义网络的基本特征。 “Neural Network Subobject Properties” 描述定义网络细节的属性。

### 常规

下列是神经网络的常规属性。

#### **net.name**

此属性由定义网络名称的字符串组成。 网络创建函数 (例如 **feedforwardnet**) 会适当地定义此属性。 但它可以根据所需设置为任何字符串。

#### **net userdata**

用户可使用此属性向网络对象添加自定义信息。 仅一个字段是预定义的。 它包含一条向所有 Deep Learning Toolbox 用户发出的秘密消息：

#### **net userdata.note**

### 架构

这些属性确定网络子对象 (包含输入、层、输出、目标、偏置和权重) 的数量以及它们之间的连接方式。

#### **net.numInputs**

此属性定义网络接收的输入的数目。 它可以设置为 0 或正整数。

#### **Clarification**

网络输入的数目和网络输入的大小不是同一件事。 输入的数目定义网络像接收输入一样接收的向量的组数。 每个输入的大小 (即每个输入向量中的元素数目) 由输入大小 (**net.inputs{i}.size**) 确定。

大多数网络只有一个输入，其大小由问题决定。

#### **副作用**

对此属性的任何更改，都会导致定义从输入到层的连接的矩阵的大小 (**net.inputConnect**) 发生更改，以及输入子对象元胞数组的大小 (**net.inputs**) 发生更改。

#### **net.numLayers**

此属性定义网络的层数。 它可以设置为 0 或正整数。

**副作用**

对此属性的任何更改都会更改定义与层的连接的每个布尔矩阵的大小:

```
net.biasConnect
net.inputConnect
net.layerConnect
net.outputConnect
```

并更改子对象结构体组成的每个元胞数组的大小 (其大小取决于层数) :

```
net.biases
net.inputWeights
net.layerWeights
net.outputs
```

并且还更改每个网络的可调参数属性的大小:

```
net.IW
net.LW
net.b
```

**net.biasConnect**

此属性定义哪些层具有偏置。它可以设置为由布尔值组成的任意  $N \times 1$  矩阵，其中  $N_l$  是网络层数 (**net.numLayers**)。第  $i$  层是否存在偏置可通过以下属性的 1 (或 0) 表示

**net.biasConnect(i)**

**副作用**

对此属性的任何更改，都会更改偏置元胞数组 (**net.biases**) 中的结构体存在与否，以及偏置向量元胞数组 (**net.b**) 中的向量存在与否。

**net.inputConnect**

此属性定义哪些层有来自输入的权重。

它可以设置为由布尔值组成的任意  $N_l \times N_l$  矩阵，其中  $N_l$  是网络层数 (**net.numLayers**)， $N_i$  是网络输入数 (**net.numInputs**)。从第  $j$  个输入进入第  $i$  层的权重存在与否通过 **net.inputConnect(i,j)** 的 1 (或 0) 表示。

**副作用**

对此属性的任何更改，都会更改输入权重子对象元胞数组 (**net.inputWeights**) 中的结构体存在与否，以及输入权重矩阵元胞数组 (**net.IW**) 中的矩阵存在与否。

**net.layerConnect**

此属性定义哪些层有来自其他层的权重。它可以设置为由布尔值组成的任意  $N_l \times N_l$  矩阵，其中  $N_l$  是网络层数 (**net.numLayers**)。从第  $j$  层进入第  $i$  层的权重存在与否通过以下属性的 1 (或 0) 表示

**net.layerConnect(i,j)**

**副作用**

对此属性的任何更改，都会更改层权重子对象元胞数组 (**net.layerWeights**) 中的结构体存在与否，以及层权重矩阵元胞数组 (**net.LW**) 中的矩阵存在与否。

**net.outputConnect**

此属性定义哪些层生成网络输出。它可以设置为由布尔值组成的任意  $1 \times N_l$  矩阵，其中  $N_l$  是网络层数 (`net.numLayers`)。来自第  $i$  层的网络输出存在与否通过 `net.outputConnect(i)` 的 1 (或 0) 表示。

**副作用**

对此属性的任何更改都会更改网络输出的数目 (`net.numOutputs`)，以及输出子对象组成的元胞数组 (`net.outputs`) 中是否存在结构体。

**net.numOutputs (只读)**

此属性指示网络有多少个输出。它始终等于 `net.outputConnect` 中 1 的数目。

**net.numInputDelays (只读)**

此属性指示仿真网络时输入必须延迟的时间步数。它始终设置为与任何网络输入权重相关联的最大延迟值：

```
numInputDelays = 0;
for i=1:net.numLayers
    for j=1:net.numInputs
        if net.inputConnect(i,j)
            numInputDelays = max( ...
                [numInputDelays net.inputWeights{i,j}.delays]);
        end
    end
end
```

**net.numLayerDelays (只读)**

此属性指示仿真网络时层输出必须延迟的时间步数。它始终设置为与任何网络层权重相关联的最大延迟值：

```
numLayerDelays = 0;
for i=1:net.numLayers
    for j=1:net.numLayers
        if net.layerConnect(i,j)
            numLayerDelays = max( ...
                [numLayerDelays net.layerWeights{i,j}.delays]);
        end
    end
end
```

**net.numWeightElements (只读)**

此属性指示网络中权重和偏置值的数目。它是存储在以下两个元胞数组中的矩阵的元素总数：

```
net.IW
new.b
```

**子对象结构体**

这些属性由相关结构体的元胞数组组成，这些结构体定义网络的每项输入、层、输出、目标、偏置和权重。

“Neural Network Subobject Properties” 中说明了每种子对象的属性。

**net.inputs**

此属性包含每个网络输入的属性结构体。它始终是一个由输入结构体组成的  $N_i \times 1$  元胞数组，其中  $N_i$  是网络输入的数目 (net.numInputs)。

定义第  $i$  个网络输入的属性的结构体位于

**net.inputs{i}**

如果一个神经网络只有一个输入，则无需元胞数组表示法即可访问 net.inputs{1}，如下所示：

**net.input**

**输入属性**

请参阅 “Inputs”，了解输入属性的说明。

**net.layers**

此属性包含每个网络层的属性结构体。它始终是一个由层结构体组成的  $N_l \times 1$  元胞数组，其中  $N_l$  是网络层的数目 (net.numLayers)。

定义第  $i$  层的属性的结构体位于 net.layers{i}。

**层属性**

请参阅 “Layers”，了解层属性的说明。

**net.outputs**

此属性包含每个网络输出的属性结构体。它始终是一个  $1 \times N_l$  元胞数组，其中  $N_l$  是网络输出的数目 (net.numOutputs)。

如果 net.outputConnect(i) 为 1 (或 0)，则定义来自第  $i$  层 (或空矩阵 []) 的输出的属性的结构体位于 net.outputs{i} 中。

如果一个神经网络在层  $i$  上只有一个输出，则无需元胞数组表示法即可访问 net.outputs{i}，如下所示：

**net.output**

**输出属性**

请参阅 “Outputs”，了解输出属性的说明。

**net.biases**

此属性包含每个网络偏置的属性结构体。它始终是一个  $N_l \times 1$  元胞数组，其中  $N_l$  是网络层的数目 (net.numLayers)。

如果 net.biasConnect(i) 为 1 (或 0)，则定义与第  $i$  层 (或空矩阵 []) 相关联的偏置的属性的结构体位于 net.biases{i} 中。

**偏置属性**

请参阅 “Biases”，了解偏置属性的说明。

### **net.inputWeights**

此属性包含每个网络输入权重的属性结构体。它始终是一个  $N_i \times N_i$  元胞数组，其中  $N_i$  是网络层数 (**net.numLayers**)， $N_i$  是网络输入的数目 (**net.numInputs**)。

如果 **net.inputConnect(i,j)** 为 1 (或 0)，则定义从第  $j$  个输入 (或空矩阵 **[]**) 进入第  $i$  层的权重的属性的结构体位于 **net.inputWeights{i,j}** 中。

#### **输入权重属性**

请参阅 “Input Weights”，了解输入权重属性的说明。

### **net.layerWeights**

此属性包含每个网络层权重的属性结构体。它始终是一个  $N_i \times N_i$  元胞数组，其中  $N_i$  是网络层的数目 (**net.numLayers**)。

如果 **net.layerConnect(i,j)** 为 1 (或 0)，则定义从第  $j$  层 (或空矩阵 **[]**) 进入第  $i$  层的权重的属性的结构体位于 **net.layerWeights{i,j}** 中。

#### **层权重属性**

请参阅 “Layer Weights”，了解层权重属性的说明。

## **函数**

下列属性定义当网络要自适应、初始化、测量其性能或训练时要使用的算法。

### **net.adaptFcn**

此属性定义网络自适应时要使用的函数。它可以设置为任何网络自适应函数的名称。每当调用 **adapt** 时，都使用网络自适应函数来执行自适应。

```
[net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)
```

要查看函数列表，请键入 **help nntrain**。

#### **副作用**

每当更改此属性时，网络的自适应参数 (**net.adaptParam**) 都会设置为包含新函数的参数和默认值。

### **net.adaptParam**

此属性定义当前自适应函数的参数和值。对当前自适应函数调用 **help** 可获得每个字段的说明：

```
help(net.adaptFcn)
```

### **net.derivFcn**

此属性定义当使用有监督算法 (如反向传播) 训练网络时用于计算误差梯度和 Jacobian 矩阵的导函数。您可以将此属性设置为任何导函数的名称。

要查看函数列表，请键入 **help nnderivative**。

**net.divideFcn**

此属性定义当使用有监督算法（如反向传播）训练网络时要使用的数据划分函数。您可以将此属性设置为划分函数的名称。

要查看函数列表，请键入 **help nndivision**。

**副作用**

每当更改此属性时，网络的自适应参数 (**net.divideParam**) 都会设置为包含新函数的参数和默认值。

**net.divideParam**

此属性定义当前数据划分函数的参数和值。要了解每个字段意义的说明，请键入以下命令：

```
help(net.divideFcn)
```

**net.divideMode**

此属性定义当调用数据划分函数时要划分的目标数据维度。对于静态网络，其默认值为 '**sample**'；对于动态网络，其默认值为 '**time**'。它也可以设置为 '**sampletime**' 来同时按样本和时间步划分目标，设置为 '**all**' 来按每个标量值划分目标，或设置为 '**none**' 来表示根本不划分数据（在这种情况下，所有数据均用于训练，没有数据用于验证或测试）。

**net.initFcn**

此属性定义用于初始化网络的权重矩阵和偏置向量的函数。每当调用 **init** 时，都会使用初始化函数来初始化网络：

```
net = init(net)
```

**副作用**

每当更改此属性时，网络的初始化参数 (**net.initParam**) 都会设置为包含新函数的参数和默认值。

**net.initParam**

此属性定义当前初始化函数的参数和值。对当前初始化函数调用 **help** 可获得每个字段意义的说明：

```
help(net.initFcn)
```

**net.performFcn**

此属性定义用于度量网络性能的函数。每当调用 **train** 时，都会在训练过程中使用性能函数来计算网络性能。

```
[net,tr] = train(NET,P,T,Pi,Ai)
```

要查看函数列表，请键入 **help nnperformance**。

**副作用**

每当更改此属性时，网络的性能参数 (**net.performParam**) 都会设置为包含新函数的参数和默认值。

**net.performParam**

此属性定义当前性能函数的参数和值。对当前性能函数调用 **help** 可获得每个字段意义的说明：

```
help(net.performFcn)
```

**net.plotFcns**

此属性由字符串行元胞数组组成，定义与网络相关联的绘图函数。train 函数打开的神经网络训练窗口为每个绘图函数显示一个按钮。在训练期间或训练后点击该按钮可打开所需绘图。

**net.plotParams**

此属性由结构体行元胞数组组成，定义 net.plotFcns 中每个绘图函数的参数和值。对每个绘图函数调用 help 可获得每个字段意义的说明：

```
help(net.plotFcns{i})
```

**net.trainFcn**

此属性定义用于训练网络的函数。它可以设置为每当调用 train 时用于训练网络的任何训练函数的名称。

```
[net,tr] = train(NET,P,T,Pi,Ai)
```

要查看函数列表，请键入 help nntrain。

**副作用**

每当更改此属性时，网络的训练参数 (net.trainParam) 都会设置为包含新函数的参数和默认值。

**net.trainParam**

此属性定义当前训练函数的参数和值。对当前训练函数调用 help 可获得每个字段意义的说明：

```
help(net.trainFcn)
```

**权重和偏置值**

下列属性定义网络的可调参数：其权重矩阵和偏置向量。

**net.IW**

此属性定义从网络输入进入各层的权重的权重矩阵。它始终是一个  $N_l \times N_i$  元胞数组，其中  $N_l$  是网络层数 (net.numLayers)， $N_i$  是网络输入的数目 (net.numInputs)。

如果 net.inputConnect(i,j) 为 1 (或 0)，则从第 j 个输入 (或空矩阵 []) 进入第 i 层的权重的权重矩阵位于 net.IW{i,j} 中。

该权重矩阵的行数和其进入的层的大小一样大 (net.layers{i}.size)。其列数与输入大小和权重相关联的延迟数的乘积一样多：

```
net.inputs{j}.size * length(net.inputWeights{i,j}.delays)
```

在某些网络中，预处理函数 net.inputs{i}.processFcns 默认指定为 'removeconstantrows'。在这种情况下，如果网络输入 X 包含 m 个行，其中所有行元素具有相同的值，则权重矩阵的列数比上述乘积少 m 个。有关网络输入 X 的详细信息，请参阅 train。

这些维度也可以从输入权重属性中获取：

```
net.inputWeights{i,j}.size
```

**net.LW**

此属性定义从其他层进入各层的权重的权重矩阵。它始终是一个  $N_l \times N_l$  元胞数组，其中  $N_l$  是网络层的数目 (**net.numLayers**)。

如果 **net.layerConnect(i,j)** 为 1 (或 0)，则从第 j 层 (或空矩阵 []) 进入第 i 层的权重的权重矩阵位于 **net.LW{i,j}** 中。

该权重矩阵的行数和其进入的层的大小一样大 (**net.layers{i}.size**)。其列数与层的大小和权重相关联的延迟数的乘积一样多：

```
net.layers{j}.size * length(net.layerWeights{i,j}.delays)
```

这些维度也可以从层权重属性中获取：

```
net.layerWeights{i,j}.size
```

**net.b**

此属性定义具有偏置的每层的偏置向量。它始终是一个  $N_l \times 1$  元胞数组，其中  $N_l$  是网络层的数目 (**net.numLayers**)。

如果 **net.biasConnect(i)** 为 1 (或 0)，则第 i 层 (或空矩阵 []) 的偏置向量位于 **net.b{i}** 中。

偏置向量中的元素数目始终等于与其相关联的层的大小 (**net.layers{i}.size**)。

此维度也可从偏置属性中获得：

```
net.biases{i}.size
```



# 函数逼近、聚类和控制示例

---

## 体脂估计

此示例说明函数拟合神经网络如何基于解剖学测量值来估计体脂率。

### 问题：估计体脂率

在此示例中，我们尝试构建一个神经网络，该网络可以估计通过 13 个身体属性进行描述的人的体脂率：

- 年龄 (岁)
- 重量 (磅)
- 身高 (英寸)
- 颈围 (cm)
- 胸围 (cm)
- 腹围 (cm)
- 臀围 (cm)
- 大腿围 (cm)
- 膝围 (cm)
- 踝围 (cm)
- 上臂 (伸展) 围 (cm)
- 前臂围 (cm)
- 腕围 (cm)

这是输入与相关目标输出匹配的拟合问题的示例，我们希望所创建的神经网络不仅可以根据已知输入估计已知目标，还能泛化成针对未被用于设计解决方案的输入准确估计输出。

### 为什么使用神经网络？

神经网络非常擅长处理函数拟合问题。具有足够元素（称为神经元）的神经网络可以以任意准确度拟合任何数据。它们特别适合处理非线性问题。鉴于真实情况（如体脂增加）的非线性特性，神经网络是处理此问题的优选方案。

十三个身体属性将作为神经网络的输入，体脂率将成为目标。

该网络将设计为使用体脂率已知的人体的解剖学数据进行训练以生成目标估值。

### 准备数据

通过将数据组织成两个矩阵（输入矩阵 X 和目标矩阵 T）来为神经网络设置函数拟合问题的数据。

输入矩阵的每个第 i 列都将有十三个元素，表示一个体脂率已知的人体。

目标矩阵的每个对应列都将有一个元素，表示体脂率。

使用以下命令加载一个这样的数据集。

```
[X,T] = bodyfat_dataset;
```

我们可以查看输入 X 和目标 T 的大小。

请注意，X 和 T 都有 252 列。这表示 252 个体型（输入）和相关体脂率（目标）。

输入矩阵 X 有十三行，表示十三个属性。目标矩阵 T 只有一行，因为每个示例只需要一个输出，即体脂率。

```
size(X)
size(T)
```

```
ans =
13 252
```

```
ans =
1 252
```

### 使用神经网络拟合函数

下一步是创建一个学习估计体脂率的神经网络。

由于神经网络以随机初始权重开始，因此该示例的结果在每次运行时都会略有不同。我们设置了随机种子来避免这种随机性。但这对于您自己的应用情形并不是必需的。

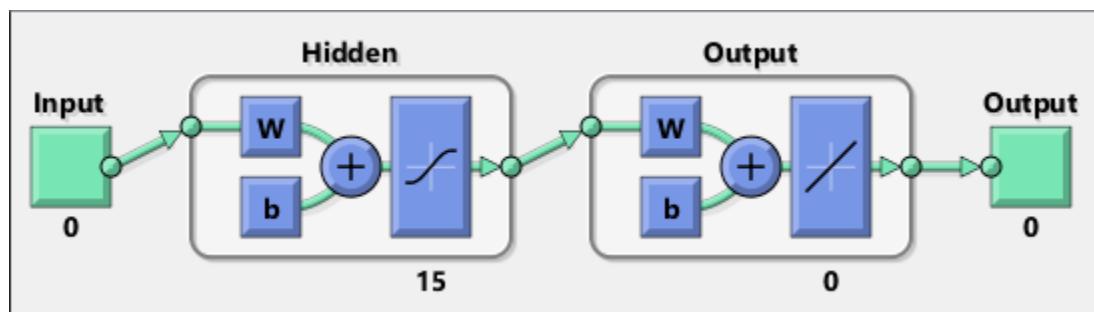
```
setdemorandstream(491218382)
```

双层（即，一个隐藏层）前馈神经网络可以拟合任何输入-输出关系，前提是隐藏层中有足够的神经元。非输出层称为隐含层。

对此示例，我们将尝试具有 15 个神经元的单隐藏层。一般情况下，问题越困难，需要的神经元和层就越多。问题越简单，需要的神经元就越少。

输入和输出的大小为 0，因为网络尚未配置成与我们的输入数据和目标数据相匹配。这将在训练网络时进行。

```
net = fitnet(15);
view(net)
```

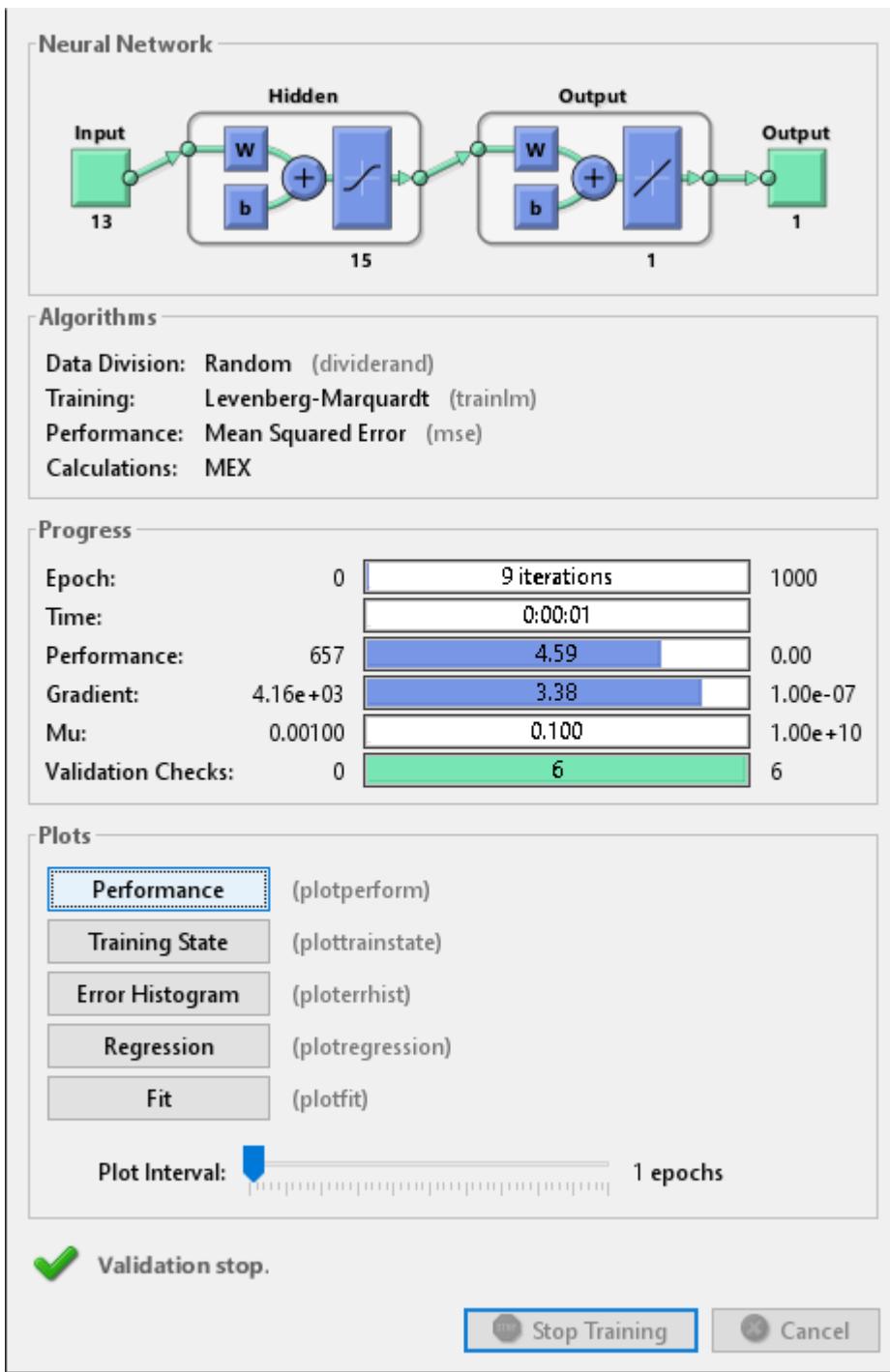


现在网络已准备就绪，可以开始训练。样本自动分为训练集、验证集和测试集。训练集用于对网络进行训练。只要网络针对验证集持续改进，训练就会继续。测试集提供完全独立的网络准确度测量。

神经网络训练工具显示正在接受训练的网络和用于训练该网络的算法。它还显示训练过程中的训练状态以及停止训练的条件（以绿色突出显示）。

底部的按钮用于打开有用的绘图，这些图可以在训练期间和训练后打开。算法名称和绘图按钮旁边的链接可打开有关这些主题的文档。

```
[net,tr] = train(net,X,T);
nntraintool
nntraintool('close')
```

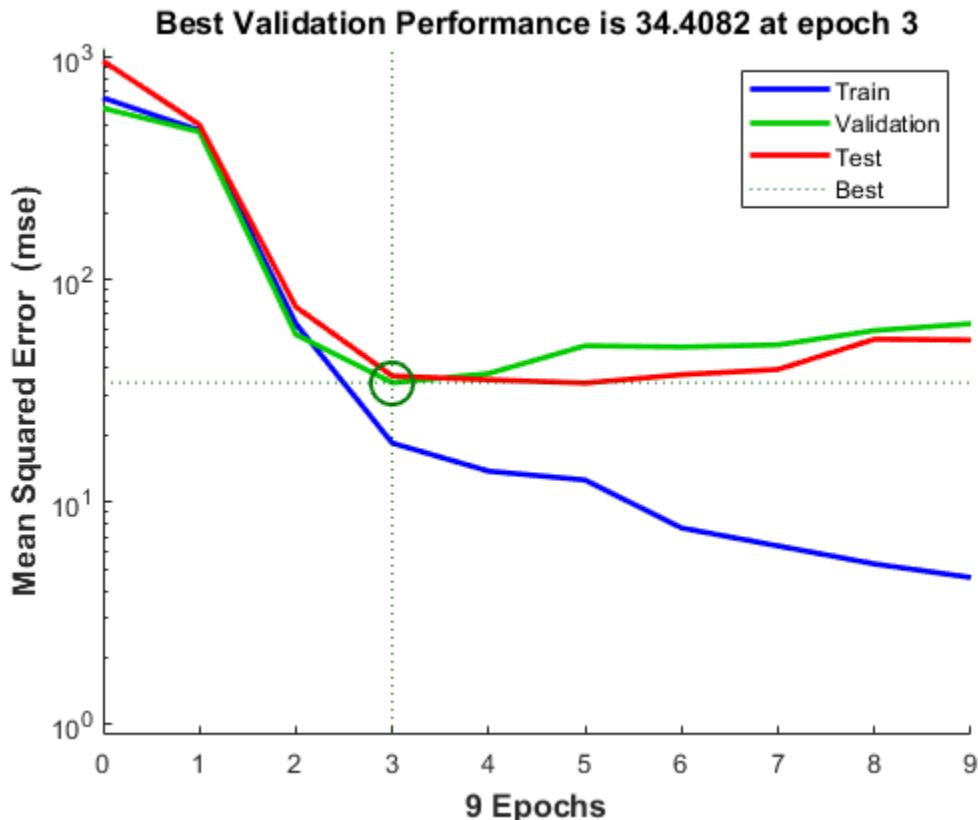


要查看在训练过程中网络性能的改善情况，请点击训练工具中的 Performance 按钮，或调用 `PLOTPERFORM`。

性能以均方误差衡量，并以对数刻度显示。随着网络训练的加深，均方误差迅速降低。

绘图会显示训练集、验证集和测试集的性能。最终网络是针对验证集性能最好的网络。

```
plotperform(tr)
```



### 测试神经网络

现在可以相对于测试样本测量经过训练的神经网络的均方误差。这将使我们能够了解网络在应用于真实数据时表现如何。

```
testX = X(:,tr.testInd);
testT = T(:,tr.testInd);

testY = net(testX);

perf = mse(net,testT,testY)

perf =
```

36.9404

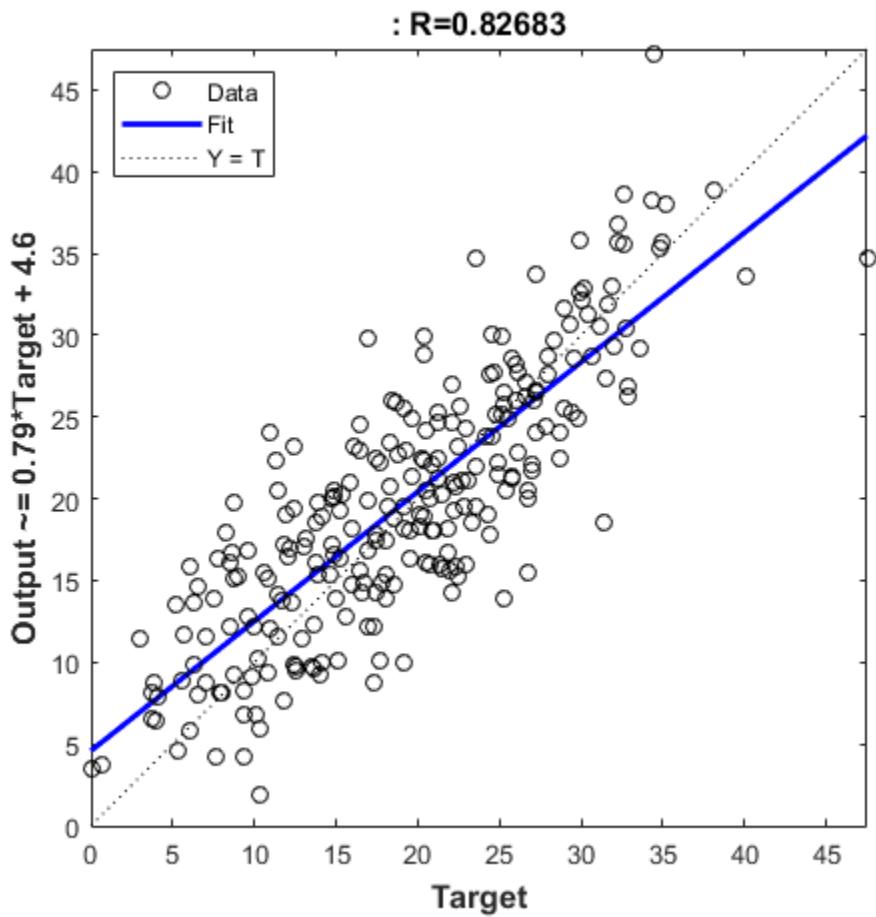
衡量神经网络数据拟合程度的另一个方法是回归图。下面绘制了所有样本的回归图。

回归图显示了根据相关目标值绘制的实际网络输出。如果网络已学会很好地拟合数据，则此输出-目标关系的线性拟合线与图边界的交点应该在左下角和右上角附近。

如果不是这样，则建议进一步进行训练，或训练具有更多隐藏神经元的网络。

```
Y = net(X);
```

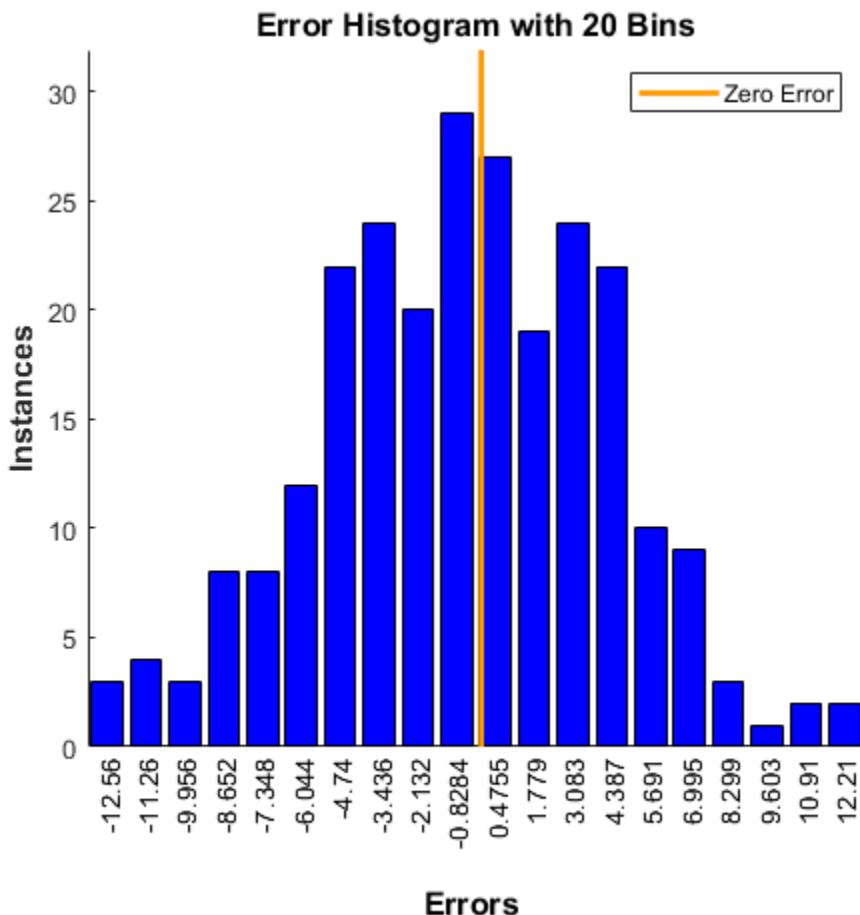
```
plotregression(T,Y)
```



衡量神经网络数据拟合程度的第三个方法是误差直方图。该图可显示误差大小的分布方式。通常，大多数误差接近于零，很少有误差大幅偏离。

```
e = T - Y;
```

```
ploterrhist(e)
```



此示例说明了如何设计一个通过身体特征来估计体脂率的神经网络。

请浏览其他示例和文档，以便更深入地了解神经网络及其应用。

## 螃蟹分类

此示例说明如何使用神经网络作为分类器来根据螃蟹的物理尺寸识别螃蟹的性别。

### 问题：螃蟹的分类

在此示例中，我们尝试构建一个可根据螃蟹的物理测量值识别螃蟹性别的分类器。我们将考虑螃蟹的六个物理特征：品种、前鳌、背宽、长度、宽度和厚度。现有问题是根据这 6 个物理特征的观测值识别螃蟹的性别。

### 为什么使用神经网络？

神经网络已证明是成熟的分类器，特别适合处理非线性问题。鉴于真实情况（如螃蟹分类）的非线性特性，神经网络无疑是解决该问题的优选方案。

六个物理特征将作为神经网络的输入，螃蟹的性别将成为目标。根据由螃蟹的六个物理特征观测值构成的输入，神经网络应识别出螃蟹是雄性还是雌性。

这通过将先前记录的输入提交给神经网络，然后调整网络以产生期望的目标输出来实现。此过程称为神经网络训练。

### 准备数据

通过将数据组织成两个矩阵（输入矩阵 X 和目标矩阵 T）来为神经网络设置分类问题的数据。

输入矩阵的每个第 i 列将具有六个元素，表示螃蟹的品种、前鳌、背宽、长度、宽度和厚度。

目标矩阵的每个对应列将具有两个元素。第一个元素中的一表示雌蟹，第二个元素中的一表示雄蟹。（所有其他元素均为零。）

使用以下命令加载该数据集。

```
[x,t] = crab_dataset;
size(x)
size(t)
```

```
ans =
```

```
6 200
```

```
ans =
```

```
2 200
```

### 构建神经网络分类器

下一步是创建一个学习识别螃蟹性别的神经网络。

由于神经网络以随机初始权重开始，因此该示例的结果在每次运行时都会略有不同。我们设置了随机种子来避免这种随机性。但这对于您自己的应用情形并不是必需的。

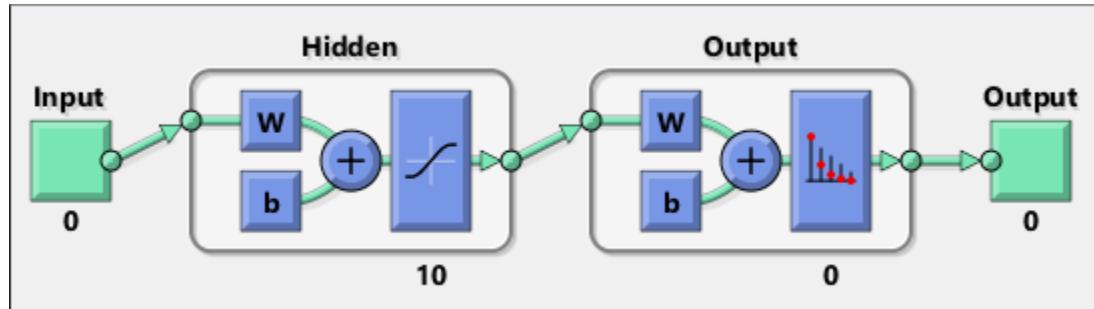
```
setdemosrandstream(491218382)
```

双层（即，一个隐藏层）前馈神经网络可以学习任何输入-输出关系，前提是隐藏层中有足够的神经元。非输出层称为隐含层。

对于此示例，我们将尝试具有 10 个神经元的单隐藏层。一般情况下，问题越困难，需要的神经元和层就越多。问题越简单，需要的神经元就越少。

输入和输出的大小为 0，因为网络尚未配置成与我们的输入数据和目标数据相匹配。这将在训练网络时进行。

```
net = patternnet(10);
view(net)
```



现在网络已准备就绪，可以开始训练。样本自动分为训练集、验证集和测试集。训练集用于对网络进行训练。只要网络针对验证集持续改进，训练就会继续。测试集提供完全独立的网络准确度测量。

```
[net,tr] = train(net,x,t);
nntraintool
```



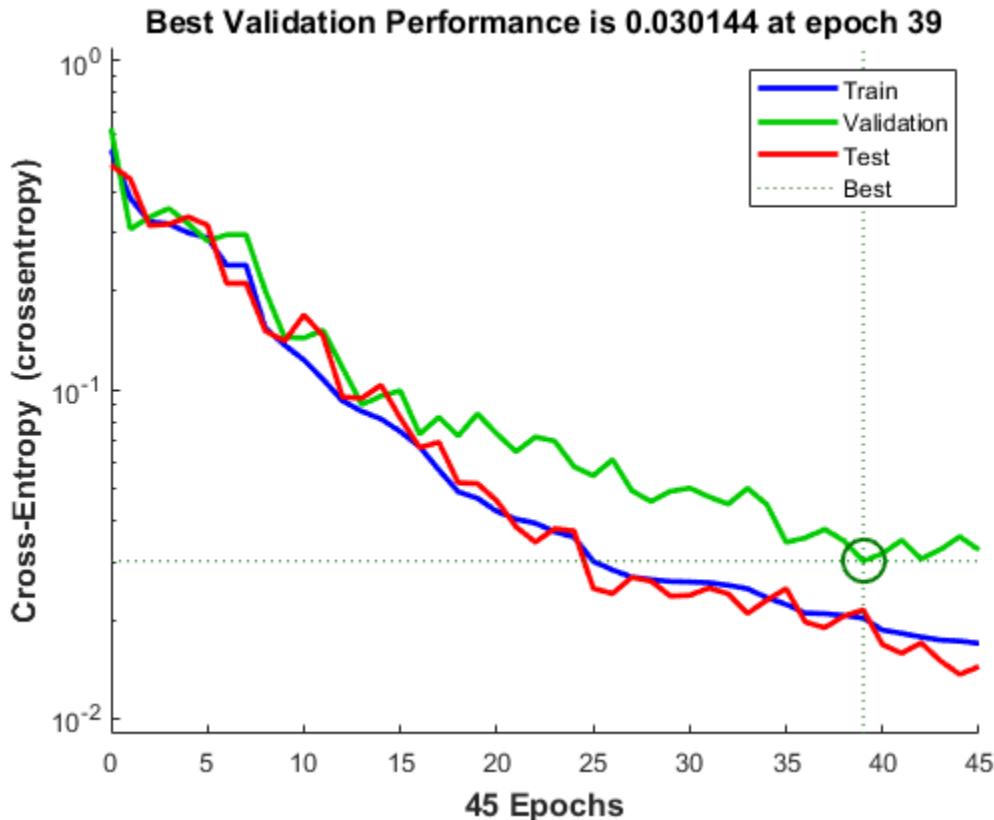
```
nntraintool('close')
```

要查看在训练过程中网络性能的改善情况，请点击训练工具中的 Performance 按钮，或调用 PLOTPERFORM。

性能以均方误差衡量，并以对数刻度显示。随着网络训练的加深，均方误差迅速降低。

绘图会显示训练集、验证集和测试集的性能。

```
plotperform(tr)
```



### 测试分类器

现在可以使用测试样本测试经过训练的神经网络。这将使我们能够了解网络在应用于真实数据时表现如何。

网络输出的范围为 0 到 1，因此我们可以使用 **vec2ind** 函数根据每个输出向量中最高元素的位置来获取类索引。

```
testX = x(:,tr.testInd);
testT = t(:,tr.testInd);

testY = net(testX);
testIndices = vec2ind(testY)

testIndices =
Columns 1 through 13
```

```
2 2 2 1 2 2 2 1 2 2 2 2 2 1
```

Columns 14 through 26

```
1 2 2 2 1 2 2 1 2 1 1 1 1 1
```

Columns 27 through 30

```
1 2 2 1
```

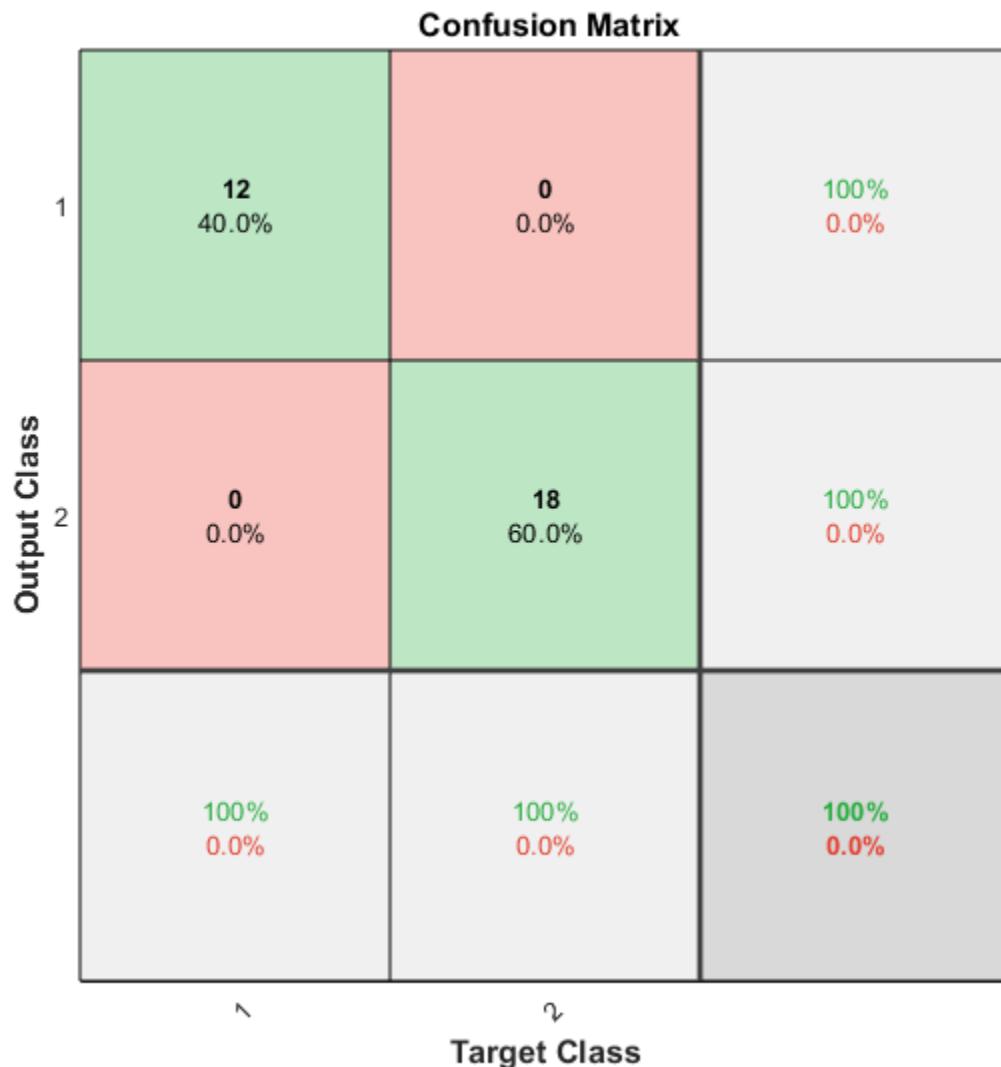
衡量神经网络数据拟合程度的一个方法是混淆矩阵图。下面绘制了所有样本的混淆矩阵图。

该混淆矩阵显示了正确和错误分类的百分比。正确分类表示为矩阵对角线上的绿色方块。错误分类表示为红色方块。

如果网络已学会正确分类，则红色方块中的百分比应该非常小，表示几乎没有错误分类。

如果不是这样，则建议进一步进行训练，或训练具有更多隐藏神经元的网络。

```
plotconfusion(testT,testY)
```



以下是正确和错误分类的总体百分比。

```
[c,cm] = confusion(testT,testY)

fprintf('Percentage Correct Classification : %f%%\n', 100*(1-c));
fprintf('Percentage Incorrect Classification : %f%%\n', 100*c);
```

c =

0.0333

cm =

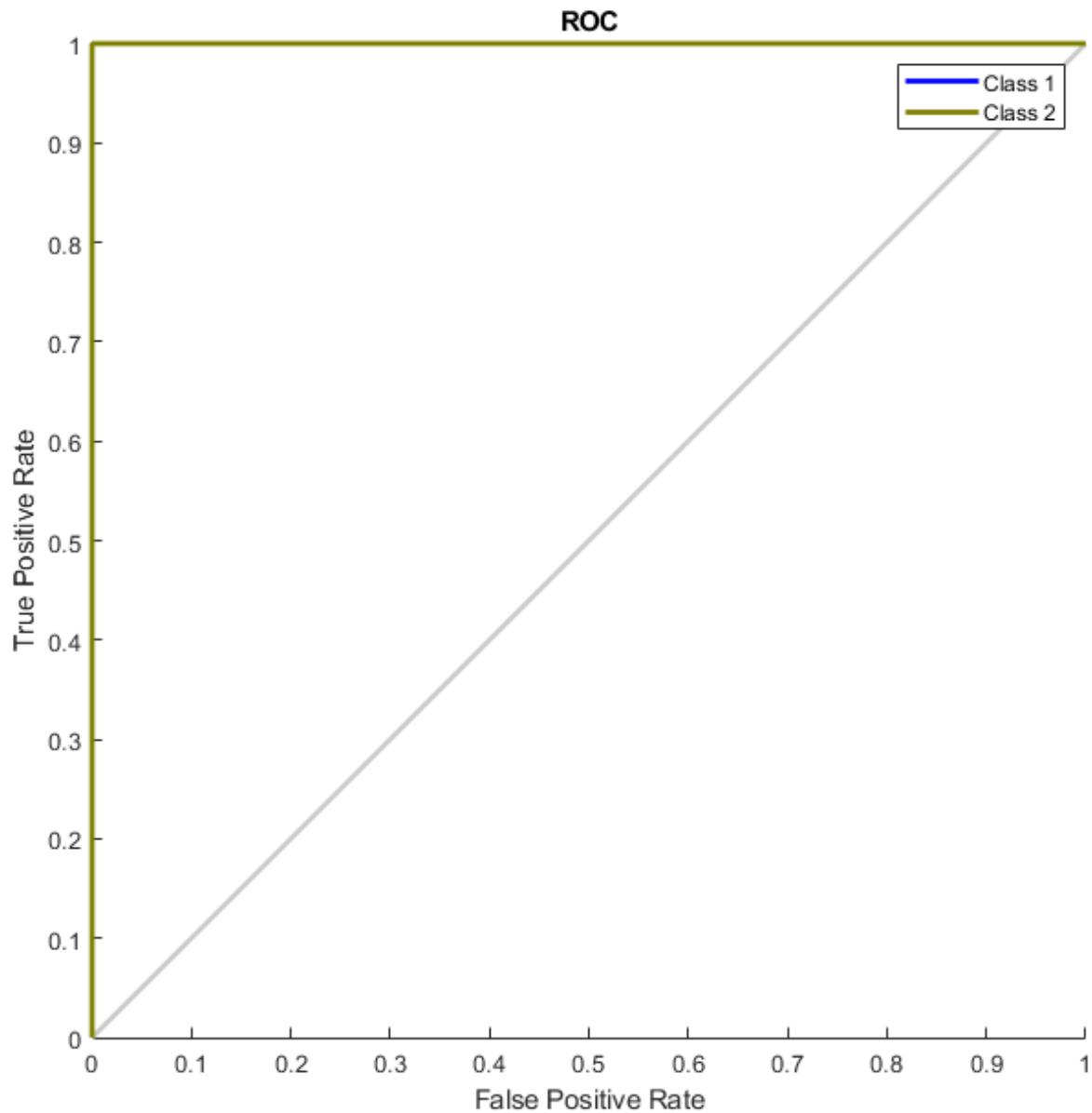
```
12    1  
0    17
```

```
Percentage Correct Classification : 96.666667%  
Percentage Incorrect Classification : 3.333333%
```

衡量神经网络数据拟合程度的另一个方法是受试者工作特征图。该图可显示随着输出阈值从 0 变为 1，假正率和真正率之间的相关性。

线条越偏向左上方，达到高的真正率所需接受的假正数越少。最佳分类器是线条从左下角到左上角再到右上角，或接近于该模式。

```
plotroc(testT,testY)
```



此示例说明了如何使用神经网络对螃蟹进行分类。

请浏览其他示例和文档，以便更深入地了解神经网络及其应用。

## 葡萄酒分类

此示例说明模式识别神经网络如何根据葡萄酒的化学特性按酒庄对葡萄酒进行分类。

### 问题：葡萄酒分类

在此示例中，我们尝试构建一个神经网络，该网络可以通过十三个属性对来自三个酒庄的葡萄酒进行分类：

- 酒精
- 苹果酸
- 灰
- 灰的碱度
- 镁
- 总酚
- 黄酮类
- 非黄酮类酚
- 原花青素
- 颜色深度
- 色调
- 稀释葡萄酒的 OD280/OD315
- 脯氨酸

这是输入与不同类相关的模式识别问题的示例，我们希望所创建的神经网络不仅可以对已知的葡萄酒正确分类，还能泛化成对未用于设计解决方案的葡萄酒进行准确分类。

### 为什么使用神经网络？

神经网络非常擅长处理模式识别问题。具有足够元素（称为神经元）的神经网络可以以任意准确度对任何数据进行分类。它们特别适合处理基于多种变数的复杂决策边界问题。因此，神经网络是处理葡萄酒分类问题的优选方案。

十三个相邻属性将作为神经网络的输入，并且每个属性的相应目标将是一个 3 元素类向量，其中相关酒庄 (#1、#2 或 #3) 的位置为 1。

该网络将设计为使用相邻属性训练网络以生成正确的目标类。

### 准备数据

通过将数据组织成两个矩阵（输入矩阵 X 和目标矩阵 T）来为神经网络设置分类问题的数据。

输入矩阵的每一列都有十三个元素，表示已知道其酒庄的葡萄酒。

目标矩阵的每个对应列将具有三个元素，包含两个零和相关酒庄位置上的 1。

使用以下命令加载一个这样的数据集。

```
[x,t] = wine_dataset;
```

我们可以查看输入 X 和目标 T 的大小。

请注意， $X$  和  $T$  都有 178 列。这些列代表 178 个葡萄酒样本属性（输入）和相关酒庄类向量（目标）。

输入矩阵  $X$  有十三行，表示十三个属性。目标矩阵  $T$  有三行，表示每个样本的三个可能酒庄。

```
size(x)
```

```
ans = 1×2
```

```
13 178
```

```
size(t)
```

```
ans = 1×2
```

```
3 178
```

### 使用神经网络进行模式识别

下一步是创建一个学习对葡萄酒进行分类的神经网络。

由于神经网络以随机初始权重开始，因此该示例的结果在每次运行时都会略有不同。

双层（即，一个隐藏层）前馈神经网络可以学习任何输入-输出关系，前提是隐藏层中有足够的神经元。非输出层称为隐含层。

对于此示例，我们将尝试具有 10 个神经元的单隐藏层。一般情况下，问题越困难，需要的神经元和层就越多。问题越简单，需要的神经元就越少。

输入和输出的大小为 0，因为网络尚未配置成与我们的输入数据和目标数据相匹配。这将在训练网络时进行。

```
net = patternnet(10);
view(net)
```

现在网络已准备就绪，可以开始训练。样本自动分为训练集、验证集和测试集。训练集用于对网络进行训练。只要网络针对验证集持续改进，训练就会继续。测试集提供完全独立的网络准确度测量。

神经网络训练工具显示正在接受训练的网络和用于训练该网络的算法。它还显示训练过程中的训练状态以及停止训练的条件（以绿色突出显示）。

底部的按钮用于打开有用的绘图，这些图可以在训练期间和训练后打开。算法名称和绘图按钮旁边的链接可打开有关这些主题的文档。

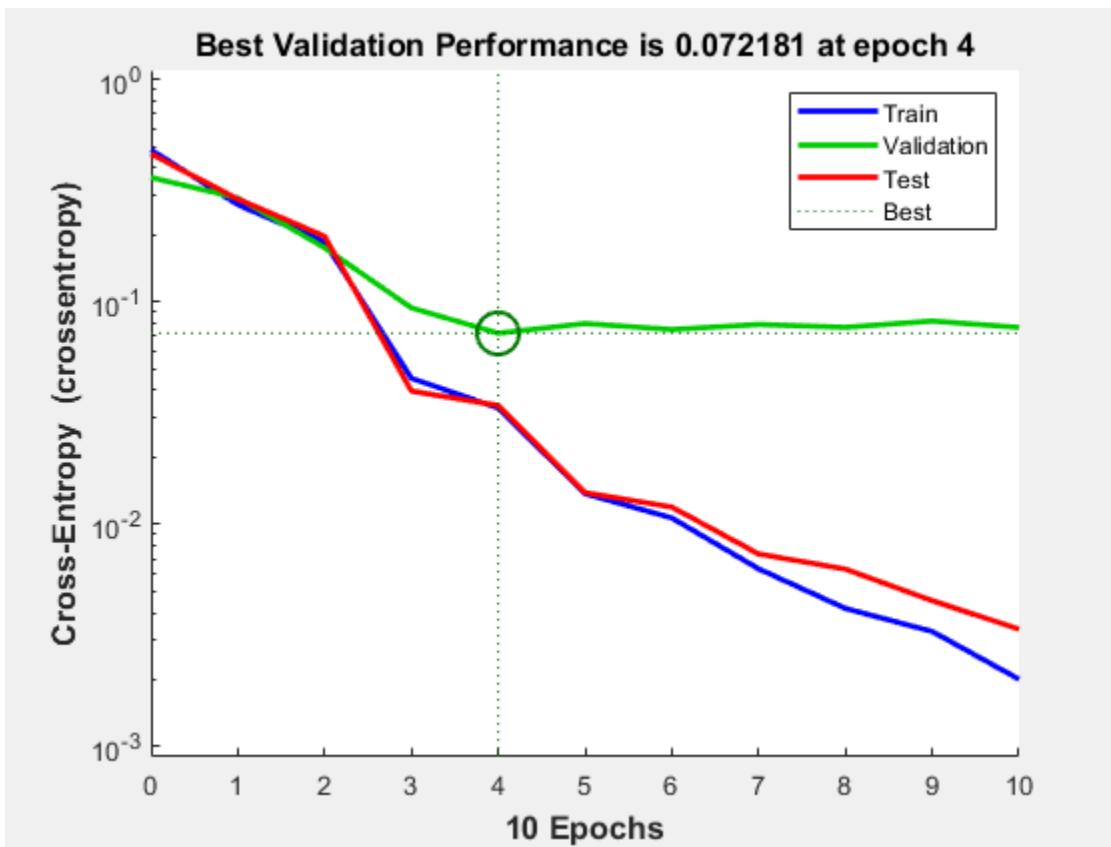
```
[net,tr] = train(net,x,t);
nntraintool
nntraintool('close')
```

要查看在训练过程中网络性能的改善情况，请点击训练工具中的 Performance 按钮，或调用 PLOTPERFORM。

性能以均方误差衡量，并以对数刻度显示。随着网络训练的加深，均方误差迅速降低。

绘图会显示训练集、验证集和测试集的性能。

```
plotperform(tr)
```



### 测试网络

现在可以相对于测试样本测量经过训练的神经网络的均方误差。这将使我们能够了解网络在应用于真实数据时表现如何。

网络输出的范围为 0 到 1，因此我们可以使用 **vec2ind** 函数根据每个输出向量中最高元素的位置来获取类索引。

```
testX = x(:,tr.testInd);
testT = t(:,tr.testInd);

testY = net(testX);
testIndices = vec2ind(testY)

testIndices = 1×27
1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3
```

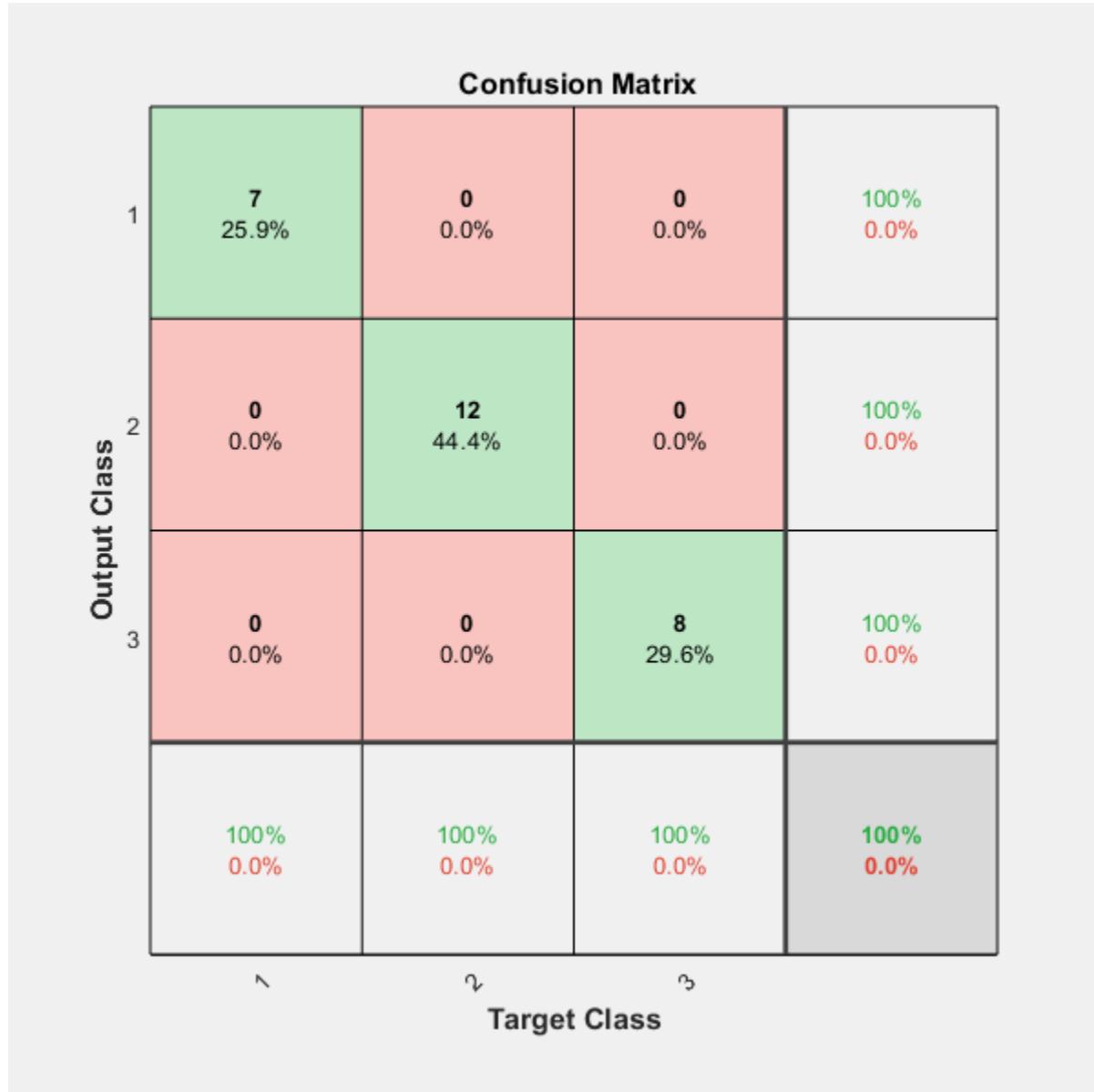
衡量神经网络数据拟合程度的另一个方法是混淆矩阵图。下面绘制了所有样本的混淆矩阵图。

该混淆矩阵显示了正确和错误分类的百分比。正确分类表示为矩阵对角线上的绿色方块。错误分类表示为红色方块。

如果网络已学会正确分类，则红色方块中的百分比应该非常小，表示几乎没有错误分类。

如果不是这样，则建议进一步进行训练，或训练具有更多隐藏神经元的网络。

```
plotconfusion(testT,testY)
```



以下是正确和错误分类的总体百分比。

```
[c,cm] = confusion(testT,testY)
```

c = 0

cm = 3×3

```
7  0  0
0  12 0
0  0  8
```

```
fprintf('Percentage Correct Classification : %f%%\n', 100*(1-c));
```

```
Percentage Correct Classification : 100.000000%
```

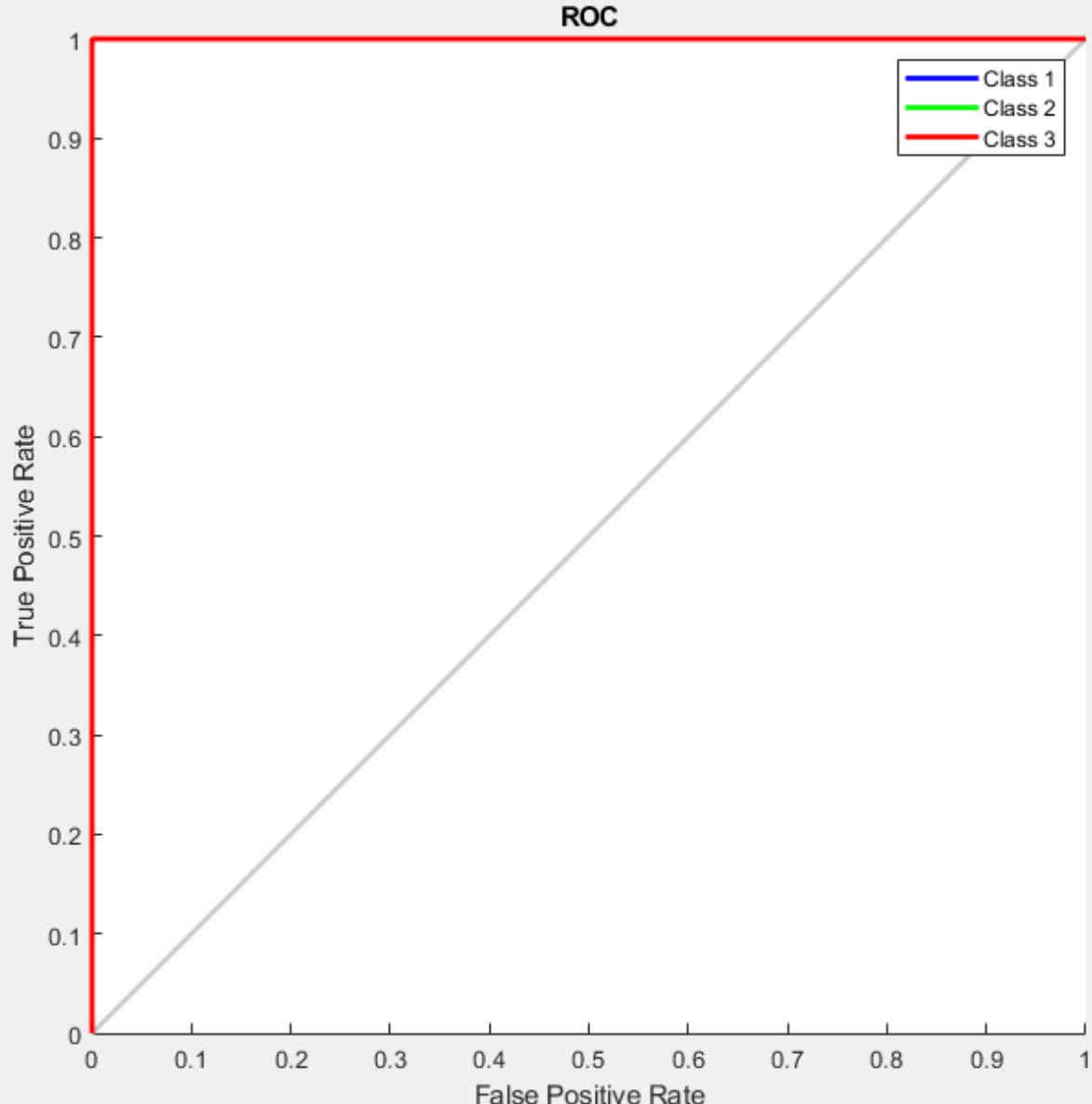
```
fprintf('Percentage Incorrect Classification : %f%%\n', 100*c);
```

```
Percentage Incorrect Classification : 0.000000%
```

衡量神经网络数据拟合程度的第三个方法是受试者工作特征图。该图可显示随着输出阈值从 0 变为 1，假正率和真正率之间的相关性。

线条越偏向左上方，达到高的真正率所需接受的假正数越少。最佳分类器是线条从左下角到左上角再到右上角，或接近于该模式。

```
plotroc(testT,testY)
```



此示例说明了如何设计一个根据每种葡萄酒的特性将葡萄酒分类到三个酒庄的神经网络。

请浏览其他示例和文档，以便更深入地了解神经网络及其应用。

## 癌症检测

此示例说明如何训练一个神经网络来使用蛋白质表达谱上的质谱数据检测癌症。

### 简介

血清蛋白质组模式诊断可用于区分疾病患者和非疾病患者的样本。表达谱模式使用表面增强激光解吸和电离 (SELDI) 蛋白质质谱生成。此技术有可能改进癌症病理学临床诊断测试。

### 问题：癌症检测

目标是构建一个可根据质谱数据区分癌症患者和对照患者的分类器。

此示例中遵循的方法是选择一组精简测量值或“特征”，这些测量值或“特征”可用于通过分类器区分癌症患者和对照患者。这些特征是特定质量/电荷值下的离子强度水平。

### 整理数据

此示例中使用的数据在文件 `ovarian_dataset.mat` 中提供，来自 FDA-NCI 临床蛋白质组学计划数据库。有关此数据集的详细说明，请参阅 [1] 和 [2]。

按照“Batch Processing of Spectra Using Sequential and Parallel Computing”(Bioinformatics Toolbox)中的步骤创建数据文件 `OvarianCancerQAQCdataset.mat`。新文件包含变量 `Y`、`MZ` 和 `grp`。

`Y` 中的每列表示从一名患者身上获取的测量值。`Y` 中有 216 列，表示有 216 个患者，其中 121 个是卵巢癌患者，95 个是非癌症患者。

`Y` 中的每行表示 `MZ` 中指示的特定质量-电荷值下的离子强度水平。`MZ` 中有 15000 个质量-电荷值，`Y` 中的每行代表在特定质量-电荷值下患者的离子强度水平。

变量 `grp` 保存关于哪些样本表示癌症患者以及哪些样本表示非癌症患者的索引信息。

### 关键特征排名

此任务是一个典型的分类问题，其中特征的数量远大于观测值的数量，但是单个特征即可实现正确分类。因此，目标是找到一个分类器，该分类器应适当学习如何加权多个特征，同时生成不会过拟合的广义映射。

找到重要特征的一个简单方法是，假设每个 M/Z 值都是独立的，并计算双向 t 检验。`rankfeatures` 返回最重要的 M/Z 值的索引，例如，按检验统计量绝对值排名的 100 个索引。

加载 `OvarianCancerQAQCdataset.mat` 并使用 `rankfeatures` (Bioinformatics Toolbox) 对特征进行排名，以选择 100 个排名最高的测量值作为输入 `x`。

```
ind = rankfeatures(Y,grp,'Criterion','ttest','NumberofIndices',100);
x = Y(ind,:);
```

为两个类定义目标 `t`，如下所示：

```
t = double(strcmp('Cancer',grp));
t = [t; 1-t];
```

上面列出的脚本和示例中的预处理步骤旨在演示一组有代表性的可能预处理和特征选择过程。使用不同的步骤或参数会产生不同并且可能更好的结果。

```
[x,t] = ovarian_dataset;
whos x t
```

Name	Size	Bytes	Class	Attributes
t	2x216	3456	double	
x	100x216	172800	double	

x 中的每一列表示 216 个不同患者中的一个。

x 中的每行表示每个患者在 100 个特定质量-电荷值之下的离子强度水平。

变量 t 具有两行，包含 216 个值，其中每个值为 [1;0]（表示癌症患者）或 [0;1]（表示非癌症患者）。

### 使用前馈神经网络进行分类

现在您已确定一些重要特征，您可以使用这些信息对癌症样本和正常样本进行分类。

由于神经网络以随机初始权重进行初始化，因此每次运行该示例来训练网络后的结果都略有不同。为了避免这种随机性，请设置随机种子以便每次都重现相同的结果。但设置随机种子对于您自己的应用情形并不是必需的。

```
setdemorandstream(672880951)
```

创建并训练具有 5 个隐藏层神经元的单隐藏层前馈神经网络。输入样本和目标样本自动分为训练集、验证集和测试集。训练集用于对网络进行训练。只要网络针对验证集持续改进，训练就会继续。测试集提供独立的网络准确度测量。

输入和输出的大小为 0，因为网络尚未配置成与输入数据和目标数据相匹配。在训练网络时会进行此配置。

```
net = patternnet(5);
view(net)
```

现在网络已准备就绪，可以开始训练。样本自动分为训练集、验证集和测试集。训练集用于对网络进行训练。只要网络针对验证集持续改进，训练就会继续。测试集提供独立的网络准确度测量。

神经网络训练工具显示正在接受训练的网络和用于训练该网络的算法。它还显示训练过程中的训练状态以及停止训练的条件（以绿色突出显示）。

底部的按钮用于打开有用的绘图，这些图可以在训练期间和训练后打开。算法名称和绘图按钮旁边的链接可打开有关这些主题的文档。

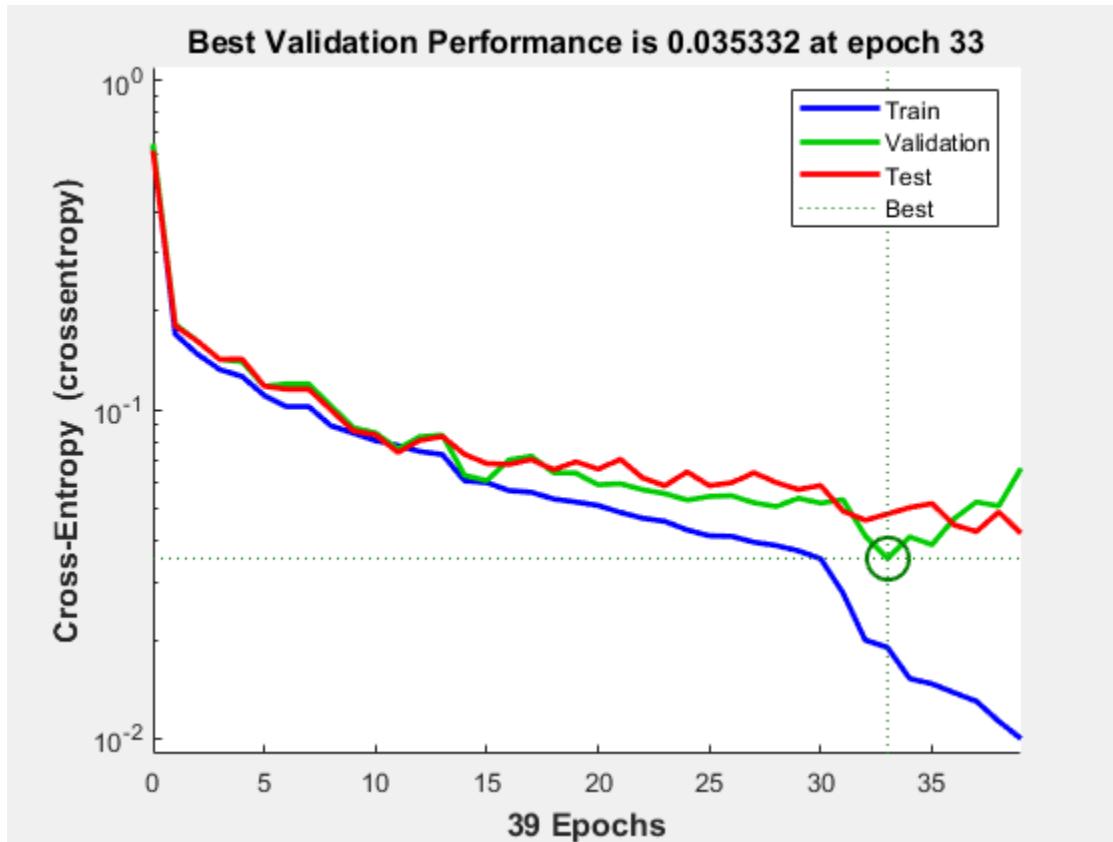
```
[net,tr] = train(net,x,t);
```

要查看在训练过程中网络性能的改善情况，请点击训练工具中的 Performance 按钮，或使用 **plotperform** 函数。

性能以均方误差衡量，并以对数刻度显示。随着网络训练的加深，均方误差迅速降低。

绘图会显示训练集、验证集和测试集的性能。

```
plotperform(tr)
```



现在可以使用我们从主数据集划分出来的测试样本测试经过训练的神经网络。在训练中没有以任何方式使用过测试数据，因此测试数据是可用来测试网络的“样本外”数据集。这样可以估计出当使用真实数据进行测试时，网络的表现如何。

网络输出的范围为 0-1。对输出应用阈值以获得 1 和 0，分别表示癌症患者和非癌症患者。

```
testX = x(:,tr.testInd);
testT = t(:,tr.testInd);

testY = net(testX);
testClasses = testY > 0.5

testClasses = 2x32 logical array
```

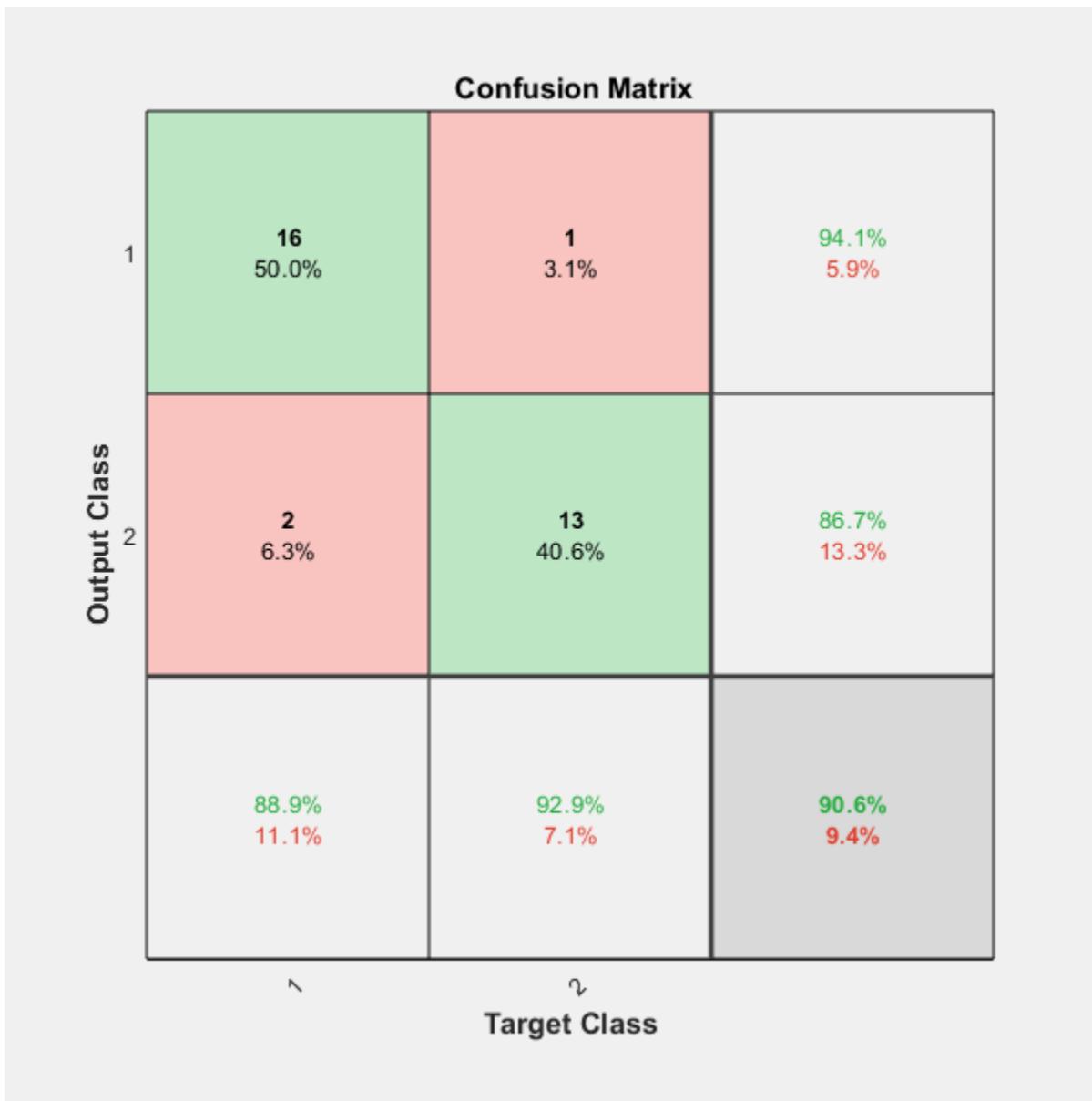
衡量神经网络数据拟合程度的一个方法是混淆矩阵图。

该混淆矩阵显示了正确和错误分类的百分比。正确分类表示为矩阵对角线上的绿色方块。红色方块表示错误分类。

如果网络是准确的，则红色方块中的百分比应该很小，表示几乎没有错误分类。

如果网络不准确，则您可以尝试训练更长时间，或者训练具有更多隐藏神经元的网络。

```
plotconfusion(testT,testY)
```



以下是正确和错误分类的总体百分比。

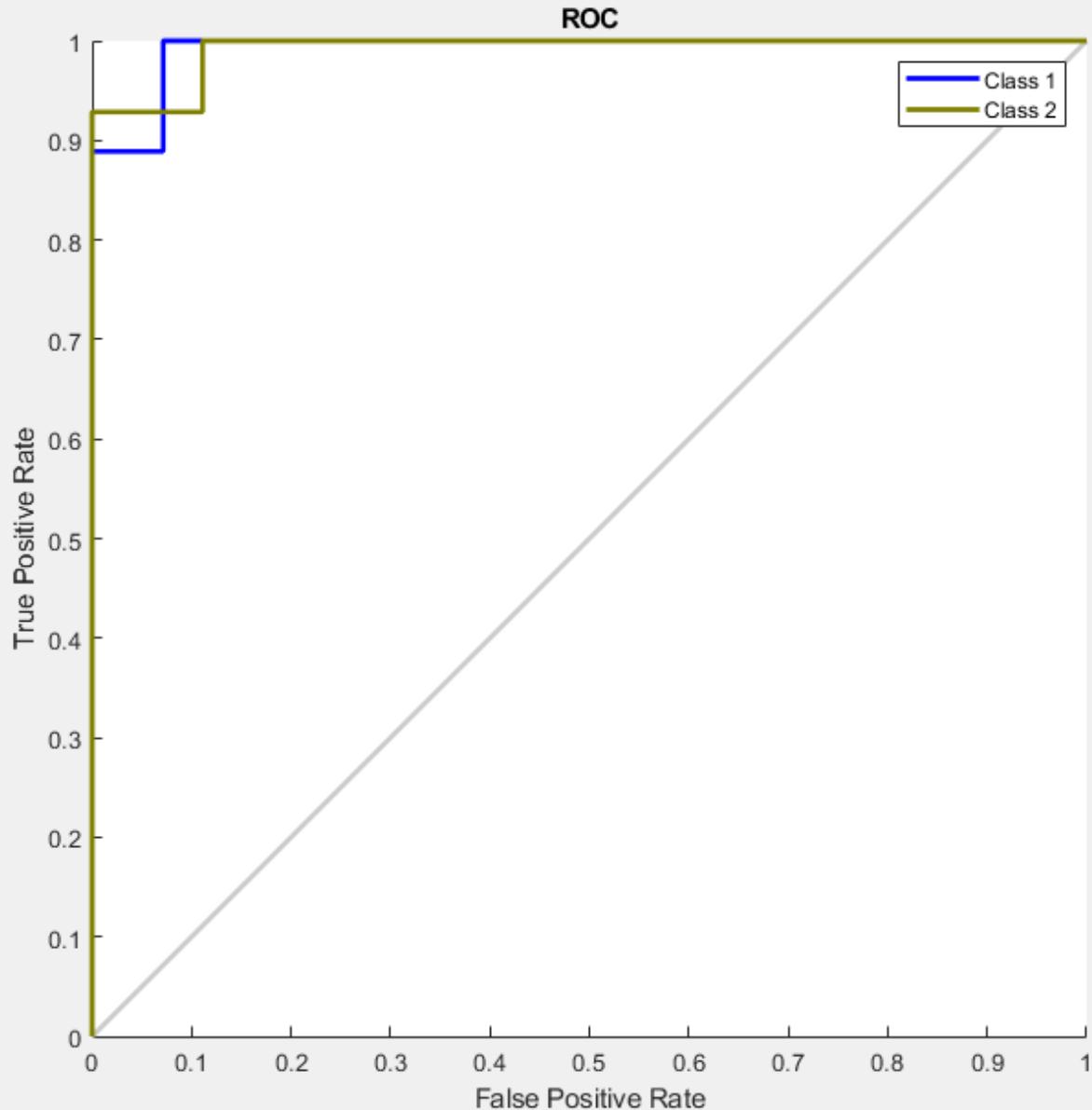
```
[c,cm] = confusion(testT,testY);
fprintf('Percentage Correct Classification : %f%%\n', 100*(1-c));
Percentage Correct Classification : 90.625000%
fprintf('Percentage Incorrect Classification : %f%%\n', 100*c);
Percentage Incorrect Classification : 9.375000%
```

衡量神经网络数据拟合程度的另一个方法是受试者工作特征图。该图显示随着输出阈值从 0 变为 1，假正率和真正率之间的相关性。

线条越偏向左上方，达到高的真正率所需接受的假正数越少。最佳分类器是线条从左下角到左上角再到右上角，或接近于该模式。

第 1 类表示癌症患者，第 2 类表示非癌症患者。

```
plotroc(testT,testY)
```



此示例演示了如何将神经网络用作癌症检测的分类器。为了提高分类器性能，您还可以尝试使用主成分分析等技术来降低用于神经网络训练的数据的维度。

## 参考资料

- [1] T.P. Conrads, et al., "High-resolution serum proteomic features for ovarian detection", *Endocrine-Related Cancer*, 11, 2004, pp. 163-178.
- [2] E.F. Petricoin, et al., "Use of proteomic patterns in serum to identify ovarian cancer", *Lancet*, 359(9306), 2002, pp. 572-577.

## 字符识别

此示例说明如何训练神经网络以执行简单的字符识别。

### 定义问题

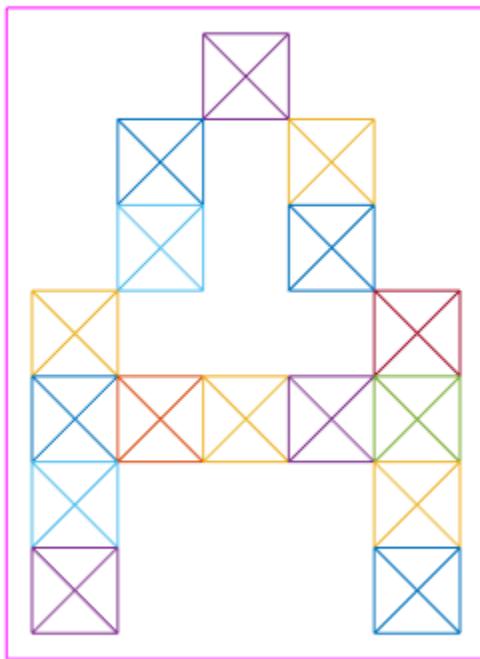
脚本 **prprob** 定义了一个包含 26 列的矩阵  $X$ ，每列对应一个字母。每列有 35 个值，值可能是 1，也可能是 0。每列（包含 35 个值）定义一个字母的  $5 \times 7$  位图。

矩阵  $T$  是一个  $26 \times 26$  的单位矩阵，它将 26 个输入向量映射到 26 个类。

```
[X,T] = prprob;
```

以下命令将第一个字母 A 绘制为一个位图。

```
plotchar(X(:,1))
```



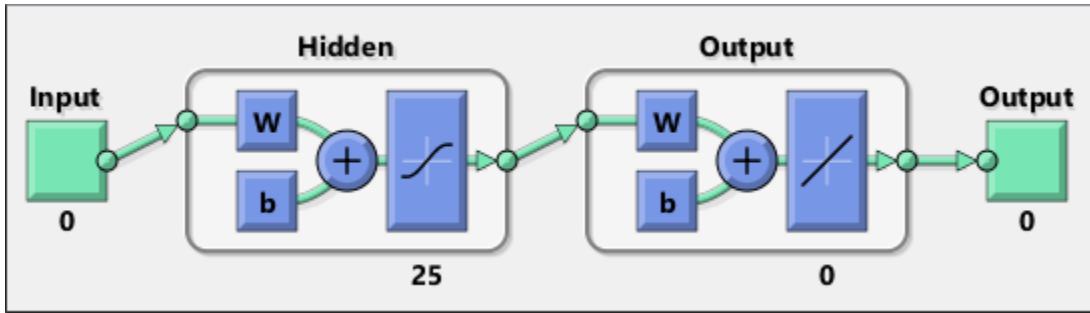
### 创建第一个神经网络

为求解此问题，我们将使用针对模式识别建立的具有 25 个隐藏神经元的前馈神经网络。

由于神经网络以随机初始权重进行初始化，因此每次运行该示例进行训练后的结果都略有不同。为了避免这种随机性，请设置随机种子以便每次都重现相同的结果。这对于您自己的应用情形不是必需的。

```
setdemorandstream(pi);
```

```
net1 = feedforwardnet(25);
view(net1)
```



### 训练第一个神经网络

函数 **train** 将数据划分为训练集、验证集和测试集。训练集用于更新网络，验证集用于在网络过拟合训练数据之前停止网络，从而保持良好的泛化。测试集用作完全独立的测量手段，用于衡量网络针对新样本的预期表现。

当网络针对训练集或验证集不再可能有改善时，训练停止。

```
net1.divideFcn = "";  
net1 = train(net1,X,T,nnMATLAB);
```

Computing Resources:  
MATLAB on GLNXA64

### 训练第二个神经网络

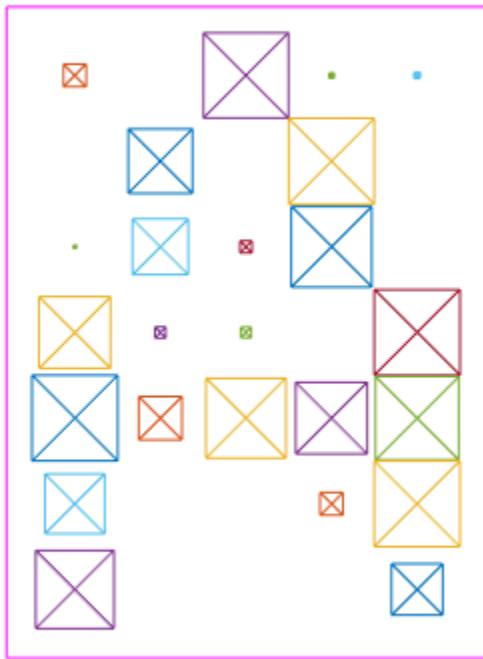
我们希望网络不仅可以识别形状标准的字母，还可以识别含噪的字母。因此，我们将尝试针对含噪数据训练第二个网络，并将其泛化能力与第一个网络进行比较。

以下命令为每个字母 Xn 创建 30 个含噪副本。值由 **min** 和 **max** 限制在 0 和 1 之间。还定义了相应的目标 Tn。

```
numNoise = 30;  
Xn = min(max(repmat(X,1,numNoise)+randn(35,26*numNoise)*0.2,0),1);  
Tn = repmat(T,1,numNoise);
```

以下是 A 的含噪版本。

```
figure  
plotchar(Xn(:,1))
```



以下命令将创建并训练第二个网络。

```
net2 = feedforwardnet(25);
net2 = train(net2,Xn,Tn,nnMATLAB);
```

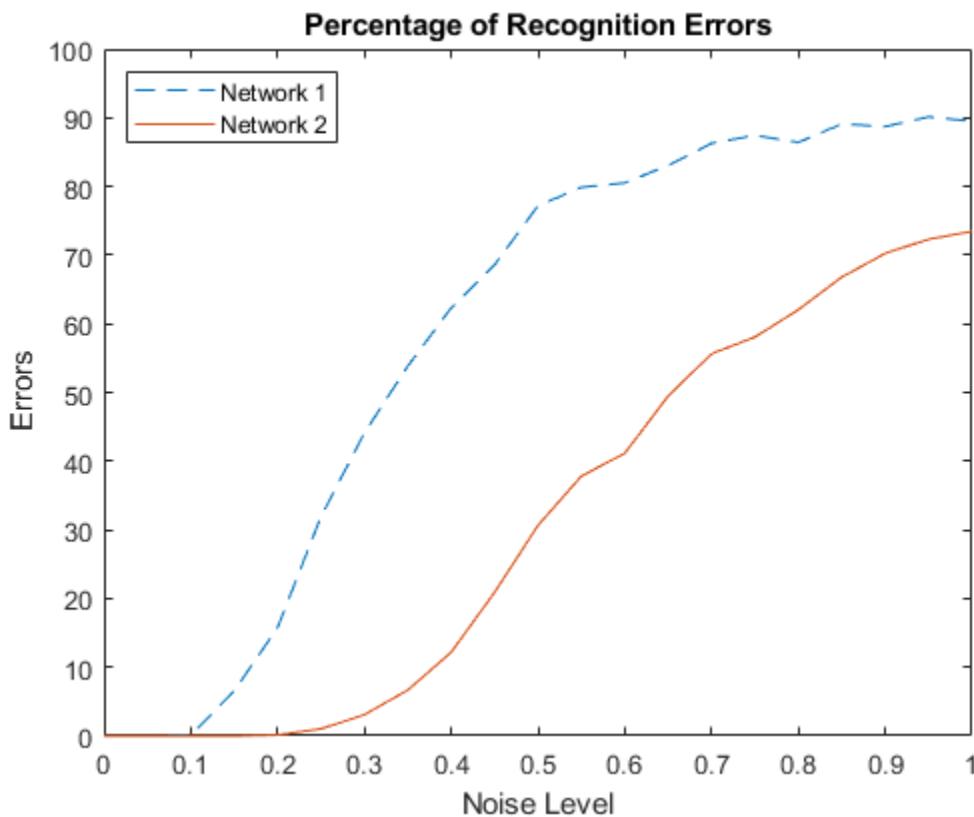
Computing Resources:  
MATLAB on GLNXA64

### 测试两个神经网络

```
noiseLevels = 0:.05:1;
numLevels = length(noiseLevels);
percError1 = zeros(1,numLevels);
percError2 = zeros(1,numLevels);
for i = 1:numLevels
    Xtest = min(max(repmat(X,1,numNoise)+randn(35,26*numNoise)*noiseLevels(i),0),1);
    Y1 = net1(Xtest);
    percError1(i) = sum(sum(abs(Tn-compet(Y1))))/(26*numNoise*2);
    Y2 = net2(Xtest);
    percError2(i) = sum(sum(abs(Tn-compet(Y2))))/(26*numNoise*2);
end

figure
plot(noiseLevels,percError1*100,'-.',noiseLevels,percError2*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
```

```
ylabel('Errors');
legend('Network 1','Network 2','Location','NorthWest')
```



由于存在噪声，在无噪声情况下训练的网络 1 的错误数多于在有噪声情况下训练的网络 2。

## 训练堆叠自编码器进行图像分类

此示例说明如何训练堆叠自编码器以对数字图像进行分类。

具有多个隐含层的神经网络可用于处理复杂数据（例如图像）的分类问题。每个层都可以学习不同抽象级别的特征。然而，在实际工作中，训练具有多个隐含层的神经网络可能会很困难。

一种有效训练具有多个层的神经网络的方法是一次训练一个层。为此，您可以为每个所需的隐含层训练一种称为自编码器的特殊类型的网络。

此示例说明如何训练具有两个隐含层的神经网络以对图像中的数字进行分类。首先，使用自编码器以无监督方式单独训练各隐含层。然后训练最终 softmax 层，并将这些层连接在一起形成堆叠网络，该网络最后以有监督方式进行训练。

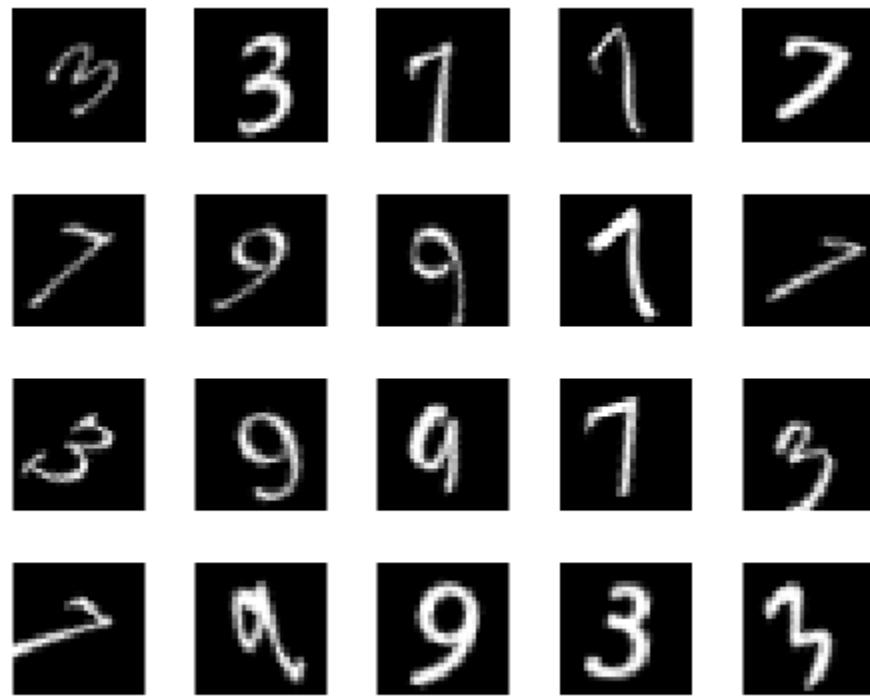
### 数据集

此示例始终使用合成数据进行训练和测试。已通过对使用不同字体创建的数字图像应用随机仿射变换来生成合成图像。

每个数字图像为  $28 \times 28$  像素，共有 5000 个训练样本。您可以加载训练数据，并查看其中一些图像。

```
% Load the training data into memory
[xTrainImages,tTrain] = digitTrainCellArrayData;

% Display some of the training images
clf
for i = 1:20
    subplot(4,5,i);
    imshow(xTrainImages{i});
end
```



图像的标签存储在一个  $10 \times 5000$  矩阵中，其中每列都有一个元素为 1，指示该数字所属的类，该列中的所有其他元素为 0。请注意，如果第十个元素是 1，则数字图像是零。

### 训练第一个自编码器

首先在不使用标签的情况下基于训练数据训练稀疏自编码器。

自编码器是一种神经网络，该网络会尝试在其输出端复制其输入。因此，其输入的大小将与其输出的大小相同。当隐藏层中的神经元数量小于输入的大小时，自编码器将学习输入的压缩表示。

神经网络在训练前具有随机初始化的权重。因此，每次训练的结果都不同。为避免此行为，请显式设置随机数生成器种子。

```
rng('default')
```

设置自编码器的隐含层的大小。对于要训练的自编码器，最好使隐含层的大小小于输入大小。

```
hiddenSize1 = 100;
```

您将训练的自编码器的类型是稀疏自编码器。该自编码器使用正则项来学习第一层中的稀疏表示。您可以设置各种参数来控制这些正则项的影响：

- **L2WeightRegularization** 控制 L2 正则项对网络权重（而不是偏置）的影响。这通常应该非常小。
- **SparsityRegularization** 控制稀疏正则项的影响，该正则项会尝试对隐含层的输出的稀疏性施加约束。请注意，这与将稀疏正则项应用于权重不同。
- **SparsityProportion** 是稀疏正则项的参数。它控制隐含层的输出的稀疏性。较低的 **SparsityProportion** 值通常导致只为少数训练样本提供高输出，从而使隐藏层中的每个神经元“专

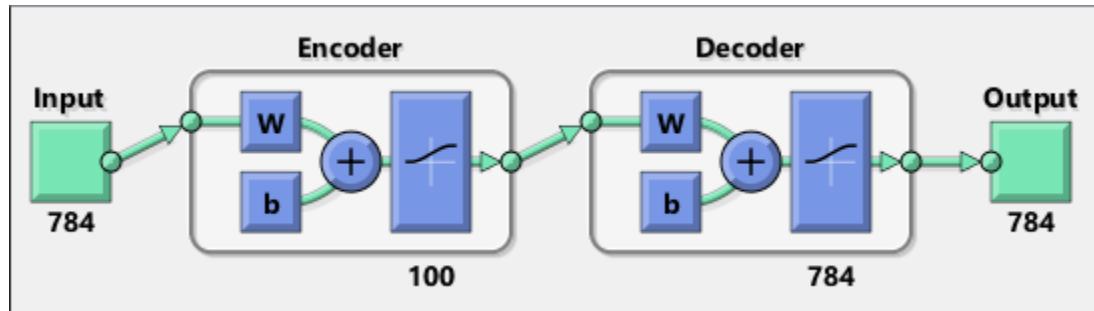
门化”。例如，如果 **SparsityProportion** 设置为 0.1，这相当于说隐藏层中的每个神经元针对训练样本的平均输出值应该为 0.1。此值必须介于 0 和 1 之间。理想值因问题的性质而异。

现在训练自编码器，指定上述正则项的值。

```
autoenc1 = trainAutoencoder(xTrainImages,hiddenSize1, ...
    'MaxEpochs',400, ...
    'L2WeightRegularization',0.004, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.15, ...
    'ScaleData', false);
```

您可以查看自编码器的图。自编码器由一个编码器和一个解码器组成。编码器将输入映射为隐含表示，解码器则尝试进行逆映射以重新构造原始输入。

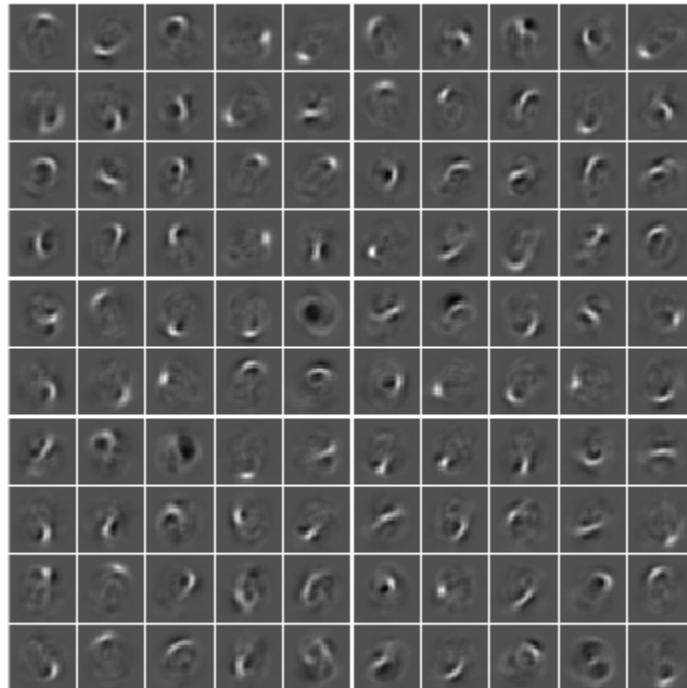
```
view(autoenc1)
```



### 可视化第一个自编码器的权重

自编码器的编码器部分所学习的映射可用于从数据中提取特征。编码器中的每个神经元都具有一个与之相关联的权重向量，该向量将进行相应调整以响应特定可视化特征。您可以查看这些特征的表示。

```
figure()
plotWeights(autoenc1);
```



您可以看到，自编码器学习的特征代表了数字图像中的弯曲和笔划图案。

自编码器的隐含层的 100 维输出是输入的压缩版本，它汇总了对上面可视化的特征的响应。基于从训练数据中提取的一组向量训练下一个自编码器。首先，必须使用经过训练的自编码器中的编码器生成特征。

```
feat1 = encode(autoenc1,xTrainImages);
```

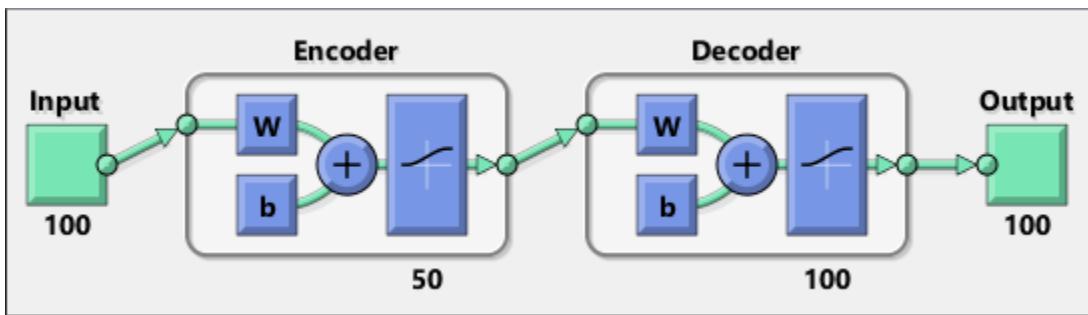
### 训练第二个自编码器

训练完第一个自编码器后，您需要以相似的方式训练第二个自编码器。主要区别在于您将使用从第一个自编码器生成的特征作为第二个自编码器中的训练数据。此外，您还需要将隐含表示的大小减小到 50，以便第二个自编码器中的编码器学习输入数据的更小表示。

```
hiddenSize2 = 50;
autoenc2 = trainAutoencoder(feat1,hiddenSize2, ...
    'MaxEpochs',100, ...
    'L2WeightRegularization',0.002, ...
    'SparsityRegularization',4, ...
    'SparsityProportion',0.1, ...
    'ScaleData', false);
```

同样，您可以使用 `view` 函数查看自编码器的图。

```
view(autoenc2)
```



您可以将前一组特征传递给第二个自编码器中的编码器，以此提取第二组特征。

```
feat2 = encode(autoenc2,feat1);
```

训练数据中的原始向量具有 784 个维度。原始数据通过第一个编码器后，维度减小到 100 维。应用第二个编码器后，维度进一步减小到 50 维。您现在可以训练最终层，以将这些 50 维向量分类为不同的数字类。

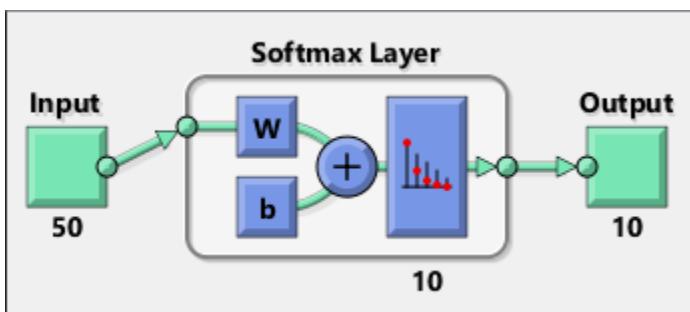
### 训练最终 softmax 层

训练 softmax 层以对 50 维特征向量进行分类。与自编码器不同，您将使用训练数据的标签以有监督方式训练 softmax 层。

```
softnet = trainSoftmaxLayer(feat2,tTrain,'MaxEpochs',400);
```

您可以使用 `view` 函数查看 softmax 层的图。

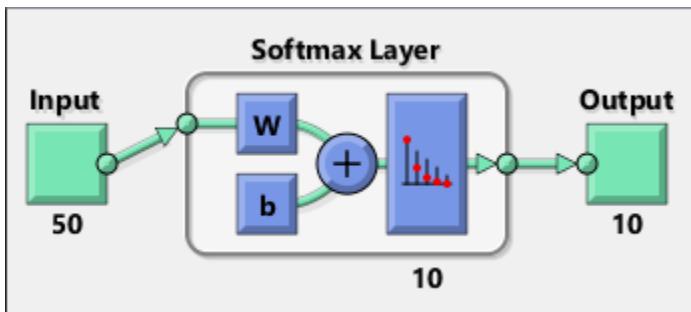
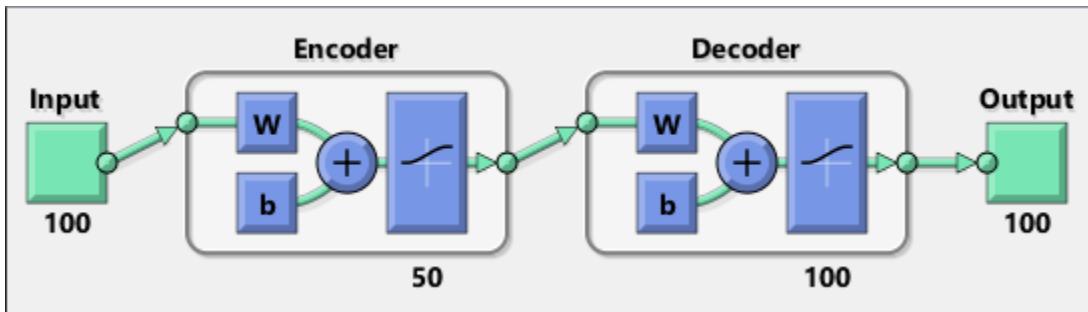
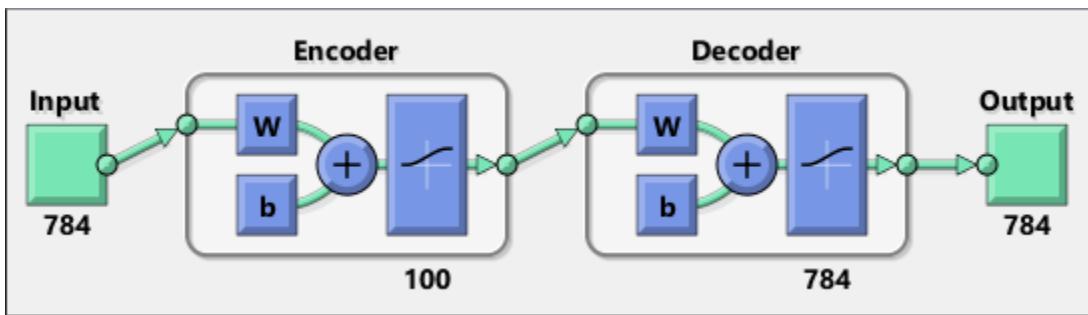
```
view(softnet)
```



### 形成堆叠神经网络

您已单独训练了组成堆叠神经网络的三个网络。现在，您可以查看已经过训练的三个神经网络。它们是 `autoenc1`、`autoenc2` 和 `softnet`。

```
view(autoenc1)
view(autoenc2)
view(softnet)
```

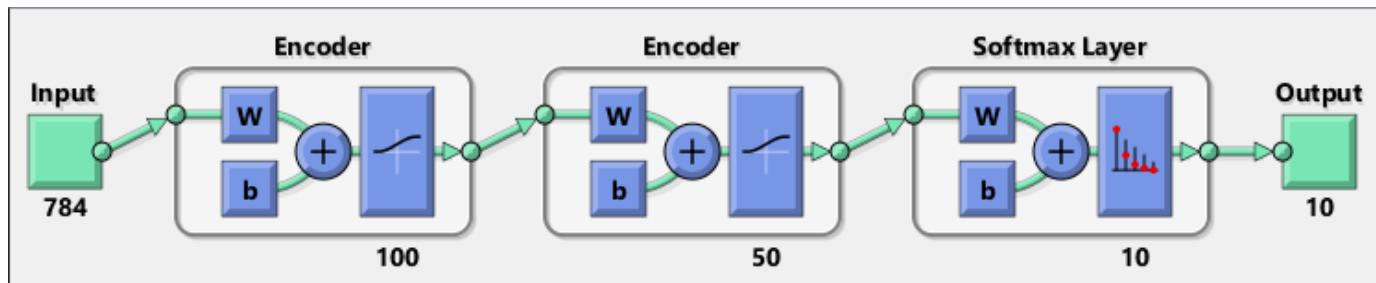


如前所述，自编码器中的编码器已用于提取特征。您可以将自编码器中的编码器与 softmax 层堆叠在一起，以形成用于分类的堆叠网络。

```
stackednet = stack(autoenc1,autoenc2,softnet);
```

您可以使用 `view` 函数查看堆叠网络的图。该网络由自编码器中的编码器和 softmax 层构成。

```
view(stackednet)
```



在搭建了完整网络之后，您可以基于测试集计算结果。要将图像用于堆叠网络，必须将测试图像重构为矩阵。这可以通过先堆叠图像的各列以形成向量，然后根据这些向量形成矩阵来完成。

```
% Get the number of pixels in each image
imageWidth = 28;
imageHeight = 28;
inputSize = imageWidth*imageHeight;

% Load the test images
[xTestImages,tTest] = digitTestCellArrayData;

% Turn the test images into vectors and put them in a matrix
xTest = zeros(inputSize,numel(xTestImages));
for i = 1:numel(xTestImages)
    xTest(:,i) = xTestImages{i}(:);
end
```

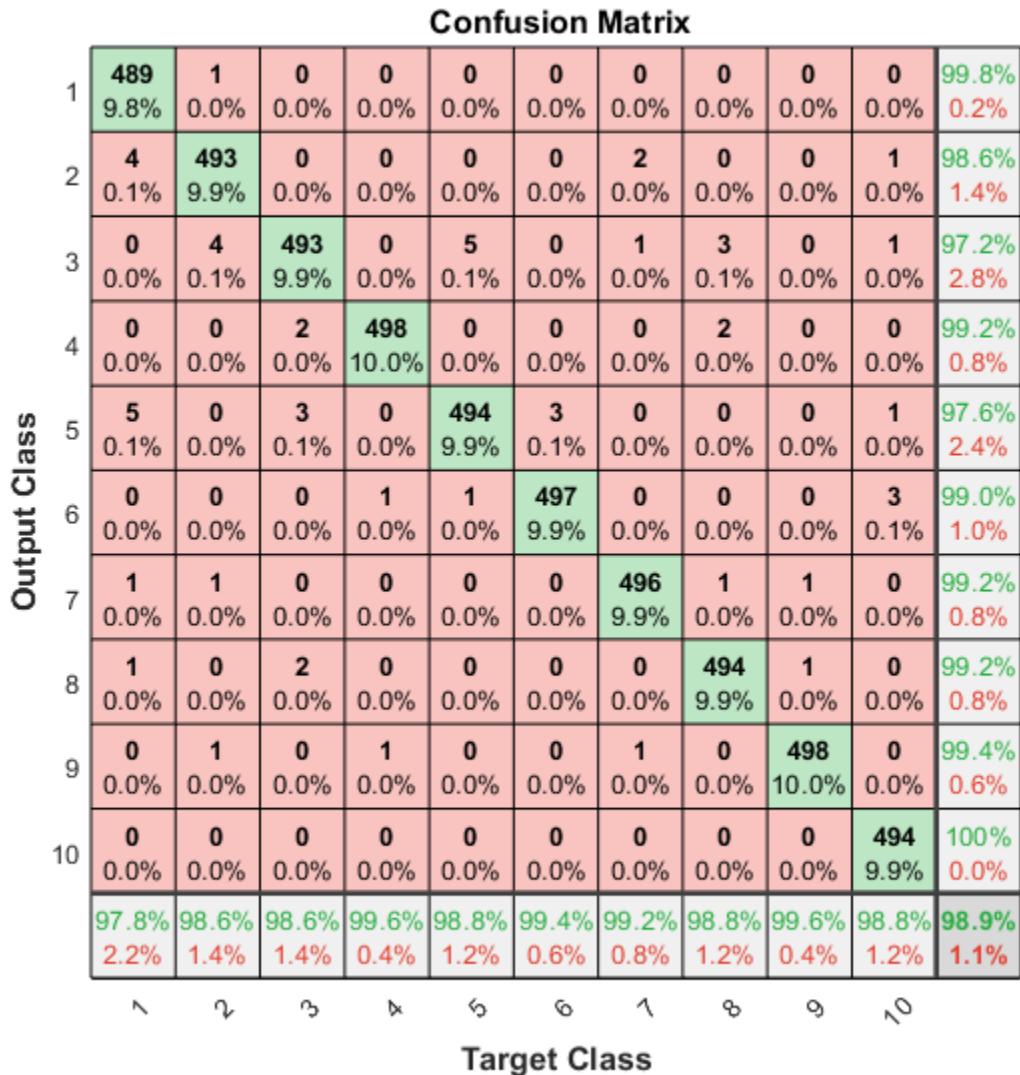
您可以使用混淆矩阵来可视化结果。矩阵右下角方块中的数字表示整体准确度。

```
y = stackednet(xTest);
plotconfusion(tTest,y);
```

Confusion Matrix												
Output Class	1	2	3	4	5	6	7	8	9	10		
	1	337 6.7%	11 0.2%	9 0.2%	39 0.8%	18 0.4%	57 1.1%	43 0.9%	14 0.3%	3 0.1%	11 0.2%	62.2% 37.8%
	2	19 0.4%	252 5.0%	50 1.0%	14 0.3%	11 0.2%	10 0.2%	35 0.7%	42 0.8%	23 0.5%	26 0.5%	52.3% 47.7%
	3	19 0.4%	39 0.8%	214 4.3%	8 0.2%	85 1.7%	2 0.0%	20 0.4%	65 1.3%	45 0.9%	6 0.1%	42.5% 57.5%
	4	2 0.0%	33 0.7%	45 0.9%	343 6.9%	21 0.4%	50 1.0%	3 0.1%	20 0.4%	71 1.4%	22 0.4%	56.2% 43.8%
	5	4 0.1%	18 0.4%	81 1.6%	22 0.4%	243 4.9%	18 0.4%	11 0.2%	40 0.8%	20 0.4%	12 0.2%	51.8% 48.2%
	6	54 1.1%	4 0.1%	1 0.0%	17 0.3%	27 0.5%	270 5.4%	0 0.0%	49 1.0%	5 0.1%	50 1.0%	56.6% 43.4%
	7	56 1.1%	68 1.4%	26 0.5%	6 0.1%	22 0.4%	4 0.1%	330 6.6%	36 0.7%	22 0.4%	5 0.1%	57.4% 42.6%
	8	1 0.0%	28 0.6%	30 0.6%	3 0.1%	22 0.4%	13 0.3%	9 0.2%	156 3.1%	18 0.4%	11 0.2%	53.6% 46.4%
	9	7 0.1%	22 0.4%	13 0.3%	33 0.7%	6 0.1%	27 0.5%	32 0.6%	13 0.3%	269 5.4%	43 0.9%	57.8% 42.2%
	10	1 0.0%	25 0.5%	31 0.6%	15 0.3%	45 0.9%	49 1.0%	17 0.3%	65 1.3%	24 0.5%	314 6.3%	53.6% 46.4%

然后使用混淆矩阵再次查看结果。

```
y = stackednet(xTest);
plotconfusion(tTest,y);
```



## 总结

此示例说明了如何训练堆叠神经网络以使用自编码器对图像中的数字进行分类。所述步骤可以应用于其他类似问题，例如对字母图像进行分类，或者对特定类别的对象的小型图像进行分类。

# 鸢尾花聚类

此示例说明自组织映射神经网络如何以拓扑方式将鸢尾花聚类为各个类，提供对花类型的深入了解以及用于进一步分析的实用工具。

## 问题：对鸢尾花进行聚类

在此示例中，我们尝试构建一个将鸢尾花聚类成多个自然类的神经网络，以使相似的类分组在一起。每朵鸢尾花都用四个特征进行描述：

- 萼片长度 (cm)
- 萼片宽度 (cm)
- 花瓣长度 (cm)
- 花瓣宽度 (cm)

这是一个聚类问题的示例，我们希望根据样本之间的相似性将样本分组到各个类。我们要创建一个神经网络，该网络不仅可以为已知输入创建类定义，还能相应地对未知输入进行分类。

## 为什么使用自组织映射神经网络？

自组织映射 (SOM) 非常擅长创建分类。而且，分类保留了关于哪些类与其他类最相似的拓扑信息。自组织映射可以创建为任何所需的详细程度级别。它们特别适合对存在于多个维度且具有复杂形状的相连特征空间的数据进行聚类。它们非常适合对鸢尾花进行聚类。

四个花属性将作为 SOM 的输入，SOM 将它们映射到二维神经元层。

## 准备数据

通过将数据组织成输入矩阵  $X$ ，为 SOM 设置聚类问题数据。

输入矩阵的每个第  $i$  列将具有四个元素，表示在一朵花上获取的四个测量值。

使用以下命令加载一个这样的数据集。

```
x = iris_dataset;
```

我们可以查看输入  $X$  的大小。

请注意， $X$  有 150 列。这些列代表 150 组鸢尾花属性。它具有四行，表示四个测量值。

```
size(x)
```

```
ans =
```

```
4 150
```

## 使用神经网络进行聚类

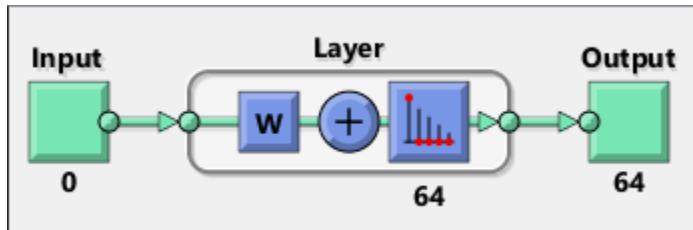
下一步是创建一个用于学习聚类的神经网络。

使用 **selforgmap** 创建自组织映射，通过选择每个层维度中的神经元数量来对样本进行所需详细程度的分类。

对于此示例，我们将尝试具有以  $8 \times 8$  六边形网格排列的 64 个神经元的二维层。通常，使用更多神经元可以获得更多细节，而使用更多维度则可对更复杂特征空间的拓扑进行建模。

输入大小为 0，因为网络尚未配置成与我们的输入数据相匹配。这将在训练网络时进行。

```
net = selforgmap([8 8]);
view(net)
```

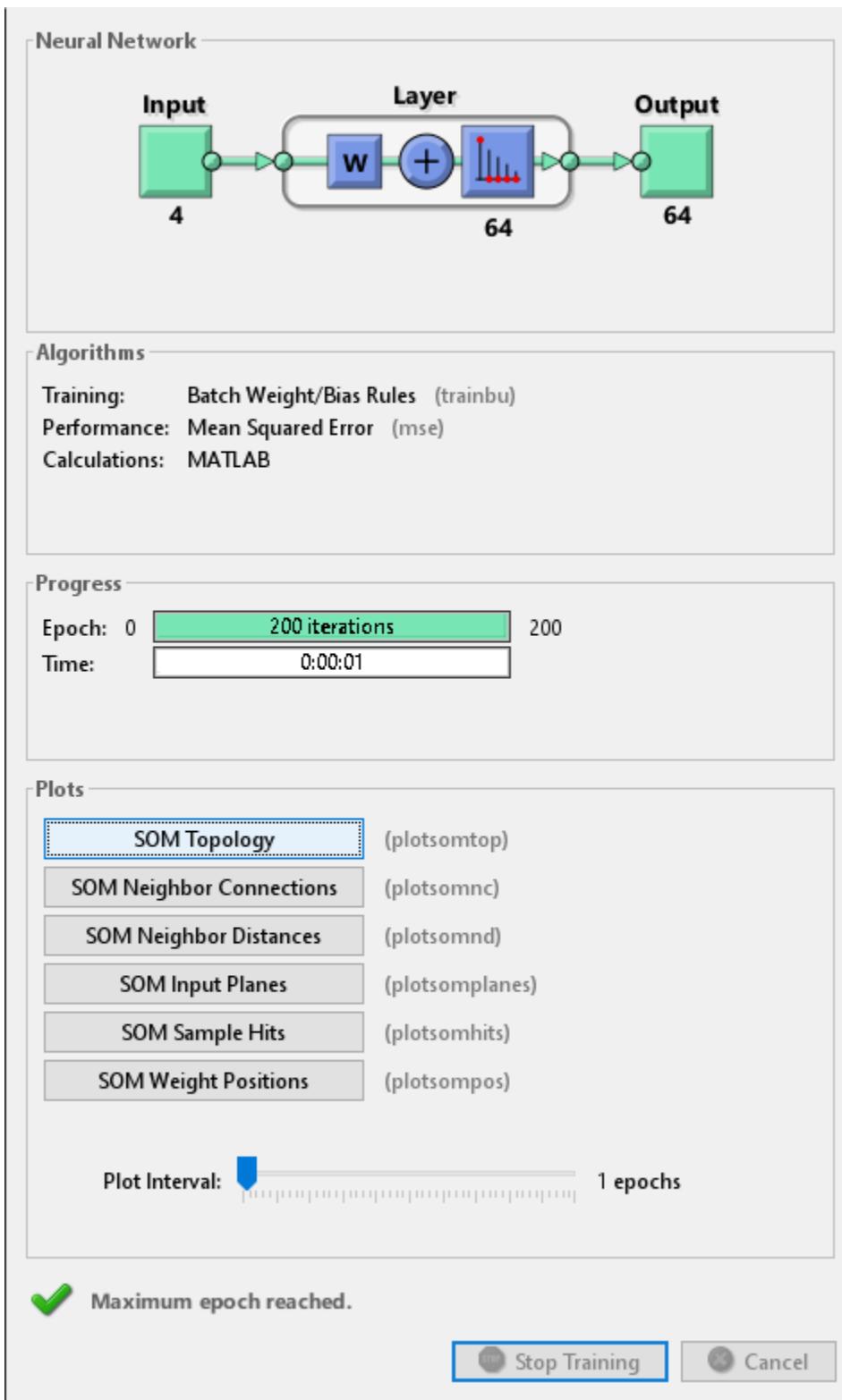


现在网络已准备好使用 **train** 进行优化。

神经网络训练工具显示正在接受训练的网络和用于训练该网络的算法。它还显示训练过程中的训练状态以及停止训练的条件（以绿色突出显示）。

底部的按钮用于打开有用的绘图，这些图可以在训练期间和训练后打开。算法名称和绘图按钮旁边的链接可打开有关这些主题的文档。

```
[net,tr] = train(net,x);
nntraintool
```



```
nntraintool('close')
```

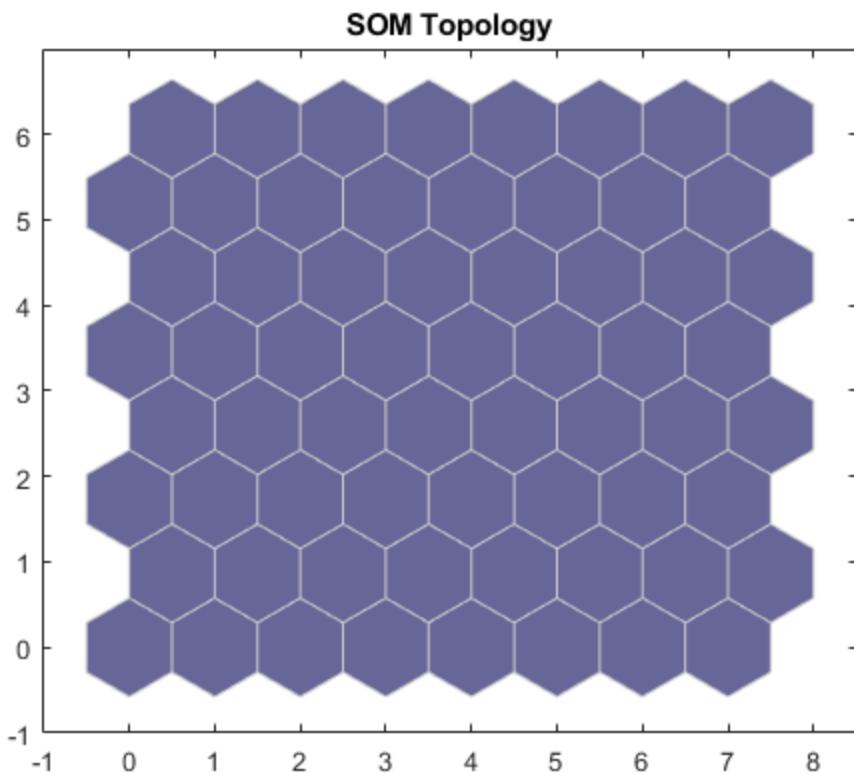
这里使用自组织映射计算每个训练输入的类向量。这些分类涵盖了已知花朵所填充的特征空间，它们现在可用于对新花朵进行相应分类。网络输出将是一个  $64 \times 150$  矩阵，其中每个第  $i$  列表示第  $i$  个输入向量（其第  $j$  个元素为 1）的第  $j$  个聚类。

函数 **vec2ind** 针对每个向量返回输出为 1 的神经元的索引。对于由 64 个神经元表示的 64 个聚类，索引值范围在 1 到 64 之间。

```
y = net(x);
cluster_index = vec2ind(y);
```

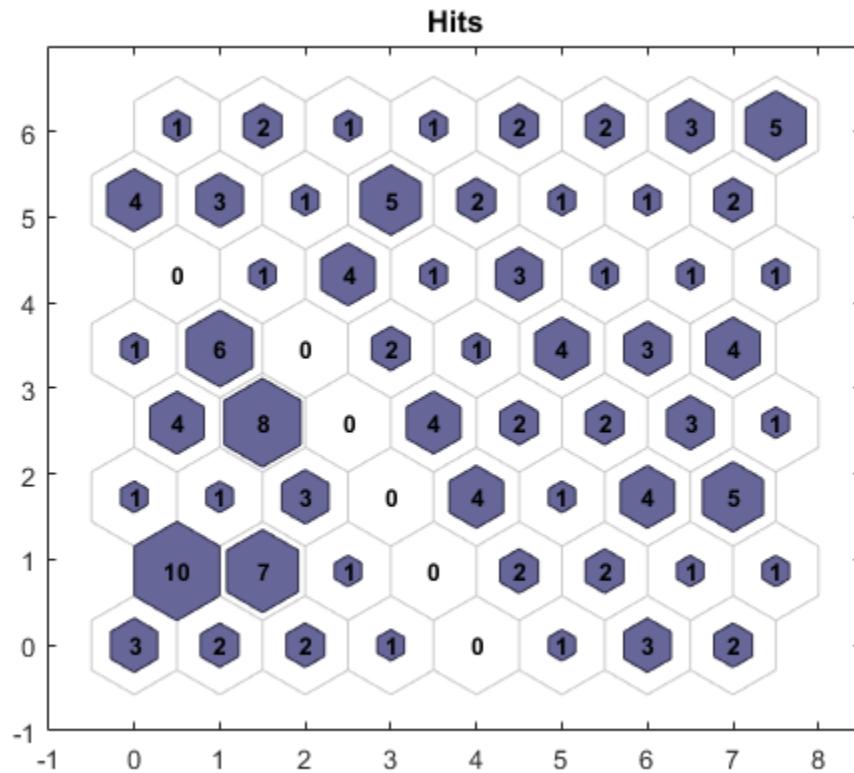
使用 **plotsomtop** 绘制位于  $8 \times 8$  六边形网格中的 64 个神经元的自组织映射拓扑。每个神经元都已经过学习，可代表不同的花类，相邻的神经元通常代表相似的类。

```
plotsomtop(net)
```



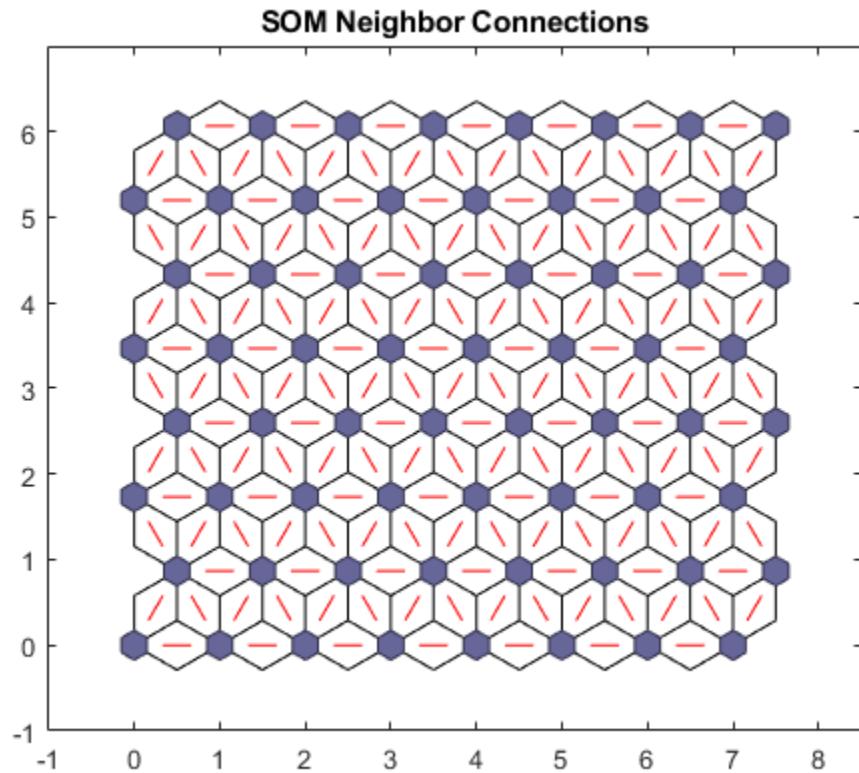
使用 **plotsomhits** 计算每朵花的类，并显示每个类中的花朵数量。具有大量命中的神经元区域所表示的类代表相似的填充度高的特征空间区域。而命中较少的区域表示填充稀疏的特征空间区域。

```
plotsomhits(net,x)
```



使用 **plotsomnc** 显示神经元邻点连接。邻点通常用于对相似样本进行分类。

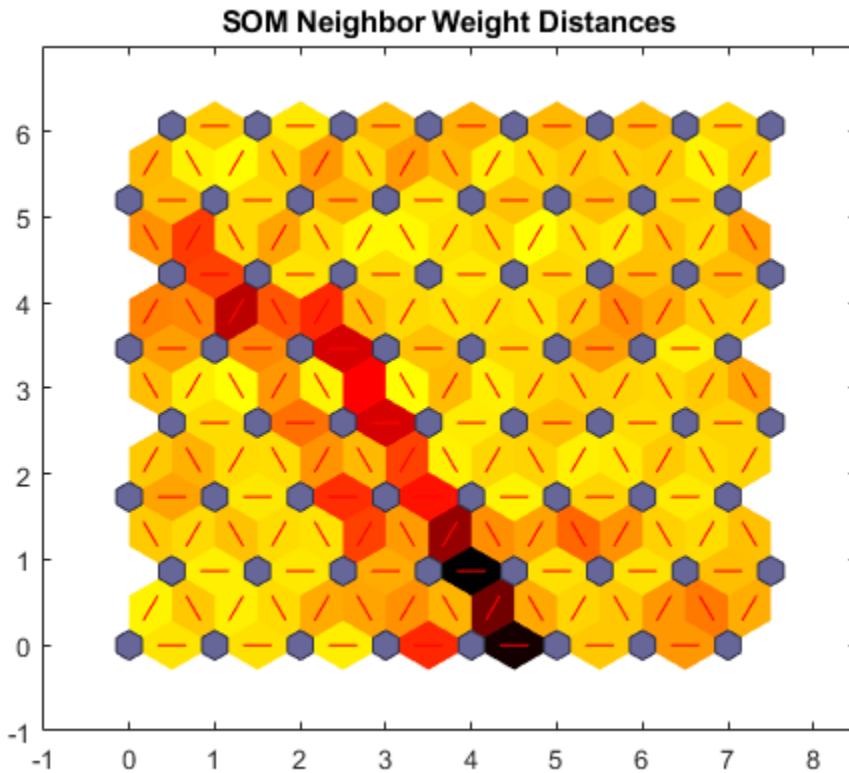
```
plotsomnc(net)
```



**plotsomnd** 显示每个神经元的类与其邻点的距离（以欧几里德距离表示）。浅色连接表示输入空间的高度连接区域。而深色连接表示的类代表相距很远且相互之间很少或没有花朵的特征空间区域。

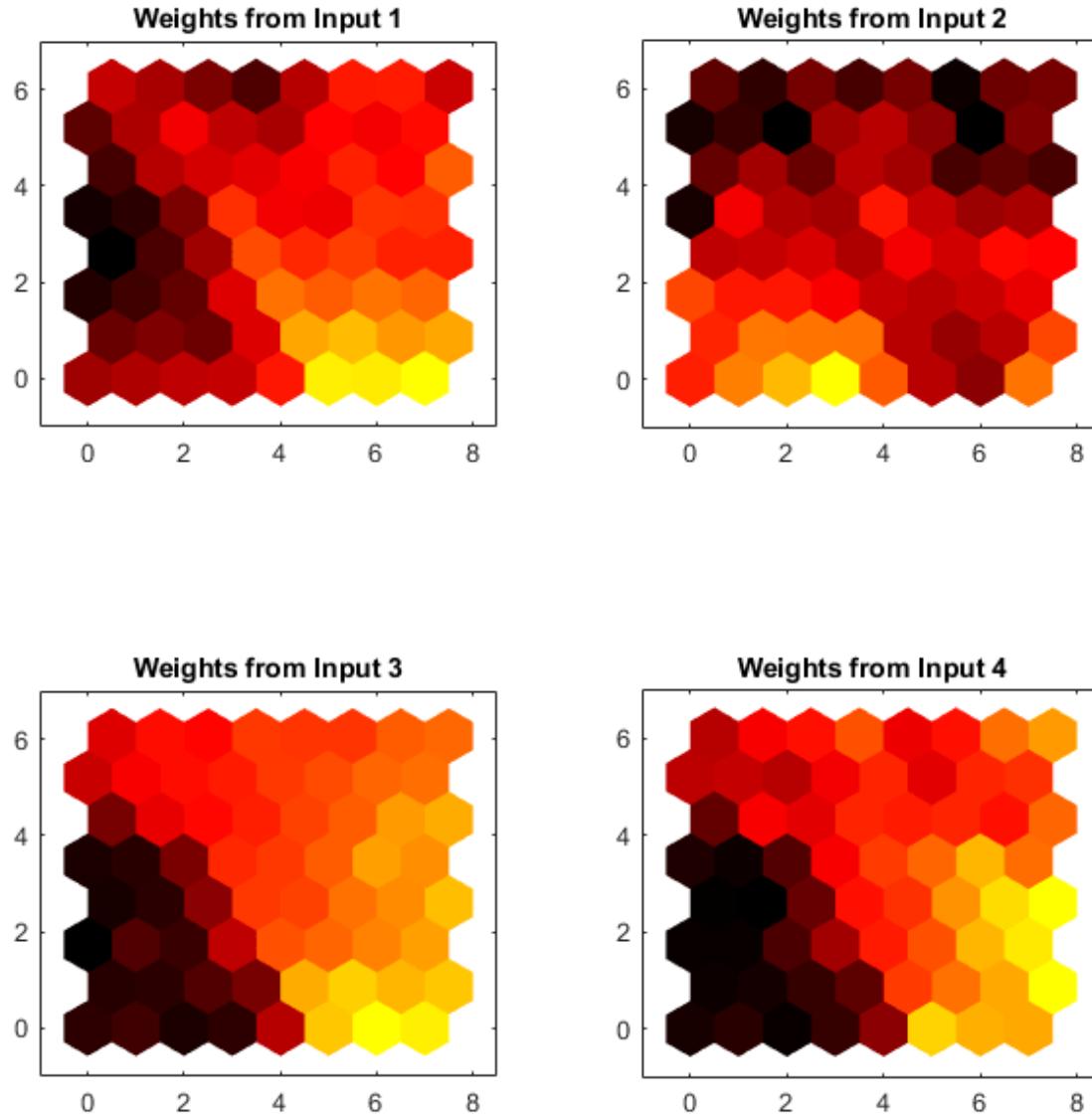
将较大输入空间区域分开的深色连接的长边界表明，边界两侧的类代表特征非常不同的花朵。

**plotsomnd(net)**



使用 **plotsomplanes** 显示四个输入特征中每个特征的权重平面。它们是权重的可视化，这些权重将每个输入连接到以  $8 \times 8$  六边形网格排列的 64 个神经元中的每一个。深色代表较大权重。如果两个输入具有相似的权重平面（它们的颜色梯度可能相同或相反），则表明它们高度相关。

**plotsomplanes(net)**



此示例说明了如何设计一个根据鸢尾花的四个特征对鸢尾花进行聚类的神经网络。

请浏览其他示例和文档，以便更深入地了解神经网络及其应用。

# 基因表达分析

此示例说明如何使用神经网络寻找面包酵母的基因表达谱模式。

## 问题：面包酵母（酿酒酵母）中的基因表达分析

这一分析的目的是了解酿酒酵母（通常称为面包酵母或啤酒酵母）的基因表达。这种真菌用于烤面包和基于葡萄发酵制造葡萄酒。

当酿酒酵母引入富含葡萄糖的培养基时，葡萄糖可以转化为乙醇。最初，酵母通过一种称为“发酵”的代谢过程将葡萄糖转化为乙醇。然而，一旦葡萄糖供应耗尽，酵母就从葡萄糖的厌氧发酵转向乙醇的有氧呼吸。此过程称为双峰转换。这需要着重关注，因为它伴随着基因表达的重大变化。

本示例使用 DNA 微阵列数据来研究酿酒酵母在双峰转换过程中几乎所有基因的瞬时基因表达。

您需要 Bioinformatics Toolbox™ 才能运行此示例。

```
if ~nnDependency.bioInfoAvailable
    errordlg('This example requires Bioinformatics Toolbox.');
    return;
end
```

## 数据

此示例使用的数据来源： DeRisi, JL, Iyer, VR, Brown, PO."Exploring the metabolic and genetic control of gene expression on a genomic scale."Science.1997 Oct 24;278(5338):680-6.PMID:9381177

完整的数据集可以从基因表达综合网站 <https://www.yeastgenome.org> 下载

首先将数据加载到 MATLAB® 中。

```
load yeastdata.mat
```

基因表达水平在双峰转换期间的七个时间点测量而得的。变量 **times** 包含在试验中测量表达水平的时间。变量 **genes** 包含测量其表达水平的基因的名称。变量 **yeastvalues** 包含试验中七个时间步的 "VALUE" 数据或 **LOG\_RAT2N\_MEAN**，即 **CH2DN\_MEAN** 与 **CH1DN\_MEAN** 之比的 log2。

要了解数据的大小，您可以使用 **numel(genes)** 来显示数据集中有多少基因。

```
numel(genes)
```

```
ans =
```

```
6400
```

**genes** 是一个由基因名称组成的元胞数组。您可以使用 MATLAB 元胞数组索引来访问条目：

```
genes{15}
```

```
ans =
```

```
'YAL054C'
```

这表明变量 **yeastvalues** 的第 15 行包含 ORF YAL054C 的表达水平。

## 筛选基因

数据集相当大，许多信息对应于在试验过程中没有显示任何要关注的变化的基因。为了更容易找到关注的基因，首先要通过删除未显示任何关注点的基因表达谱来缩小数据集的大小。共有 6400 个表达谱。您可以使用多种方法将其缩减为包含最重要基因的子集。

如果您浏览基因列表，会看到几个标记为 'EMPTY' 的点。这些是数组上的空点，虽然它们可能有相关联的数据，但出于此示例的目的，您可以将这些点视为噪声。这些点可以使用 **strcmp** 函数找到，并使用索引命令从数据集中删除。

```
emptySpots = strcmp('EMPTY',genes);
yeastvalues(emptySpots,:) = [];
genes(emptySpots) = [];
numel(genes)
```

ans =

6314

在 **yeastvalues** 数据中，您还会看到表达式级别标记为 NaN 的几个位置。这表明在特定的时间步没有收集到该点的数据。处理这些缺失值的一种方法是使用特定基因在某段时间的均值或中位数来估算它们。此示例使用一种不太严格的方法，即直接丢弃未测量到一个或多个表达水平的任何基因的数据。

函数 **isnan** 用于识别具有缺失数据的基因，索引命令用于删除具有缺失数据的基因。

```
nanIndices = any(isnan(yeastvalues),2);
yeastvalues(nanIndices,:) = [];
genes(nanIndices) = [];
numel(genes)
```

ans =

6276

如果您绘制所有其余谱的表达廓，会看到大多数谱是平坦的，并且与其他谱没有显著差异。此平坦数据显然是有用的，因为它表明与这些谱相关联的基因没有受到双峰转换的显著影响；然而，在此示例中，您关注的是伴随双峰转换而在表达中发生巨大变化的基因。您可以使用 Bioinformatics Toolbox™ 中的筛选函数来删除各种其表达谱不能提供基因受代谢变化影响的有用信息的基因。

您可以使用 **genevarfilter** 函数筛选出随时间变化很小的基因。该函数返回与可变基因大小相同的逻辑数组，其中 1 对应于方差大于第 10 个百分位数的 **yeastvalues** 行，0 对应于低于阈值的 **yeastvalues** 的行。

```
mask = genevarfilter(yeastvalues);
% Use the mask as an index into the values to remove the filtered genes.
yeastvalues = yeastvalues(mask,:);
genes = genes(mask);
numel(genes)
```

ans =

5648

函数 **genelowvalfilter** 删除绝对表达值非常低的基因。请注意，基因筛选函数也可以自动计算筛选后的数据和名称。

```
[mask, yeastvalues, genes] = ...
    genelowvalfilter(yeastvalues,genes,'absval',log2(3));
numel(genes)
```

ans =

822

使用 **geneentropyfilter** 删除熵低的谱的基因：

```
[mask, yeastvalues, genes] = ...
    geneentropyfilter(yeastvalues,genes,'prctile',15);
numel(genes)
```

ans =

614

## 主成分分析

现在您已获得可处理的基因列表，可以寻找这些谱之间的关系。

归一化数据的标准差和均值使网络可以基于每个输入值的范围以同等权重看待它们。

主成分分析 (PCA) 是一种有用的方法，可用于降低大型数据集（如微阵列分析中的数据集）的维度。该方法分离数据集的主成分，消除那些对数据集变化贡献极小的成分。

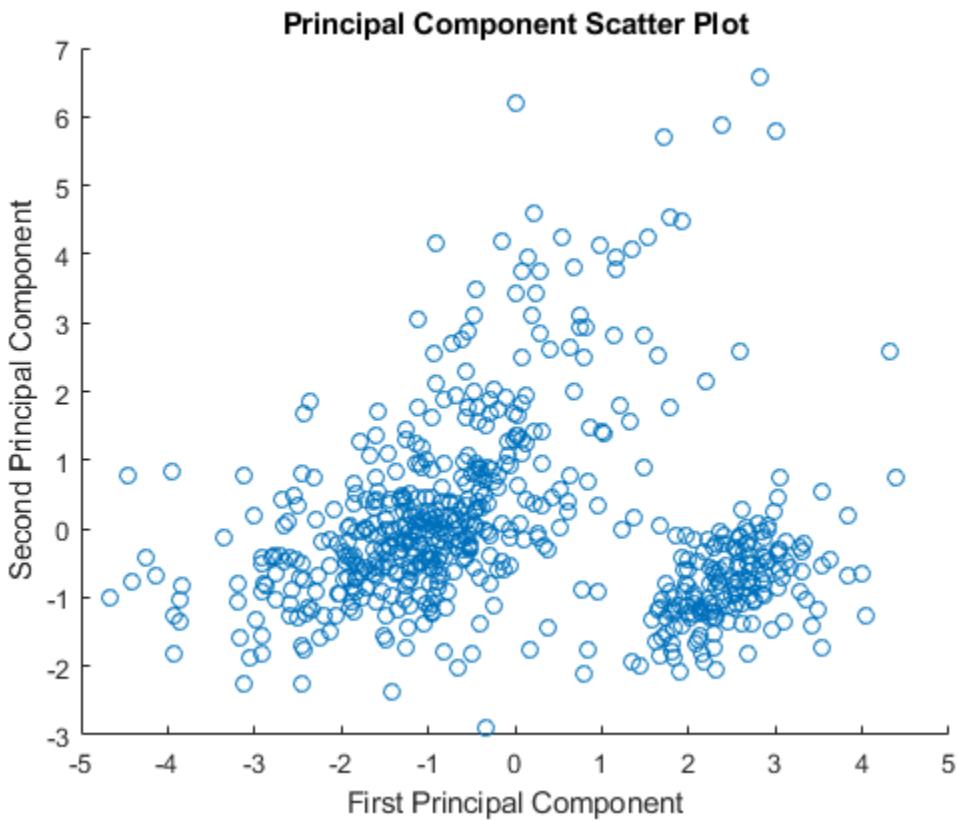
这两个设置变量可用于将 **mapstd** 和 **processpca** 应用于新数据以保持一致。

```
[x,std_settings] = mapstd(yeastvalues'); % Normalize data
[x,pca_settings] = processpca(x,0.15); % PCA
```

首先使用 **mapstd** 对输入向量进行归一化，使它们具有零均值和单位方差。**processpca** 是实现 PCA 算法的函数。传递给 **processpca** 的第二个参数是 0.15。这意味着 **processpca** 会消除那些对数据集总变异贡献不到 15% 的主成分。变量 **pc** 现在包含 **yeastvalues** 数据的主成分。

使用 **scatter** 函数可以可视化主成分。

```
figure
scatter(x(1,:),x(2,:));
xlabel('First Principal Component');
ylabel('Second Principal Component');
title('Principal Component Scatter Plot');
```



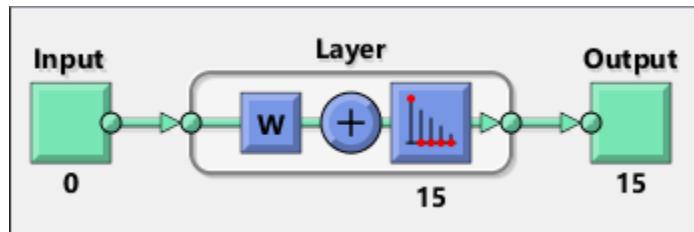
### 聚类分析：自组织映射

现在可以使用自组织映射 (SOM) 聚类算法对主成分进行聚类。

**selforgmap** 函数创建一个自组织映射网络，然后可以使用 **train** 函数训练该网络。

输入大小为 0，因为网络尚未配置成与我们的输入数据相匹配。这将在训练网络时进行。

```
net = selforgmap([5 3]);
view(net)
```

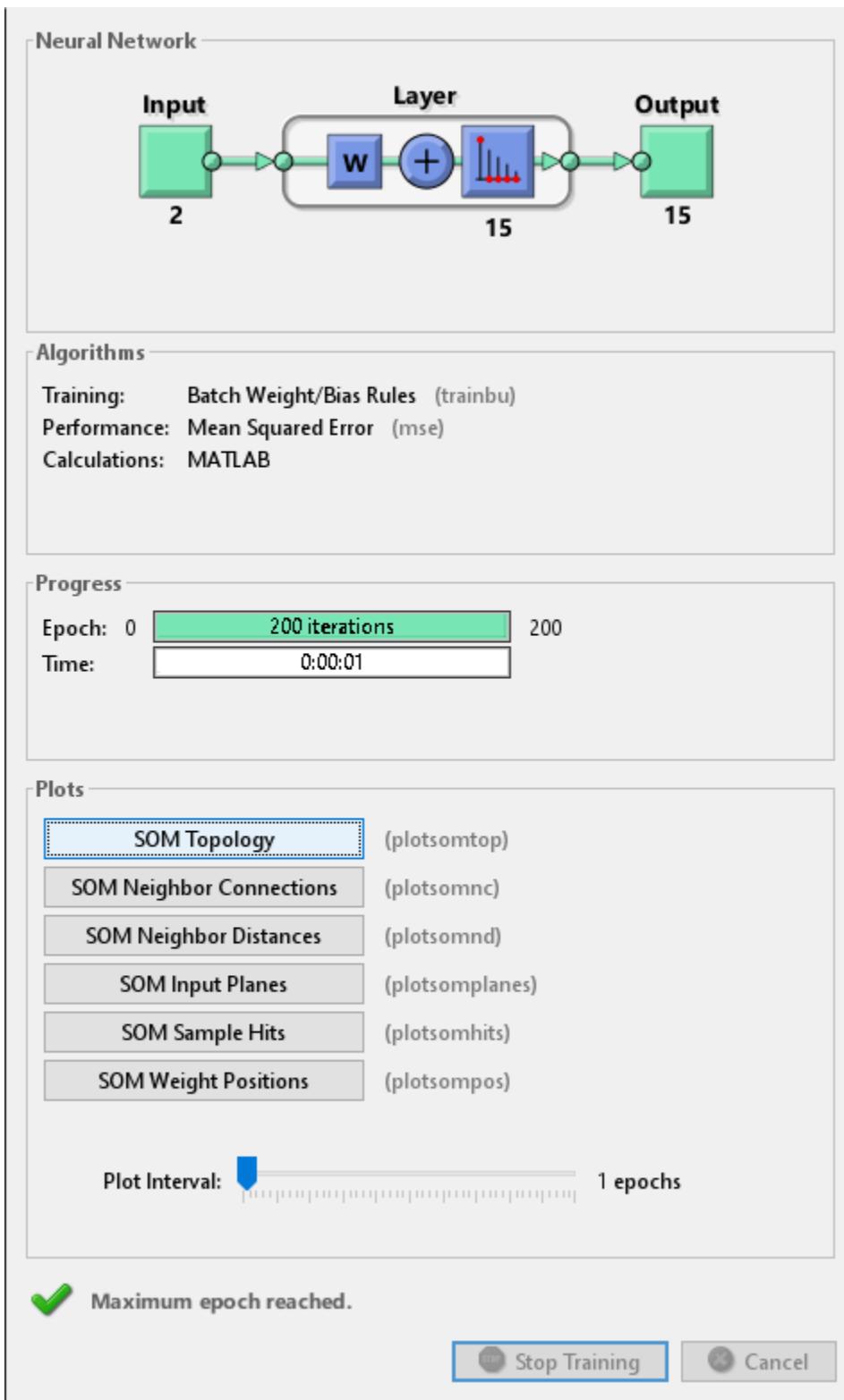


现在网络已准备就绪，可以开始训练。

神经网络训练工具显示正在接受训练的网络和用于训练该网络的算法。它还显示训练过程中的训练状态以及停止训练的条件（以绿色突出显示）。

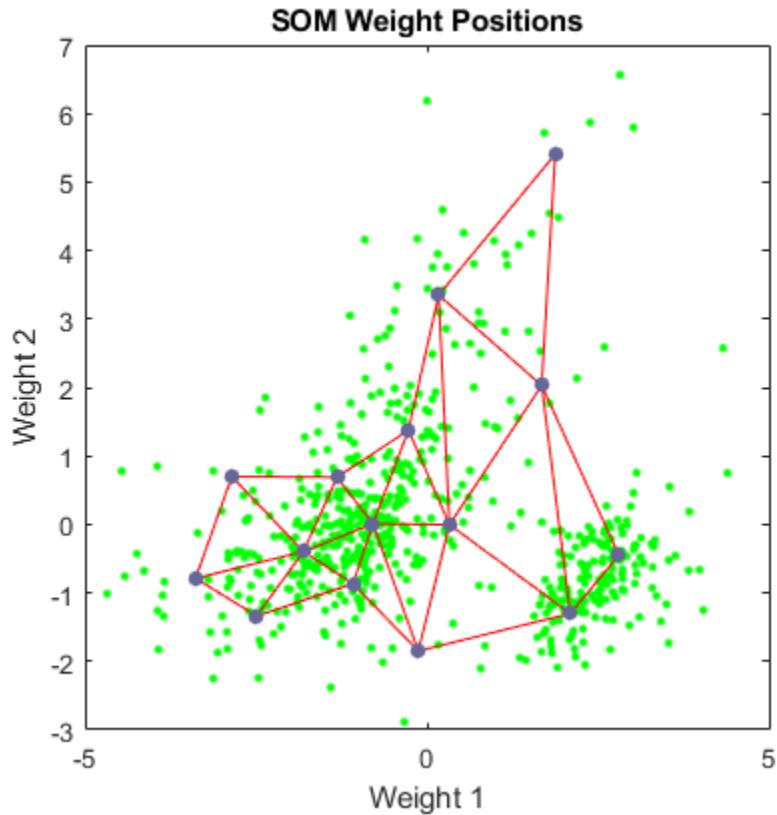
底部的按钮用于打开有用的绘图，这些图可以在训练期间和训练后打开。算法名称和绘图按钮旁边的链接可打开有关这些主题的文档。

```
net = train(net,x);
nntraintool
nntraintool('close')
```



使用 **plotsompos** 在数据的前两个维度的散点图上显示网络。

```
figure  
plotsompos(net,x);
```

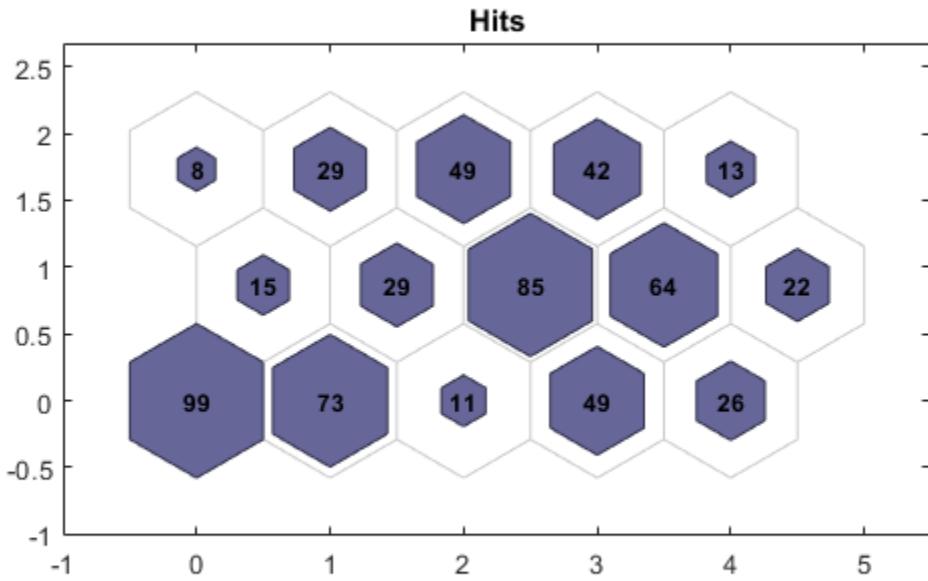


您可以通过查找数据集中每个点的最近节点，使用 SOM 来分配聚类。

```
y = net(x);  
cluster_indices = vec2ind(y);
```

使用 **plotsomhits** 查看向映射中的每个神经元分配了多少向量。

```
figure  
plotsomhits(net,x);
```



您还可以使用 Statistics and Machine Learning Toolbox™ 中提供的其他聚类算法（如分层聚类和 K 均值）进行聚类分析。

#### 术语表

**ORF** - 开放阅读框 (ORF) 是基因序列的一部分，包含一个碱基序列，不被终止序列打断，可能对蛋白质编码。

# 磁悬浮建模

此示例说明 NARX（具有外部输入的非线性自回归）神经网络如何对磁悬浮动态系统建模。

## 问题：对磁悬浮系统进行建模

在此示例中，我们尝试构建一个可以对使用控制电流悬浮的磁体的动态行为进行预测的神经网络。

该系统的特点是磁体的位置和控制电流共同决定了磁体在片刻之后的位置。

这是一个时序问题示例，它使用反馈时序的过去值（磁体位置）和外部输入序列（控制电流）来预测反馈序列的将来值。

## 为什么使用神经网络？

神经网络非常擅长处理时序问题。具有足够元素（称为神经元）的神经网络可以以任意准确度对动态系统进行建模。它们特别适合处理非线性动态问题。因此，神经网络是处理此问题的优选方案。

通过使用响应控制电流的实际悬浮磁体位置的记录来设计网络。

## 准备数据

通过将数据组织成两个矩阵（输入时序 X 和目标时序 T）来为神经网络设置函数拟合问题数据。

输入时序 X 是一个行元胞数组，其中每个元素是控制电流的相关时间步。

目标时序 T 也是一个行元胞数组，其中每个元素是悬浮磁体位置的相关时间步。

使用以下命令加载一个这样的数据集。

```
[x,t] = maglev_dataset;
```

我们可以查看输入 X 和目标 T 的大小。

请注意，X 和 T 都有 4001 列。这些列表示控制电流和磁体位置的 4001 个时间步。

```
size(x)
size(t)
```

```
ans =
```

```
1      4001
```

```
ans =
```

```
1      4001
```

## 使用神经网络进行时序建模

下一步是创建一个学习对磁体位置变化方式进行建模的神经网络。

由于神经网络以随机初始权重开始，因此该示例的结果在每次运行时都会略有不同。我们设置了随机种子来避免这种随机性。但这对于您自己的应用情形并不是必需的。

```
setdemorandstream(491218381)
```

双层（即，一个隐藏层）NARX 神经网络可以拟合任何动态输入-输出关系，前提是隐藏层中有足够的神经元。非输出层称为隐含层。

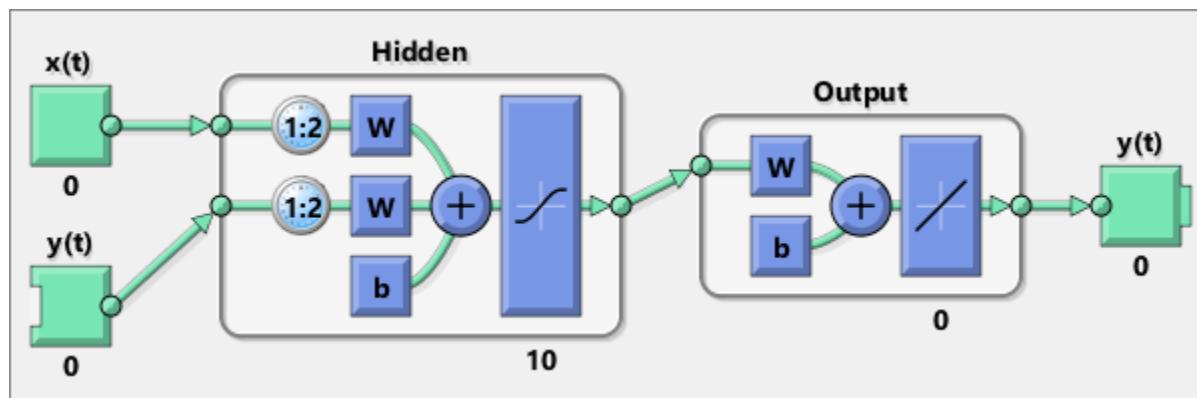
对于此示例，我们将尝试具有 10 个神经元的单隐藏层。一般情况下，问题越困难，需要的神经元和层就越多。问题越简单，需要的神经元就越少。

我们还会尝试将具有两个延迟的抽头延迟用于外部输入（控制电流）和反馈（磁体位置）。延迟越多，网络便能对越复杂的动态系统进行建模。

输入和输出的大小为 0，因为网络尚未配置成与我们的输入数据和目标数据相匹配。这将在训练网络时进行。

输出  $y(t)$  也是输入，其延迟版本反馈到网络中。

```
net = narxnet(1:2,1:2,10);
view(net)
```



训练网络之前，我们必须先使用外部输入和反馈时序的前两个时间步来填充网络的两个抽头延迟状态。

此外，我们需要将两个反馈时序同时用作输入时序和目标时序。

函数 PREPARETS 帮我们准备用于仿真和训练的时序数据。 $X_s$  将包含要提交到网络的移位输入时序和目标时序。 $X_i$  是初始输入延迟状态。 $A_i$  是层延迟状态（在这种情况下为空，因为没有层到层的延迟）， $T_s$  是移位反馈时序。

```
[Xs,Xi,Ai,Ts] = preparets(net,x,{},t);
```

现在网络已准备就绪，可以开始训练。时间步自动分为训练集、验证集和测试集。训练集用于对网络进行训练。只要网络针对验证集持续改进，训练就会继续。测试集提供完全独立的网络准确度测量。

神经网络训练工具显示正在接受训练的网络和用于训练该网络的算法。它还显示训练过程中的训练状态以及停止训练的条件（以绿色突出显示）。

底部的按钮用于打开有用的绘图，这些图可以在训练期间和训练后打开。算法名称和绘图按钮旁边的链接可打开有关这些主题的文档。

```
[net,tr] = train(net,Xs,Ts,Xi,Ai);
nntraintool
```



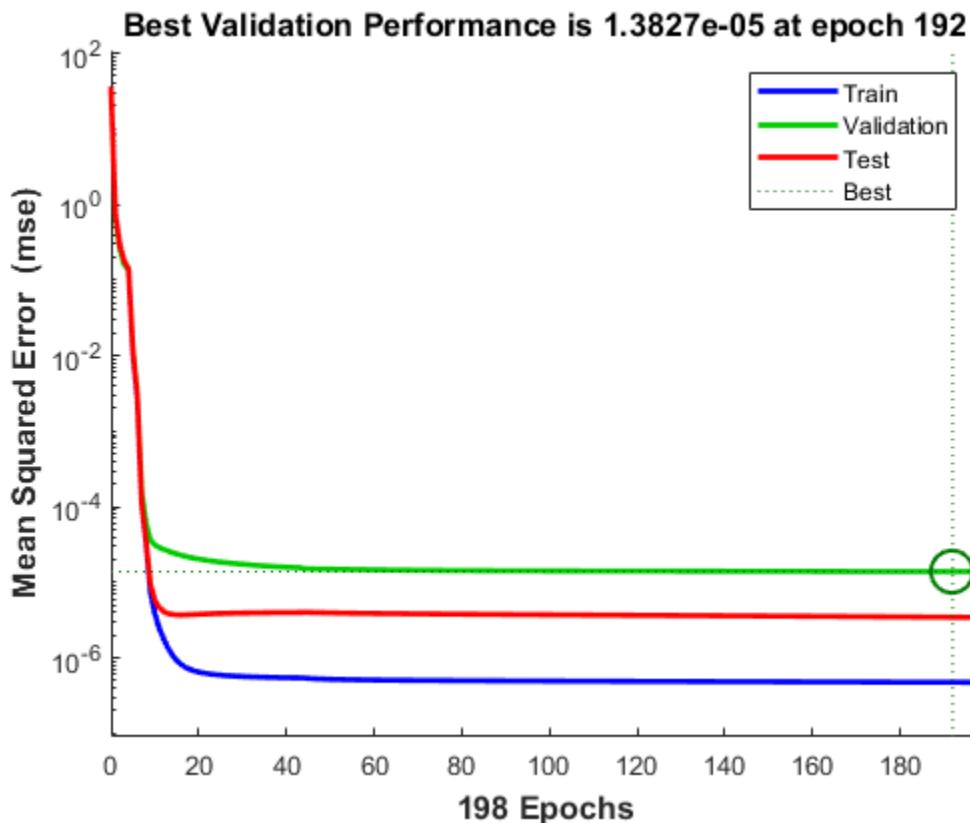
```
nntraintool('close')
```

要查看在训练过程中网络性能的改善情况，请点击训练工具中的 Performance 按钮，或调用 PLOTPERFORM。

性能以均方误差衡量，并以对数刻度显示。随着网络训练的加深，均方误差迅速降低。

绘图会显示训练集、验证集和测试集的性能。

```
plotperform(tr)
```



### 测试神经网络

现在可以测量经过训练的神经网络在所有时间步的均方误差。

```
Y = net(Xs,Xi,Ai);
```

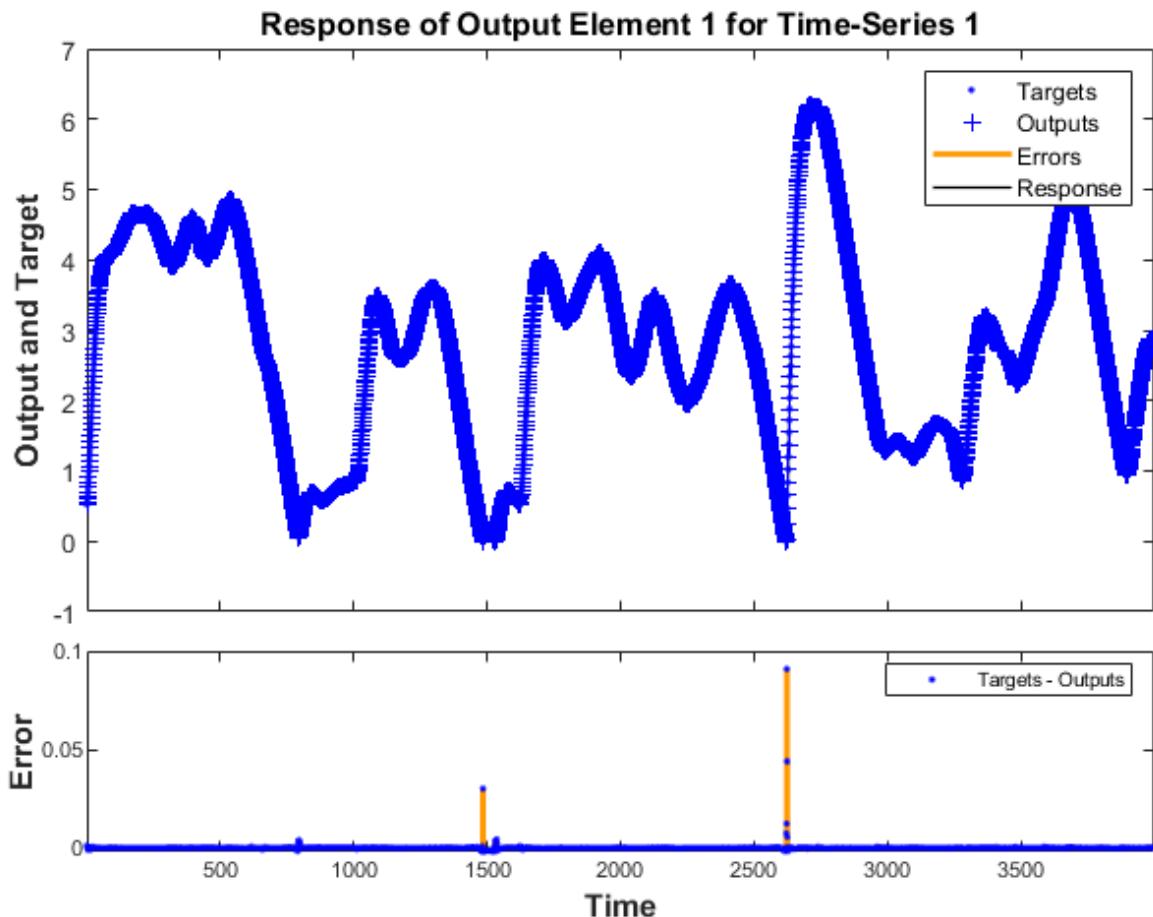
```
perf = mse(net,Ts,Y)
```

```
perf =
```

```
2.9245e-06
```

PLOTRESPONSE 将显示网络的响应与实际磁体位置的比较。如果模型准确，“+”点将跟随菱形点，底轴的误差将非常小。

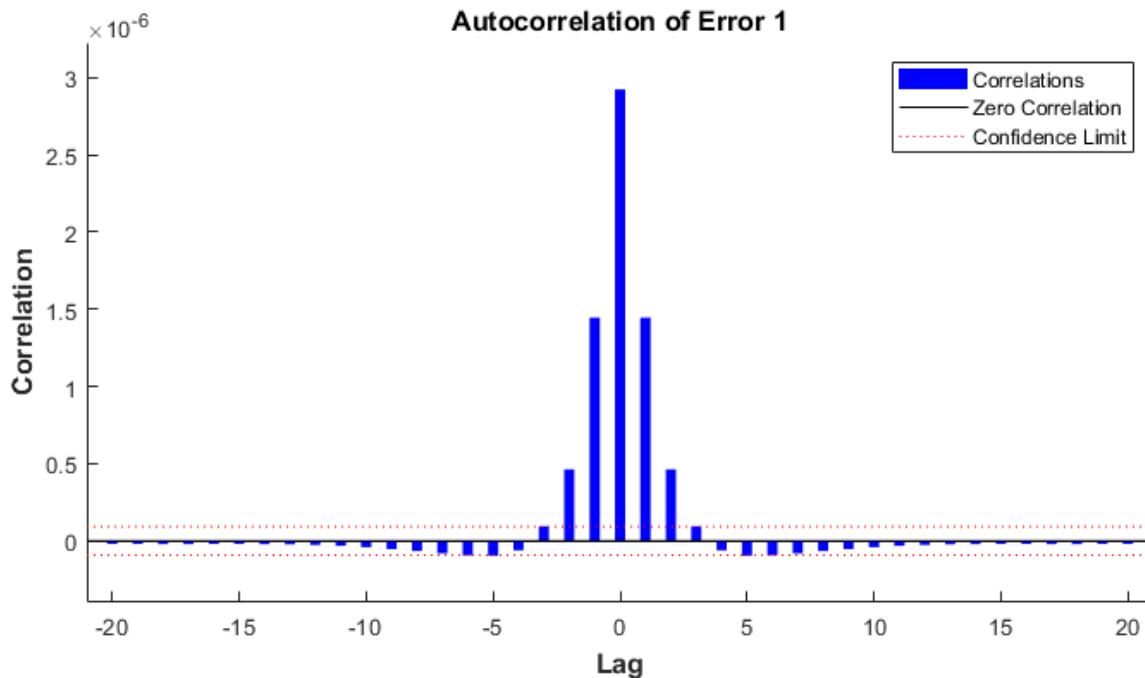
```
plotresponse(Ts,Y)
```



PLOTERRCORR 显示时间  $t$  处的误差  $e(t)$  与基于不同滞后的误差  $e(t+lag)$  之间的相关性。中心线显示均方误差。如果网络训练良好，所有其他线将短得多，并且大多数（如果不是全部）将在红色置信界限范围内。

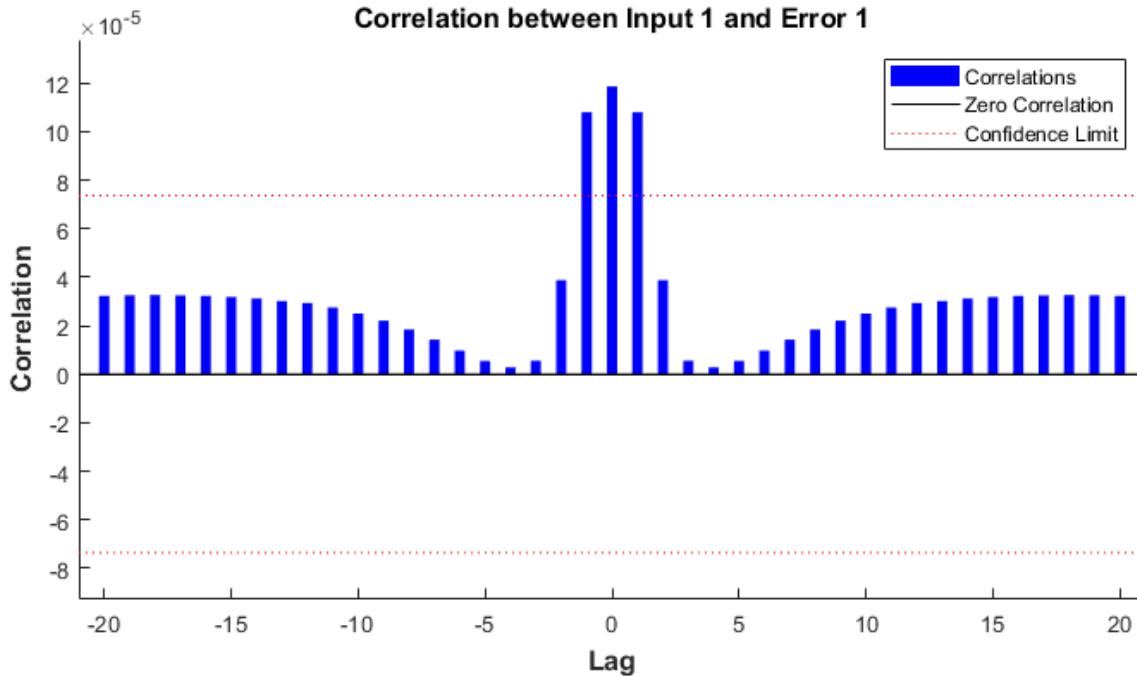
函数 GSUBTRACT 用于计算误差。该函数对减法进行泛化，以支持求元胞数组数据之间的差。

```
E = gsubtract(Ts,Y);
plotterrcorr(E)
```



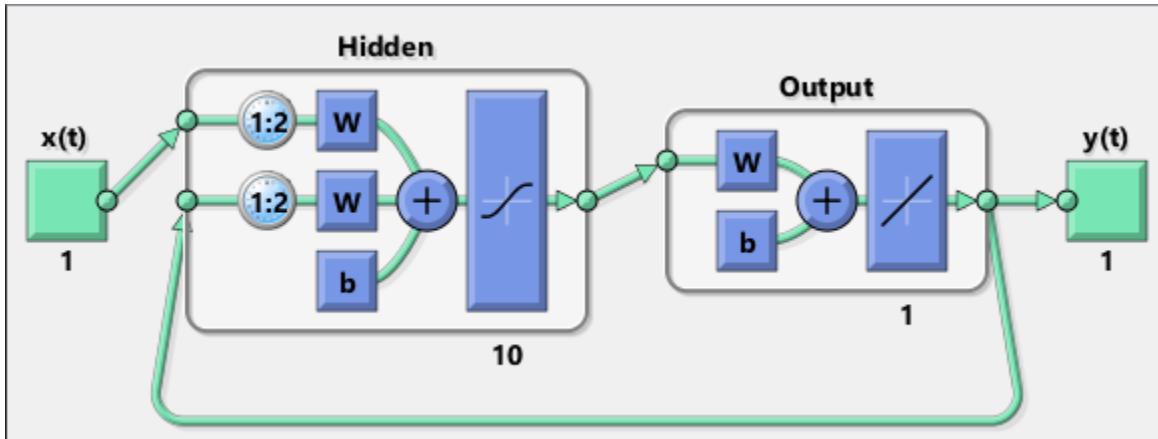
类似地，PLOTINERRCORR 显示误差与具有不同程度滞后的输入的相关性。在这种情况下，大多数或所有线都应在置信界限范围内，包括中心线。

```
plotinerrcorr(Xs,E)
```



网络以开环形式进行训练，其中目标被用作反馈输入。也可以将网络转换为闭环形式，其中，其自身预测成为反馈输入。

```
net2 = closeloop(net);
view(net2)
```

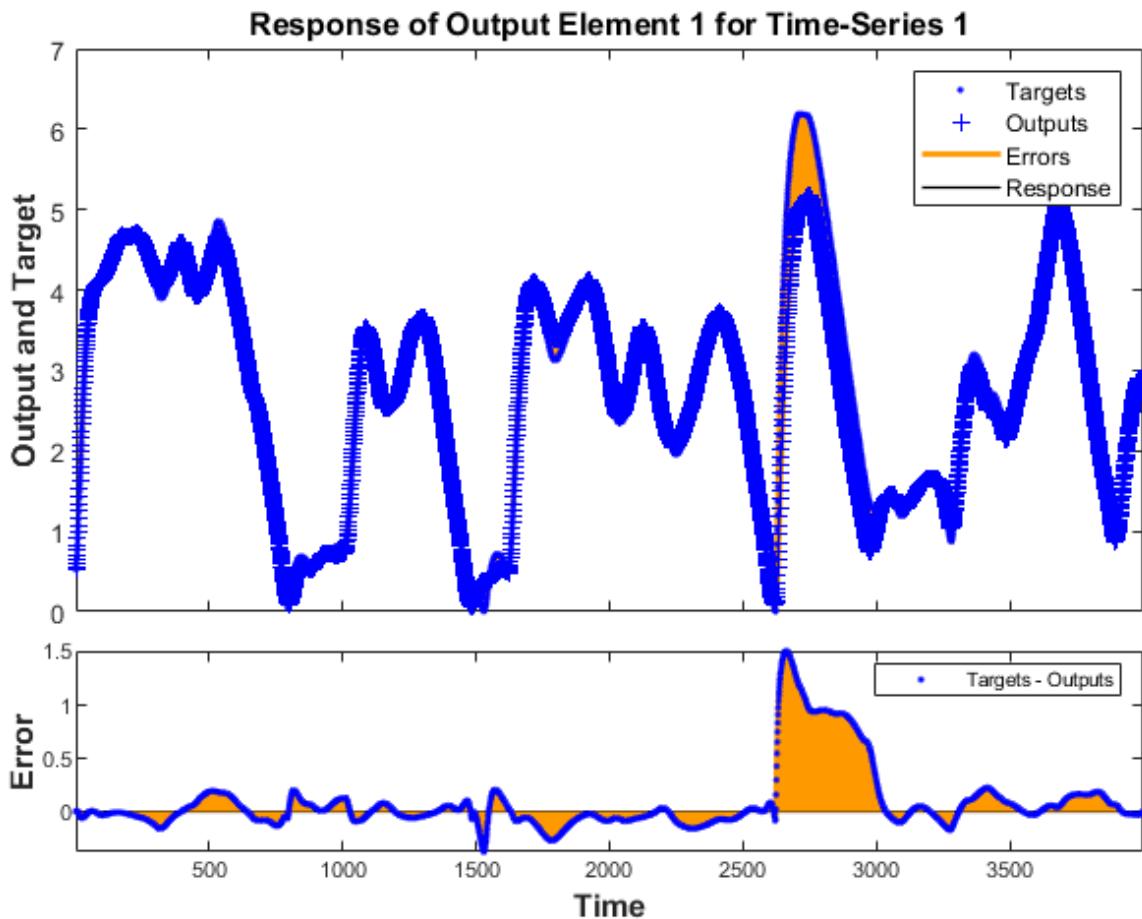


我们可以以闭环形式仿真网络。在这种情况下，仅为网络提供初始磁体位置，然后，网络必须递归使用其自己的预测位置来预测新位置。

这会迅速导致预测响应与实际响应之间的拟合变差。即使模型非常好，也会发生这种情况。但了解它们在分离之前匹配了多少步也是很有趣的。

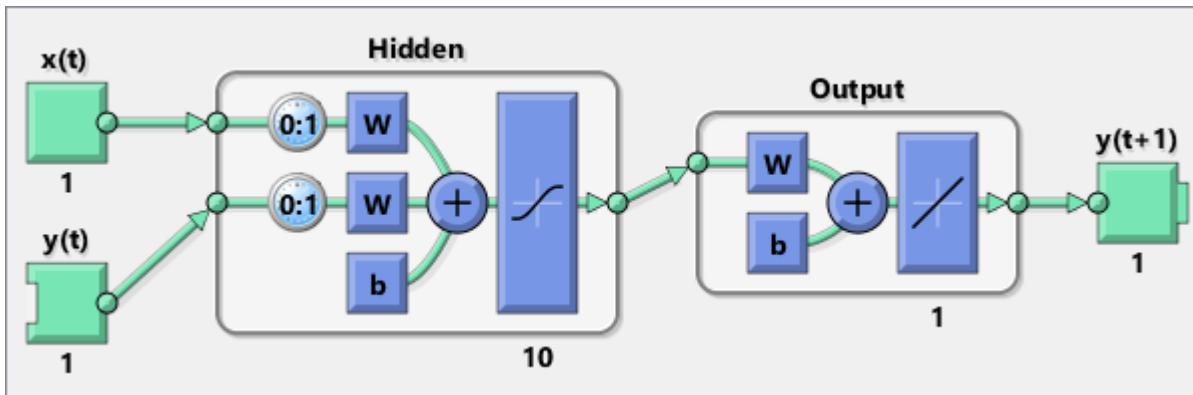
PREPARETS 会再次准备时序数据，同时会考虑更改后的网络。

```
[Xs,Xi,Ai,Ts] = preparets(net2,x,{},t);
Y = net2(Xs,Xi,Ai);
plotresponse(Ts,Y)
```



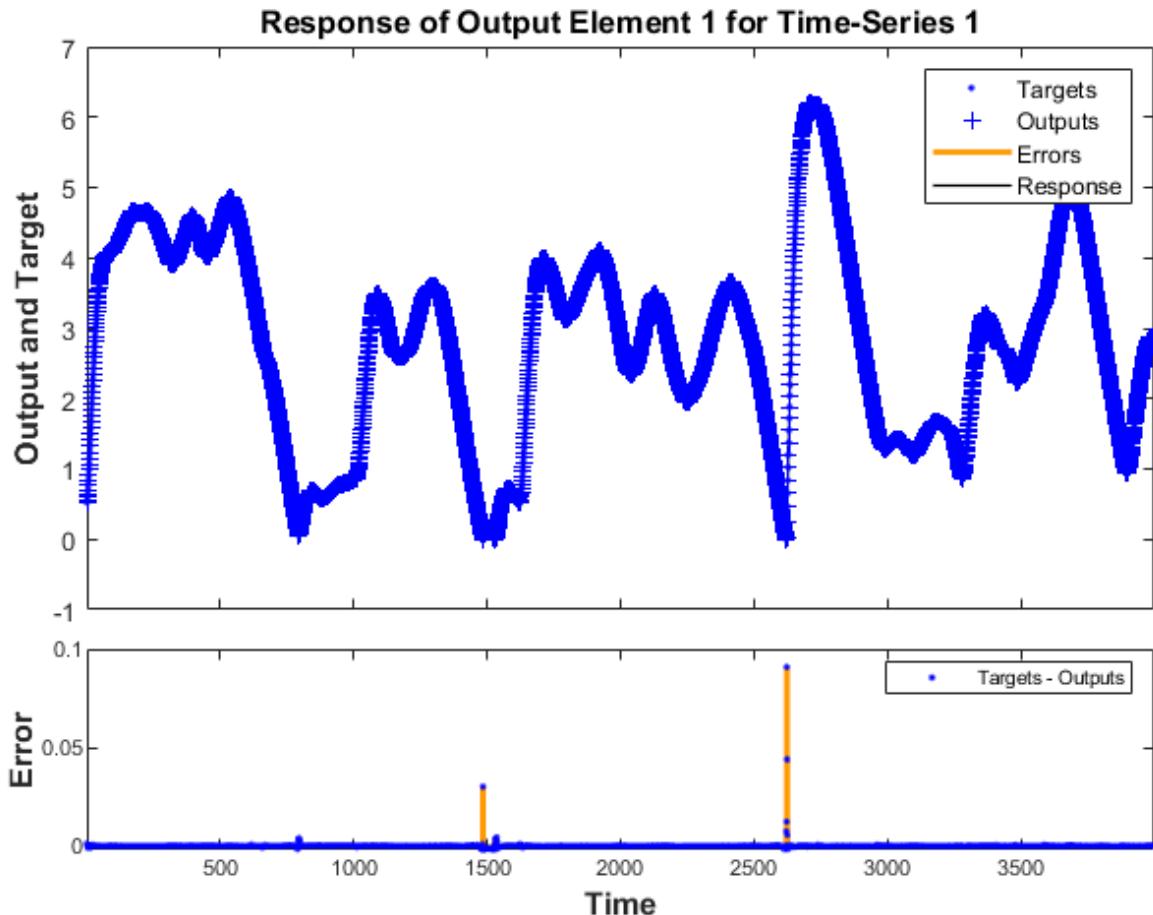
如果应用程序要求我们提前实际磁体位置一个时间步访问预测磁体位置，我们可以从网络中删除一个延迟，以便在任何给定时间  $t$ ，输出均为时间  $t+1$  处的位置的估计值。

```
net3 = removedelay(net);
view(net3)
```



我们再次使用 PREPARETS 准备用于仿真的时序。这次网络又是非常准确，因为它进行的是开环预测，但输出移位一个时间步。

```
[Xs,Xi,Ai,Ts] = preparets(net3,x,{},t);
Y = net3(Xs,Xi,Ai);
plotresponse(Ts,Y)
```



此示例说明了如何设计一个对动态磁悬浮系统的行为进行建模的神经网络。

请浏览其他示例和文档，以便更深入地了解神经网络及其应用。

## 竞争学习

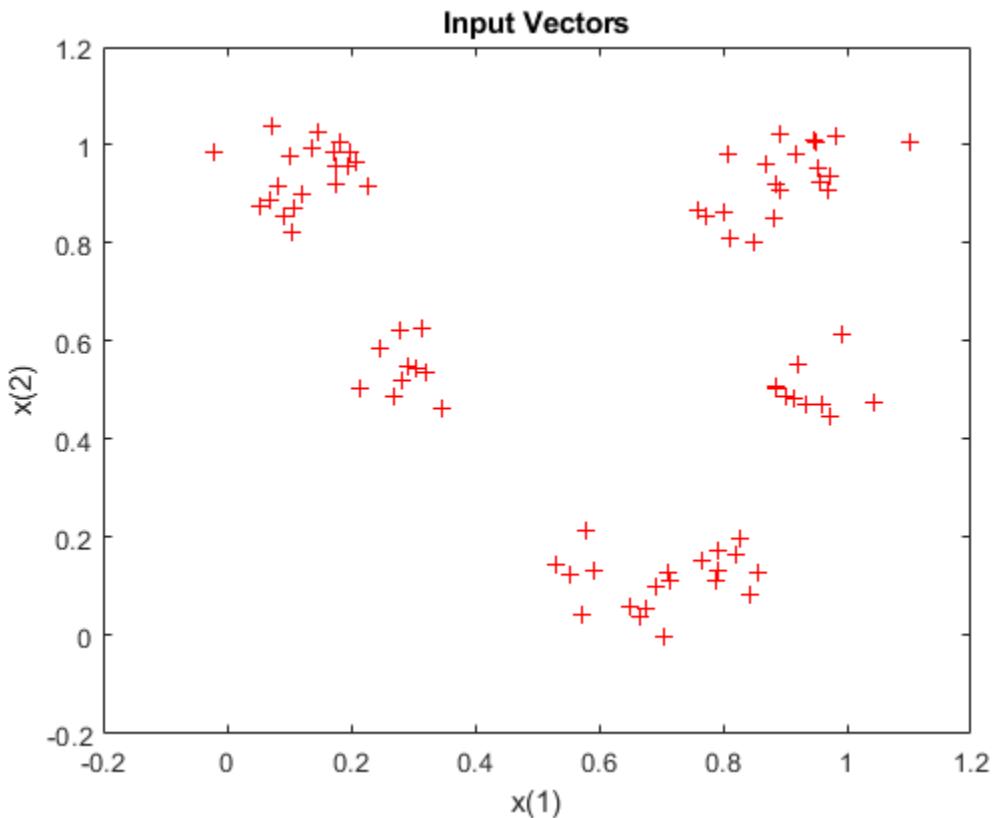
竞争层中的神经元可通过学习来表示输入向量在输入空间中出现的不同区域。

P 是一组随机生成但聚类的测试数据点。下面绘制了这些数据点。

竞争网络将用于将这些点分成若干自然类。

```
% Create inputs X.
bounds = [0 1; 0 1]; % Cluster centers to be in these bounds.
clusters = 8; % This many clusters.
points = 10; % Number of points in each cluster.
std_dev = 0.05; % Standard deviation of each cluster.
x = nngenc(bounds,clusters,points,std_dev);

% Plot inputs X.
plot(x(1,:),x(2,:),'+r');
title('Input Vectors');
xlabel('x(1)');
ylabel('x(2)');
```



此处 COMPETLAYER 接受两个参数，即神经元数量和学习率。

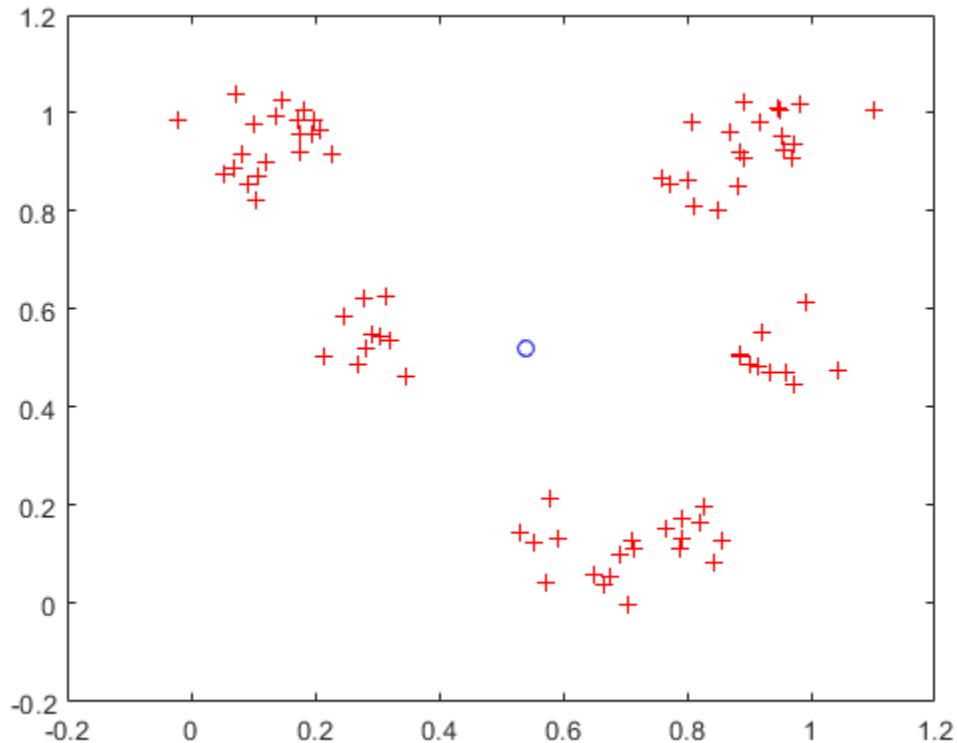
我们可以配置网络输入（通常由 TRAIN 自动完成），并绘制初始权重向量以查看其分类尝试过程。

对权重向量 (o) 进行训练，使它们出现在输入向量 (+) 的聚类的中心。

```

net = competlayer(8,.1);
net = configure(net,x);
w = net.IW{1};
plot(x(1,:),x(2,:),'+r');
hold on;
circles = plot(w(:,1),w(:,2),'ob');

```



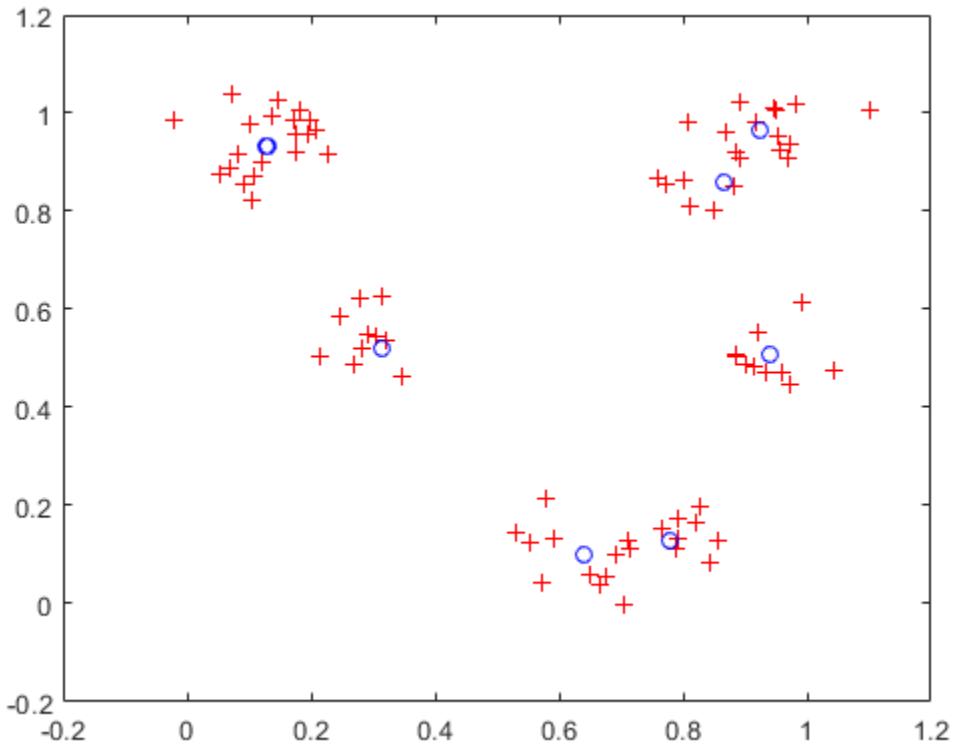
设置在停止之前需要训练的轮数，并训练此竞争层（可能需要几秒）。

在同一图上绘制更新后的层权重。

```

net.trainParam.epochs = 7;
net = train(net,x);
w = net.IW{1};
delete(circles);
plot(w(:,1),w(:,2),'ob');

```



现在我们可以使用竞争层作为分类器，其中每个神经元都对应于一个不同的类别。此处，我们将一个输入向量  $X_1$  定义为  $[0; 0.2]$ 。

输出  $Y$  指示哪个神经元正在响应，从而指示输入属于哪个类。

```
x1 = [0; 0.2];
y = net(x1)
```

$y = 8 \times 1$

```
0
1
0
0
0
0
0
0
```

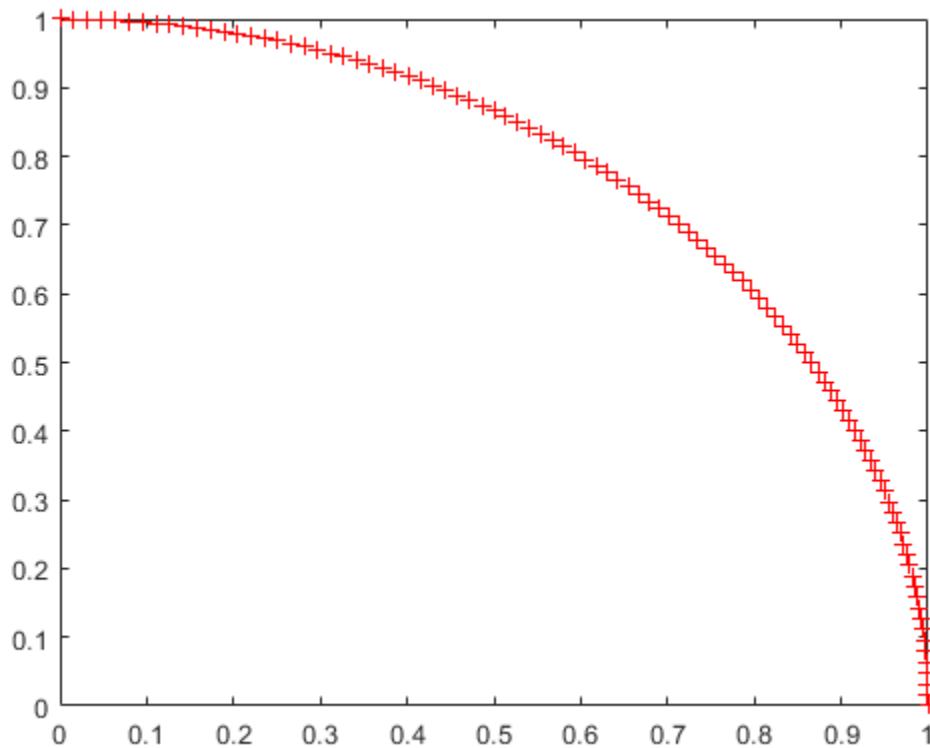
## 一维自组织映射

二维层中的神经元可学习表示输入向量在输入空间出现的不同区域。此外，邻近的神经元可学习对相似的输入进行响应，从而该层可学习所呈现的输入空间的拓扑。

在本示例中，我们创建了位于单位圆上的 100 个数据点。

竞争网络将用于将这些点分成若干自然类。

```
angles = 0:0.5*pi/99:0.5*pi;
X = [sin(angles); cos(angles)];
plot(X(1,:),X(2,:),'+r')
```



映射将是由 10 个神经元组成的一维层。

```
net = selforgmap(10);
```

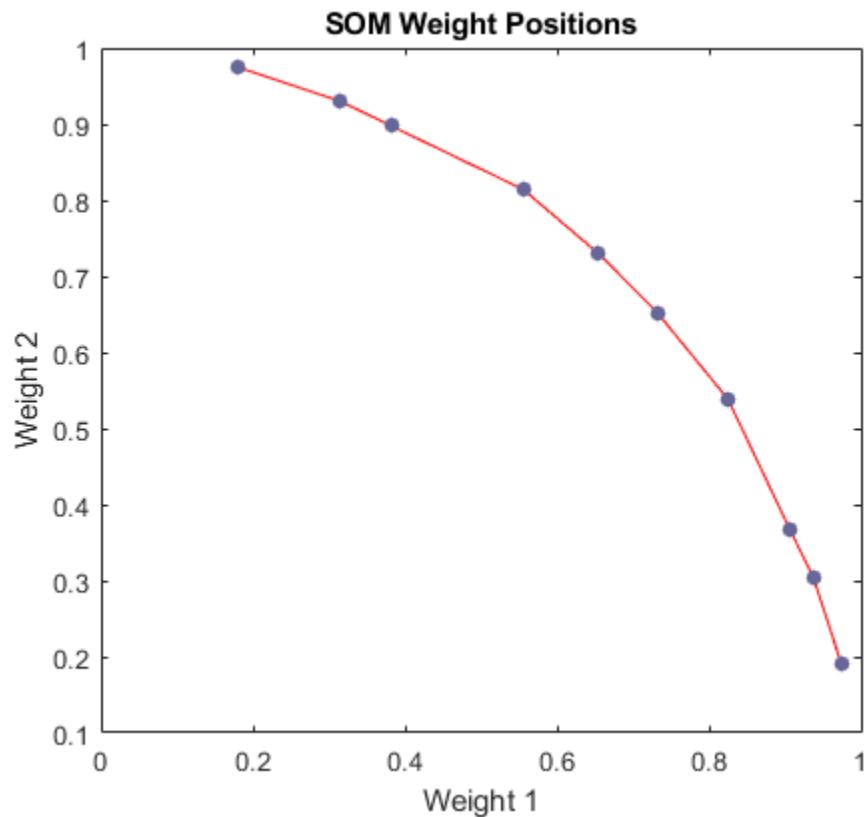
指定网络将接受 10 轮训练，并使用 **train** 基于输入数据对网络进行训练。

```
net.trainParam.epochs = 10;
net = train(net,X);
```

现在使用 **plotsompos** 绘制经过训练的网络的权重位置。

红点是神经元的权重向量，蓝线连接在距离 1 内的每对红点。

```
plotsompos(net)
```



该映射现在可用于对输入进行分类，例如 [1; 0]。神经元 1 或 10 的输出应该为 1，因为上述输入向量位于所呈现的输入空间的一端。第一对数字表示神经元，单个数字表示其输出。

```
x = [1;0];
a = net(x)
```

a = 10×1

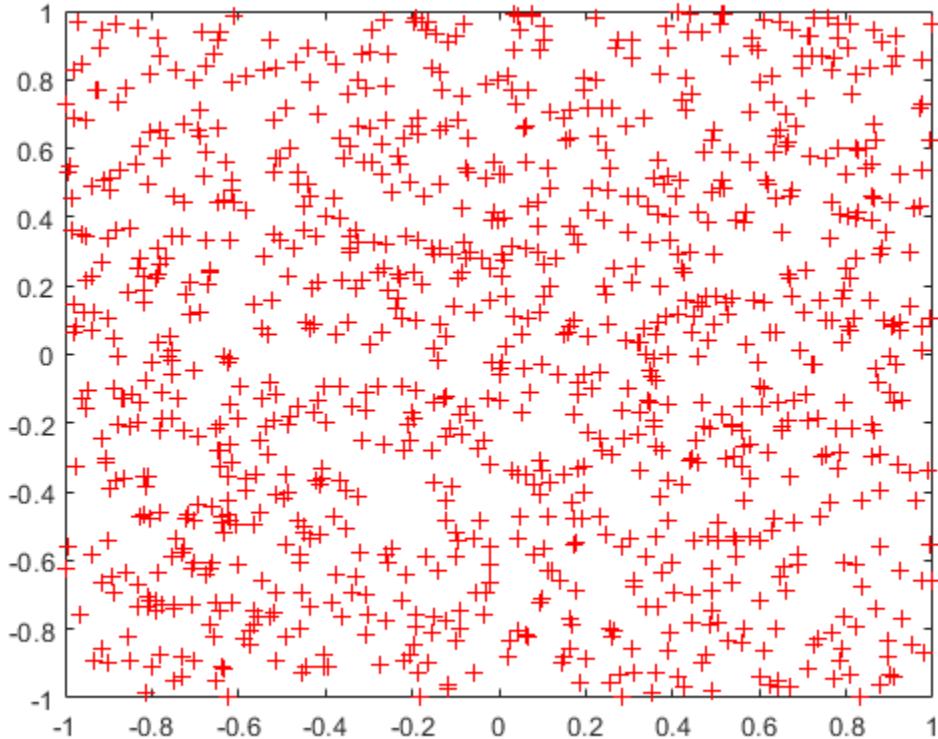
```
0
0
0
0
0
0
0
0
0
1
```

## 二维自组织映射

如同一维问题一样，这种自组织映射将学习表示输入向量在输入空间中出现的不同区域。然而，在此示例中，神经元会形成二维网格，而不是一条线。

我们要对一个矩形中的 1000 个二元素向量进行分类。

```
X = rands(2,1000);
plot(X(1,:),X(2,:),'+r')
```



我们将使用  $5 \times 6$  神经元层对上述向量进行分类。我们希望每个神经元对矩形的不同区域作出响应，相邻神经元对相邻区域作出响应。

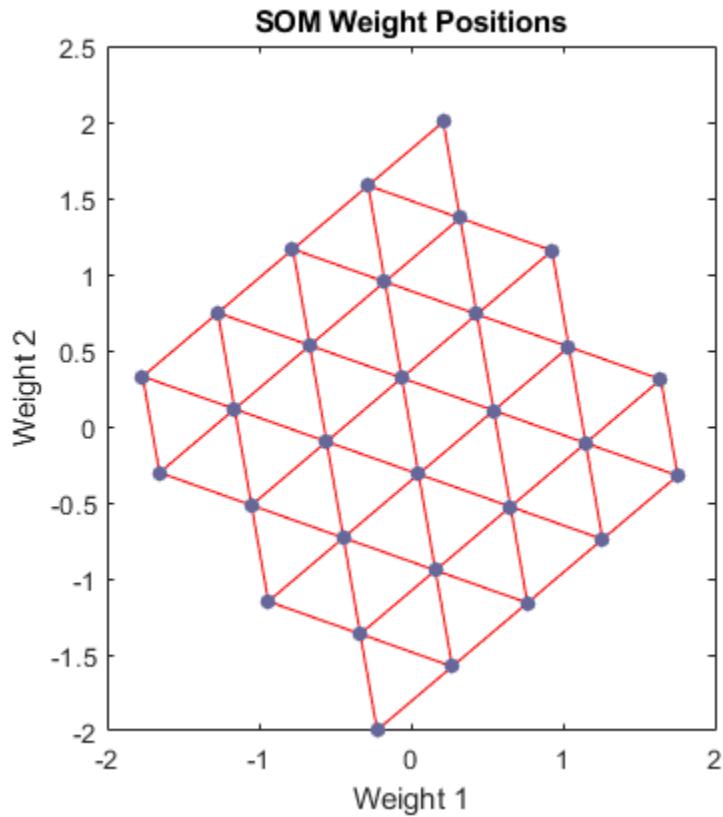
将网络配置为匹配输入的维度。此步骤是必需的，因为我们要绘制初始权重。通常情况下，训练时会自动进行配置。

```
net = selforgmap([5 6]);
net = configure(net,X);
```

我们可以通过使用 **plotsompos** 可视化我们刚刚创建的网络。

每个神经元在其两个权重的位置用红点表示。最初，所有神经元的权重相同，位于向量的中间，因此只出现一个点。

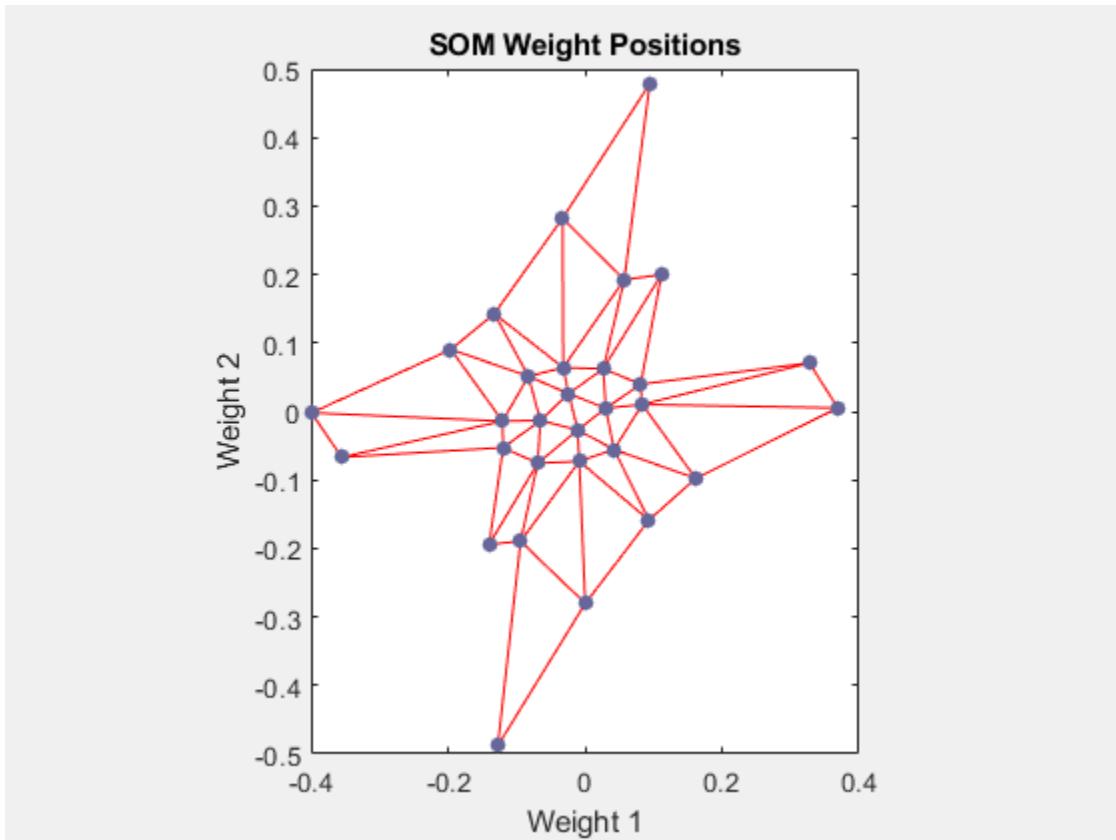
```
plotsompos(net)
```



现在我们基于 1000 个向量对映射进行 1 轮训练，并重新绘制网络权重。

在训练后，注意神经元层已开始自组织，每个神经元现在界定输入空间的不同区域，并且相邻（连接的）神经元对相邻区域作出响应。

```
net.trainParam.epochs = 1;  
net = train(net,X);  
plotsompos(net)
```



现在，我们可以通过将向量输入网络并观察哪个神经元有响应来对向量进行分类。

由“1”表示的神经元有响应，因此  $x$  属于该类。

```
x = [0.5;0.3];
y = net(x)
```

$y = 30 \times 1$

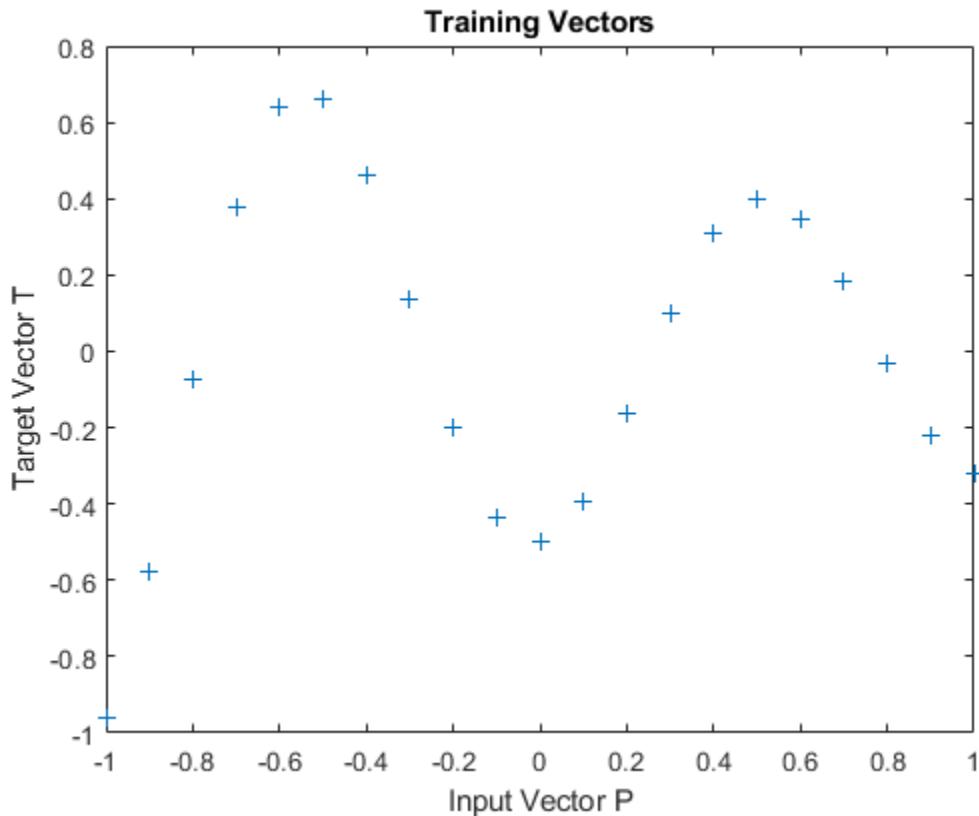
```
0
0
0
0
0
0
0
0
0
0
⋮
```

## 径向基逼近

此示例使用 NEWRB 函数创建一个径向基网络，该网络可逼近由一组数据点定义的函数。

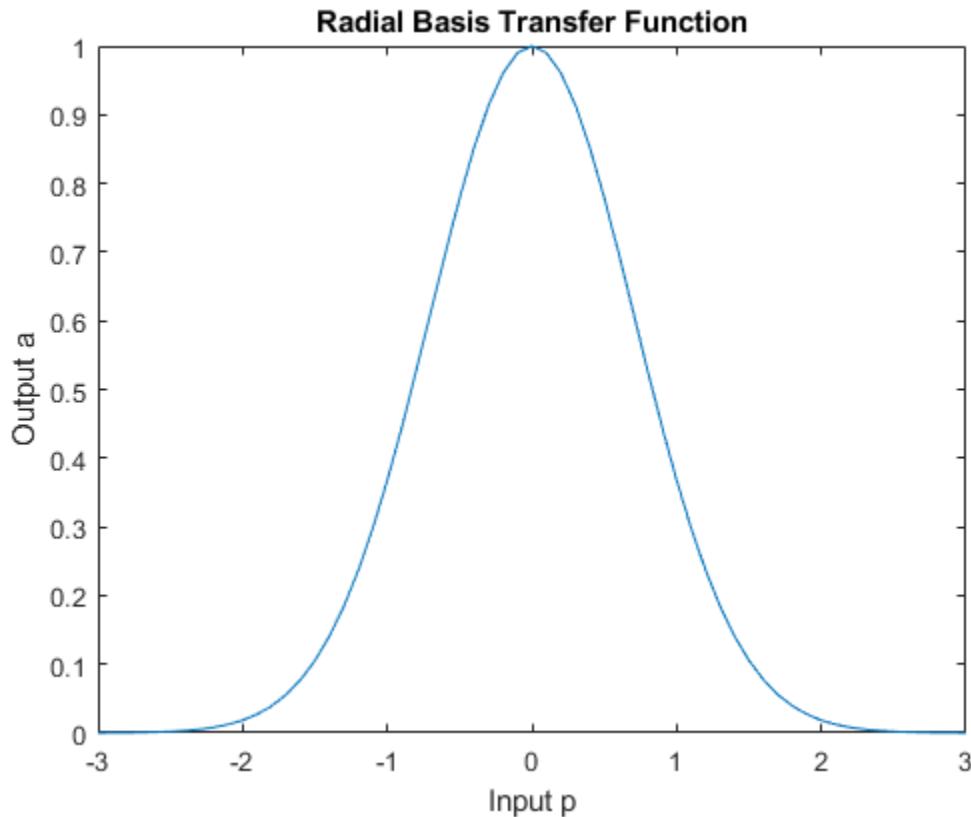
定义 21 个输入 P 和相关目标 T。

```
X = -1.:1:1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 ...
       .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 ...
       .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201];
plot(X,T,'+');
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
```



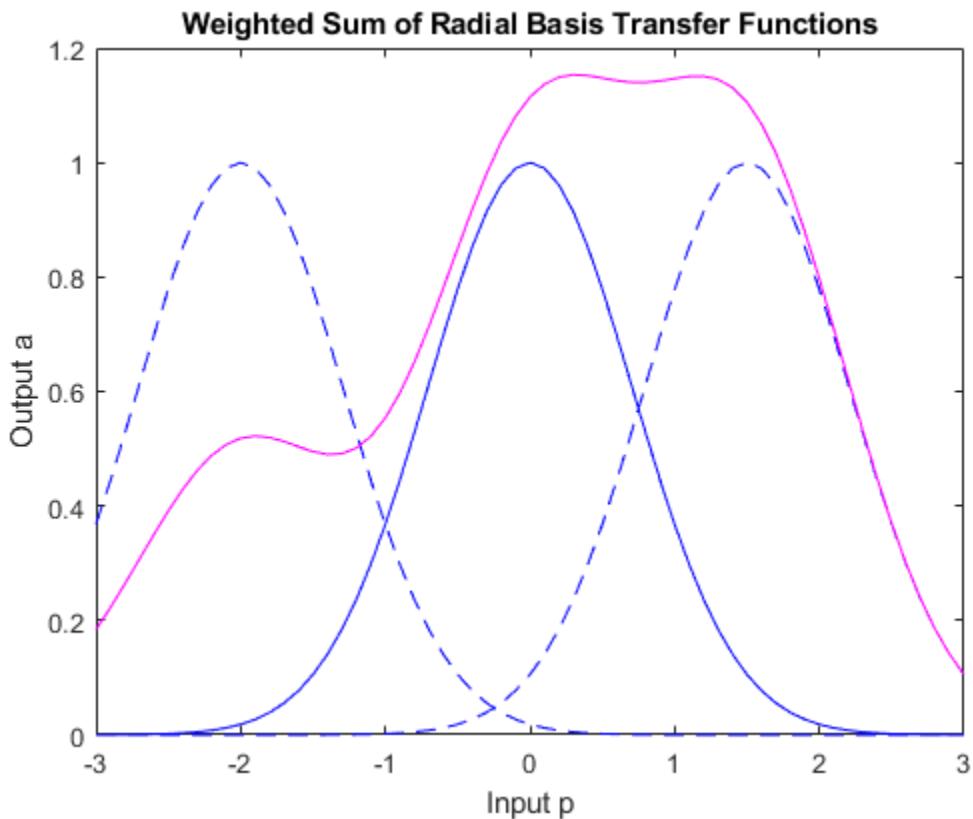
我们希望找到一个可拟合这 21 个数据点的函数。一种方法是使用径向基网络来实现。径向基网络具有两个层，分别是径向基神经元的隐藏层和线性神经元的输出层。以下是隐含层使用的径向基传递函数。

```
x = -3:1:3;
a = radbas(x);
plot(x,a)
title('Radial Basis Transfer Function');
xlabel('Input p');
ylabel('Output a');
```



隐藏层中每个神经元的权重和偏置定义了径向基函数的位置和宽度。各个线性输出神经元形成了这些径向基函数的加权和。利用每层的正确权重和偏置值，以及足够的隐藏神经元，径向基网络可以以任何所需准确度拟合任何函数。以下是三个径向基函数（蓝色）经过缩放与求和后生成一个函数（品红色）的示例。

```
a2 = radbas(x-1.5);
a3 = radbas(x+2);
a4 = a + a2*1 + a3*0.5;
plot(x,a,'b-',x,a2,'b-',x,a3,'b-',x,a4,'m-')
title('Weighted Sum of Radial Basis Transfer Functions');
xlabel('Input p');
ylabel('Output a');
```



函数 NEWRB 可快速创建一个逼近由 P 和 T 定义的函数的径向基网络。除了训练集和目标，NEWRB 还使用了两个参数，分别为误差平方和目标与分布常数。

```
eg = 0.02; % sum-squared error goal
sc = 1; % spread constant
net = newrb(X,T,eg,sc);
```

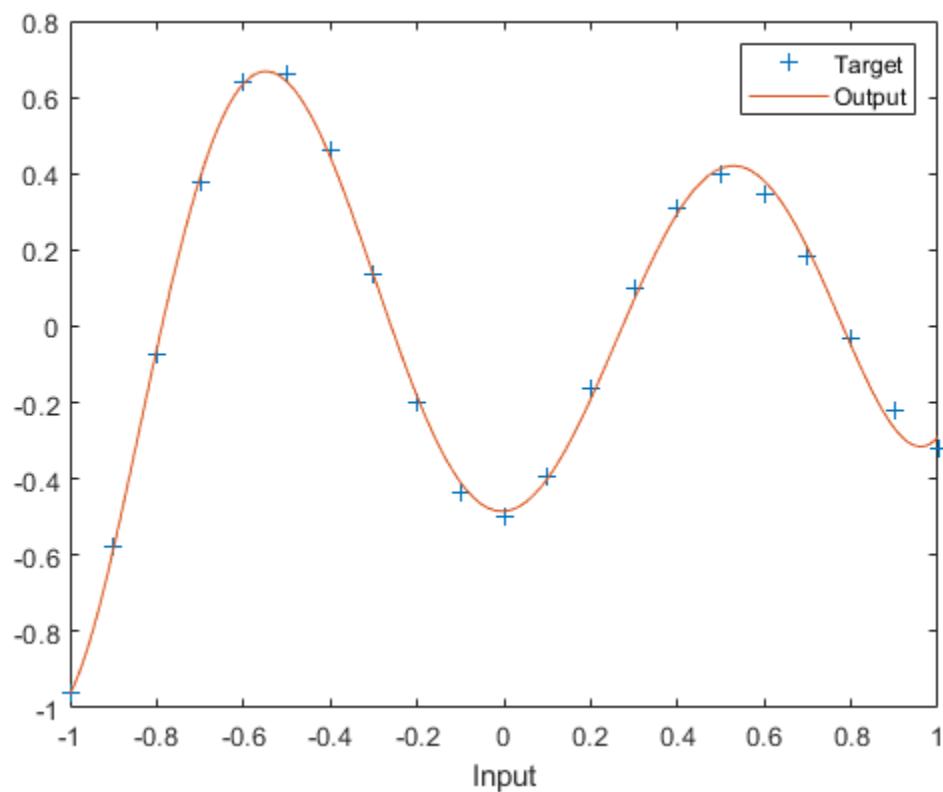
NEWRB, neurons = 0, MSE = 0.176192

要了解网络性能如何，请重新绘制训练集。然后仿真网络对相同范围内的输入的响应。最后，在同一图上绘制结果。

```
plot(X,T,'+');
xlabel('Input');

X = -1:.01:1;
Y = net(X);

hold on;
plot(X,Y);
hold off;
legend({'Target','Output'})
```

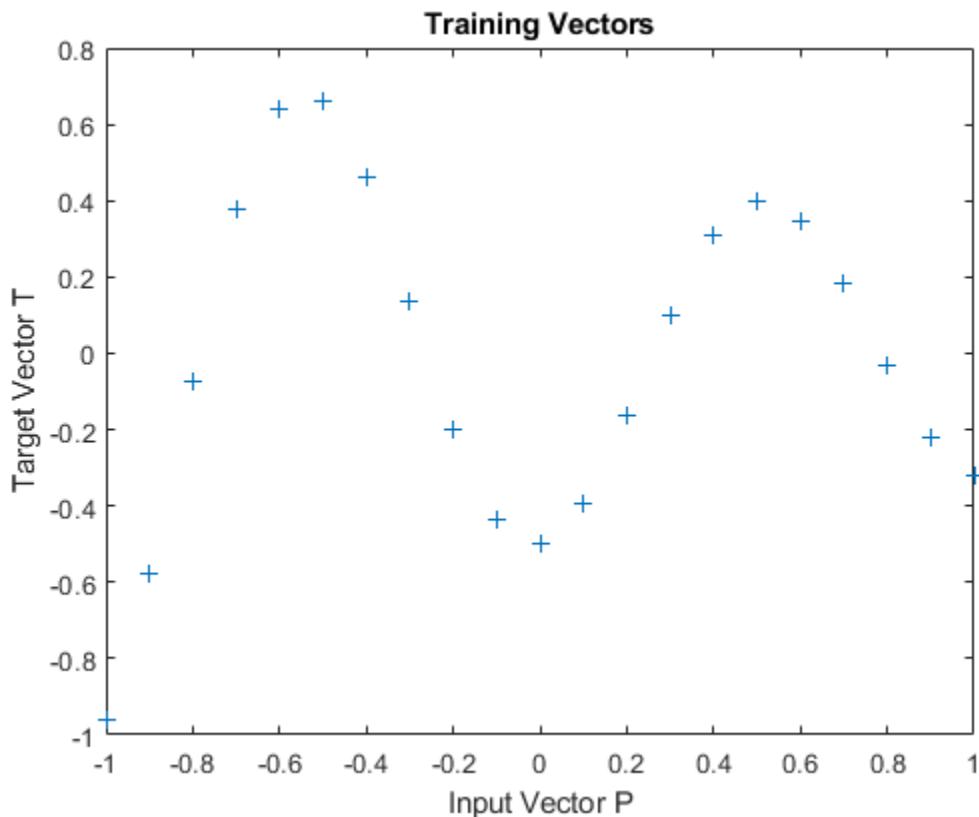


## 径向基神经元欠叠

径向基网络被训练为用目标输出响应特定输入。然而，因为径向基神经元的分布程度太低，网络需要许多神经元。

定义 21 个输入 P 和相关目标 T。

```
P = -1:1:1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 ...
       .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 ...
       .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201];
plot(P,T,'+');
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
```



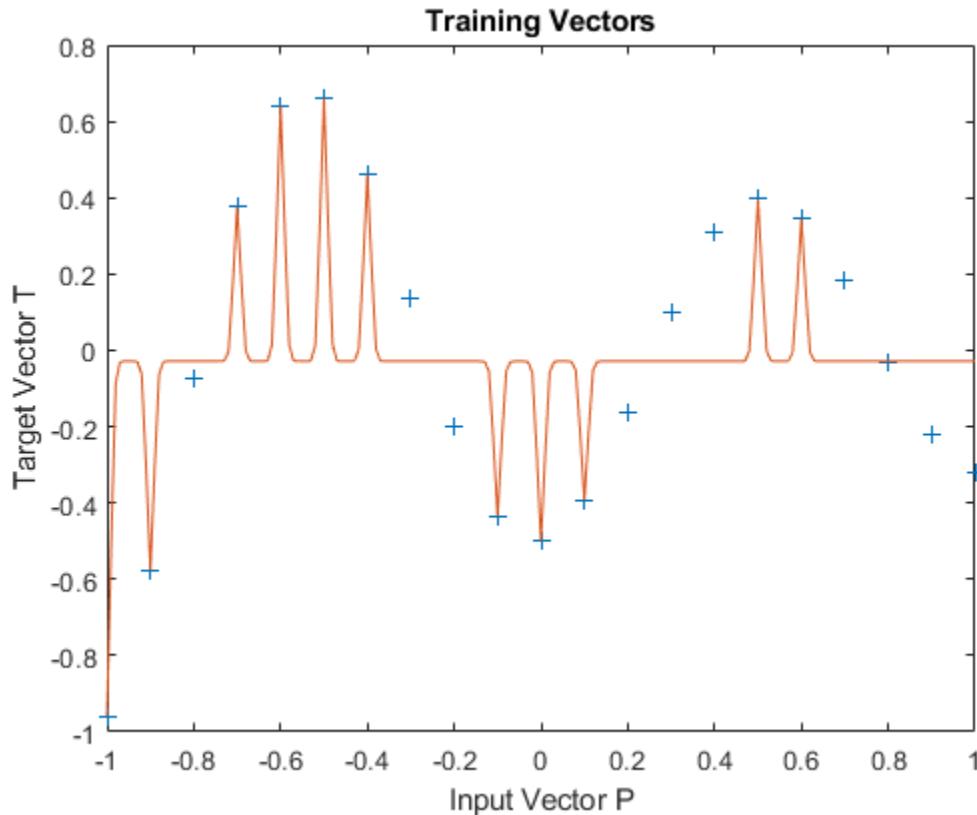
函数 NEWRB 可快速创建一个逼近由 P 和 T 定义的函数的径向基网络。除了训练集和目标，NEWRB 还使用了两个参数，分别为误差平方和目标与分布常数。径向基神经元的分布设置为非常小的数量。

```
eg = 0.02; % sum-squared error goal
sc = .01; % spread constant
net = newrb(P,T,eg,sc);
```

NEWRB, neurons = 0, MSE = 0.176192

要检查网络是否以平滑方式拟合该函数，请定义另一组测试输入向量，并用这些新输入对网络进行仿真。将结果绘制在与训练集相同的图上。测试向量显示该函数已过拟合！如果有更高的分布常数，网络可以做得更好。

```
X = -1:.01:1;
Y = net(X);
hold on;
plot(X,Y);
hold off;
```

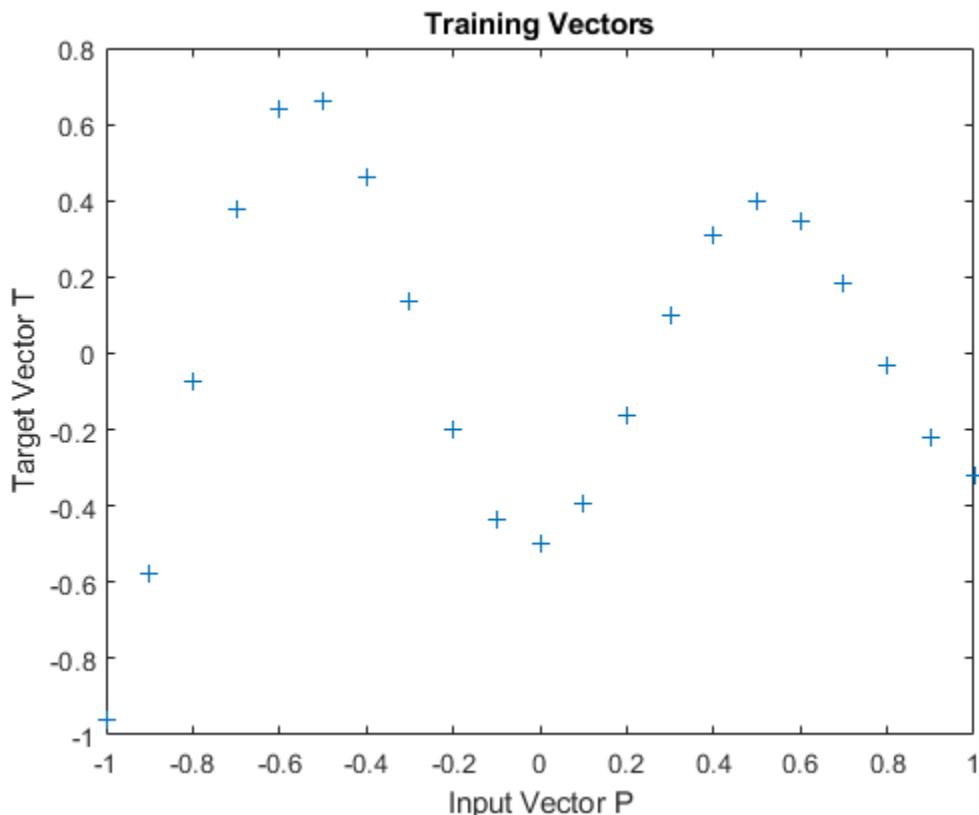


## 径向基神经元过叠

径向基网络被训练为用目标输出响应特定输入。然而，由于径向基神经元的分布程度太高，每个神经元的响应基本相同，因此无法设计网络。

定义 21 个输入 P 和相关目标 T。

```
P = -1:.1:1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 ...
    .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988 ...
    .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201];
plot(P,T,'+');
title('Training Vectors');
xlabel('Input Vector P');
ylabel('Target Vector T');
```



函数 NEWRB 可快速创建一个逼近由 P 和 T 定义的函数的径向基网络。

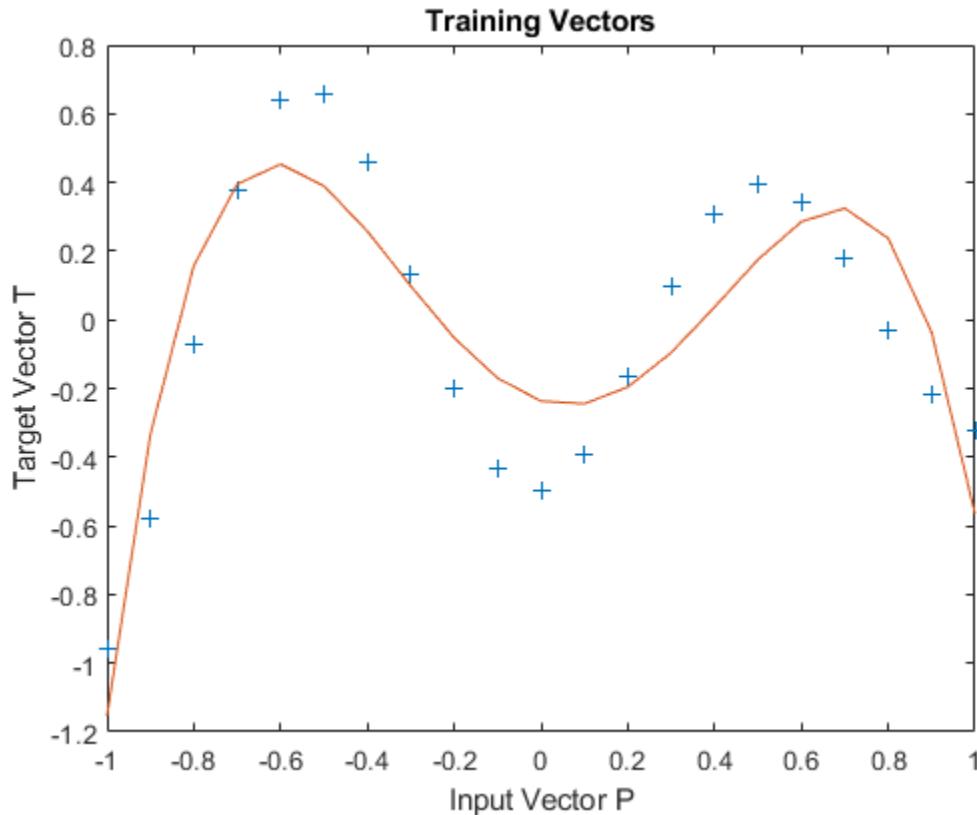
除了训练集和目标，NEWRB 还使用了两个参数，分别为误差平方和目标与分布常数。径向基神经元的分布设置为非常大的数量。

```
eg = 0.02; % sum-squared error goal
sc = 100; % spread constant
net = newrb(P,T,eg,sc);
```

NEWRB, neurons = 0, MSE = 0.176192

由于径向基神经元的输入区域有很大的重叠，NEWRB 无法正确设计径向基网络。所有神经元始终输出 1，因此不能用于产生不同响应。要查看网络在训练集上的表现，请使用原始输入对网络进行仿真。将结果绘制在与训练集相同的图上。

```
Y = net(P);
hold on;
plot(P,Y);
hold off;
```



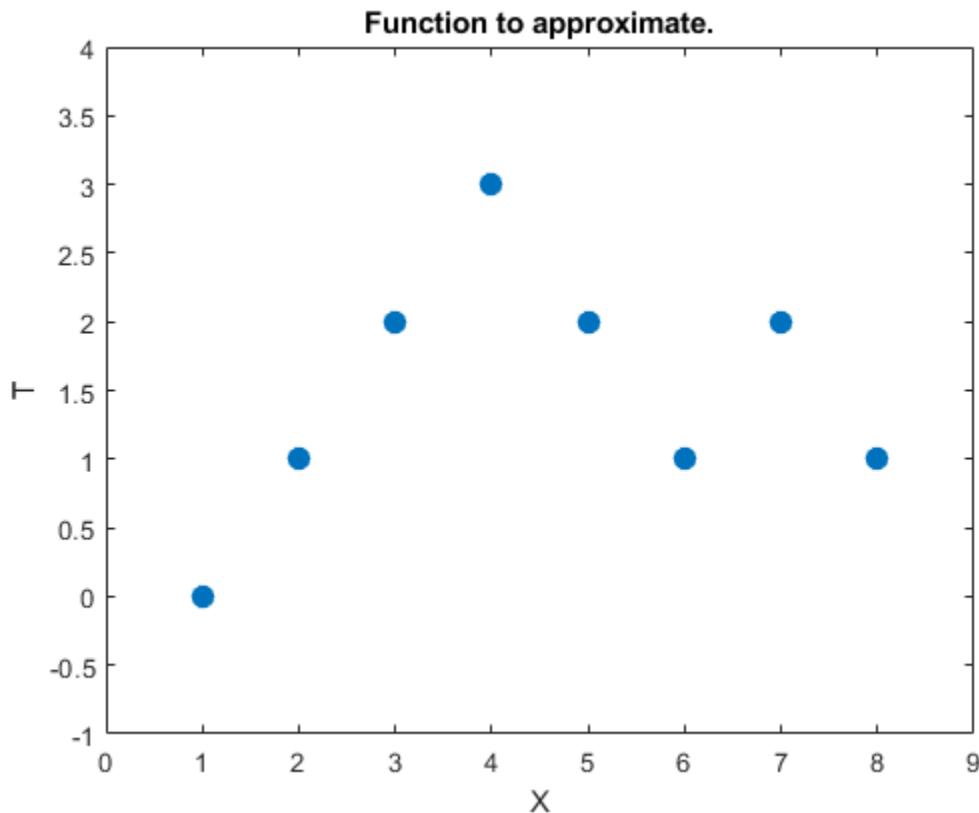
## GRNN 函数逼近

此示例使用 NEWGRNN 和 SIM 函数。

此处我们要拟合  $y$  函数的八个数据点。函数输入  $X$  应该导致目标输出  $T$ 。

```
X = [1 2 3 4 5 6 7 8];
T = [0 1 2 3 2 1 2 1];
```

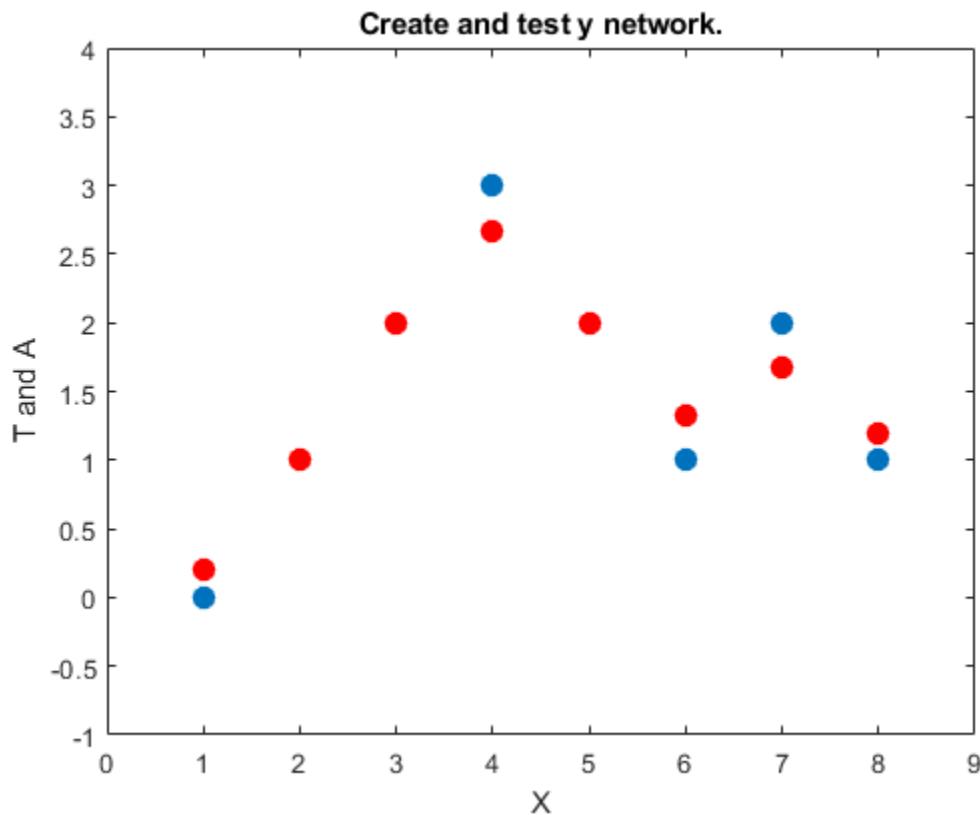
```
plot(X,T,'.','markersize',30)
axis([0 9 -1 4])
title('Function to approximate.')
xlabel('X')
ylabel('T')
```



我们使用 NEWGRNN 创建  $y$  广义回归网络。我们使用略低于 1 的  $y$  SPREAD (即输入值之间的距离) ,以便得到与各数据点拟合程度很高的  $y$  函数。较小的分布能更好地拟合数据,但不太平滑。

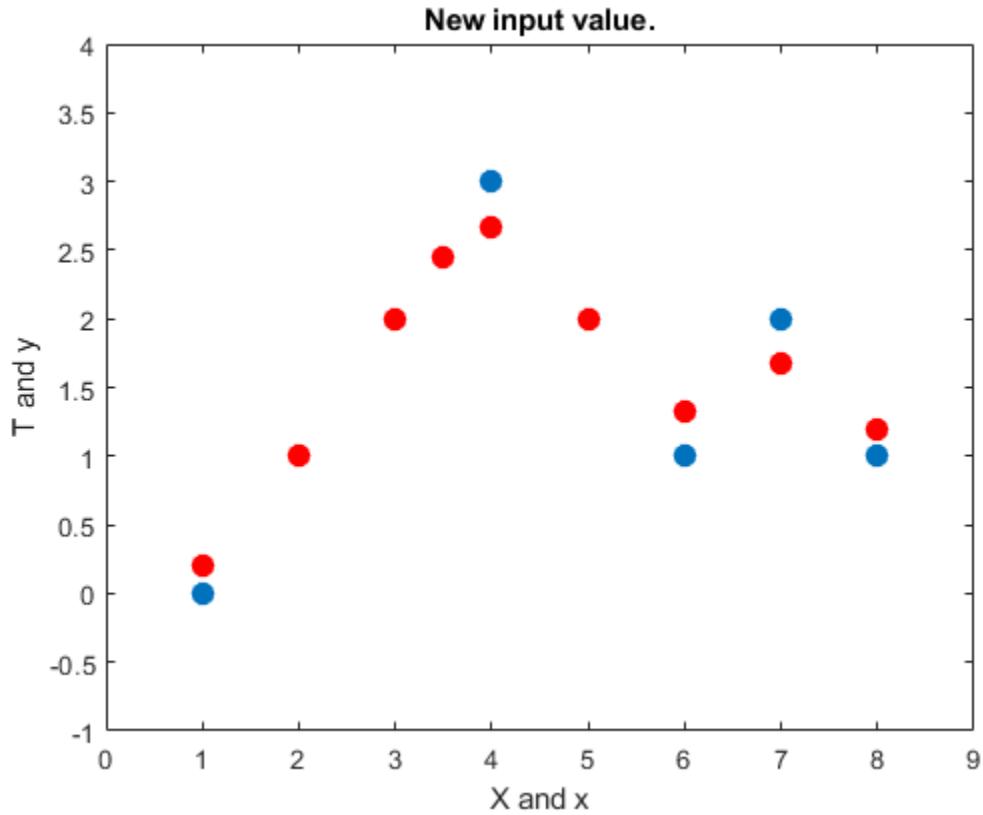
```
spread = 0.7;
net = newgrnn(X,T,spread);
A = net(X);

hold on
outputline = plot(X,A,'.','markersize',30,'color',[1 0 0]);
title('Create and test y network.')
xlabel('X')
ylabel('T and A')
```



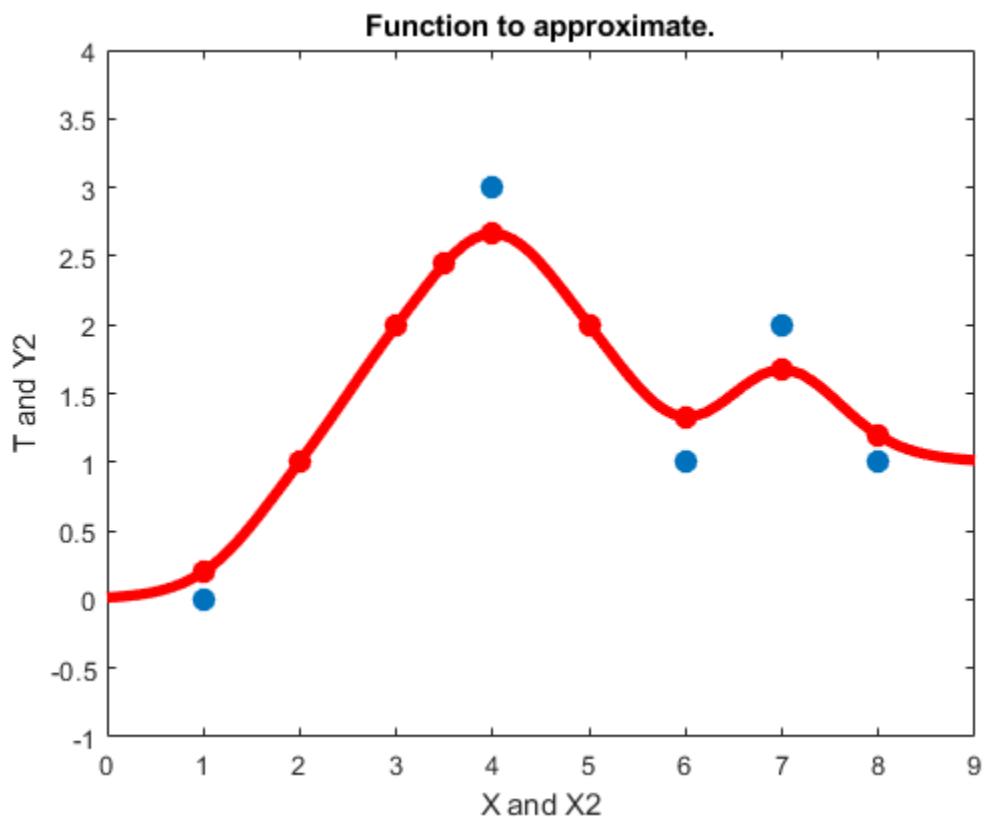
我们可以用网络在  $y$  新输入值处逼近函数。

```
x = 3.5;
y = net(x);
plot(x,y,'.','markersize',30,'color',[1 0 0]);
title('New input value.')
xlabel('X and x')
ylabel('T and y')
```



在此处，针对许多值对网络的响应进行仿真，从而允许我们看到它所表示的函数。

```
X2 = 0:.1:9;
Y2 = net(X2);
plot(X2,Y2,'linewidth',4,'color',[1 0 0])
title('Function to approximate.')
xlabel('X and X2')
ylabel('T and Y2')
```

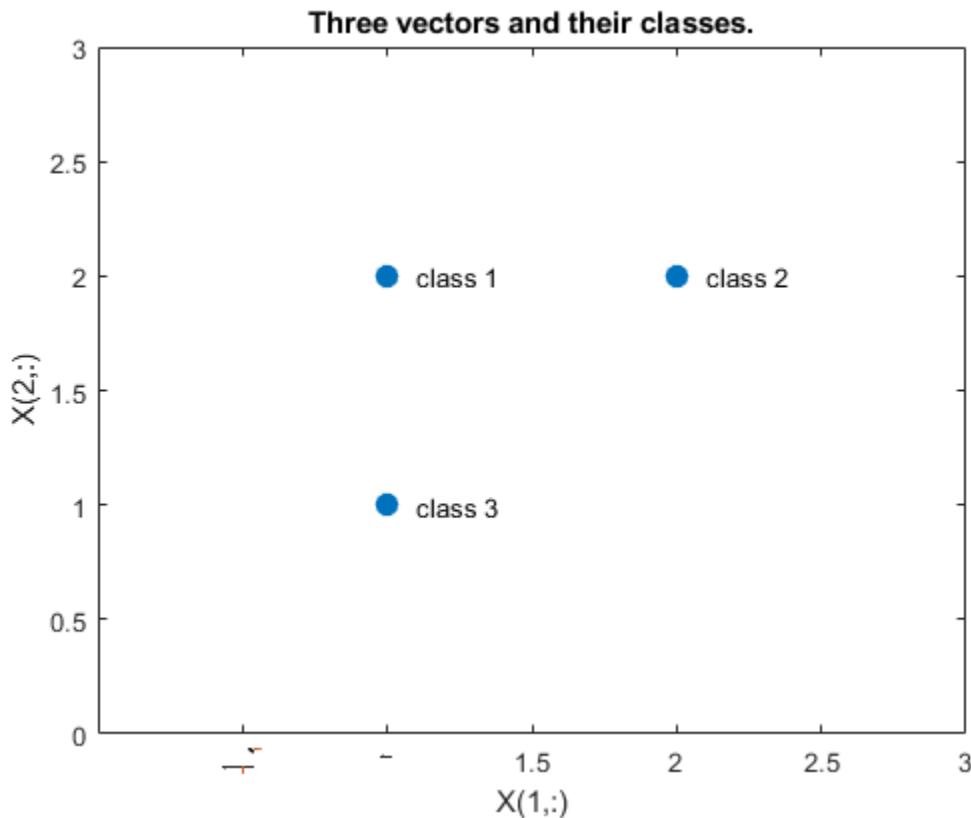


## PNN 分类

此示例使用 NEWPNN 和 SIM 函数。

此处有三个二元输入向量  $X$  和它们相关联的类  $Tc$ 。我们想创建  $y$  概率神经网络，对这些向量正确分类。

```
X = [1 2; 2 2; 1 1];
Tc = [1 2 3];
plot(X(1,:),X(2,:),'.','markersize',30)
for i = 1:3, text(X(1,i)+0.1,X(2,i),sprintf('class %g',Tc(i))), end
axis([0 3 0 3])
title('Three vectors and their classes.')
xlabel('X(1,:)')
ylabel('X(2,:)'')
```



首先，我们将目标类索引  $Tc$  转换为向量  $T$ 。然后，我们用 NEWPNN 设计  $y$  概率神经网络。我们使用  $y$  SPREAD 值 1，因为这是输入向量之间的  $y$  典型距离。

```
T = ind2vec(Tc);
spread = 1;
net = newpnn(X,T,spread);
```

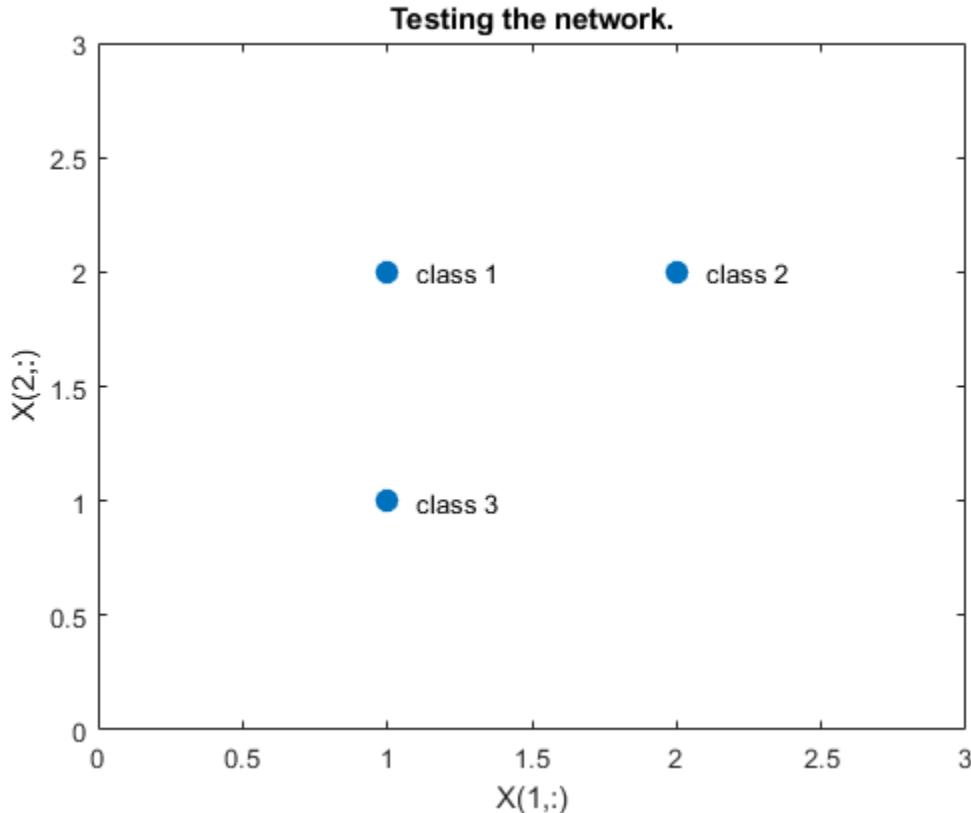
现在我们基于设计输入向量测试网络。我们通过对网络进行仿真并将其向量输出转换为索引来实现此目的。

```
Y = net(X);
Yc = vec2ind(Y);
```

```

plot(X(1,:),X(2,:),'!', 'markersize',30)
axis([0 3 0 3])
for i = 1:3, text(X(1,i)+0.1,X(2,i),sprintf('class %g',Yc(i))),end
title('Testing the network.')
xlabel('X(1,:)')
ylabel('X(2,:)')

```

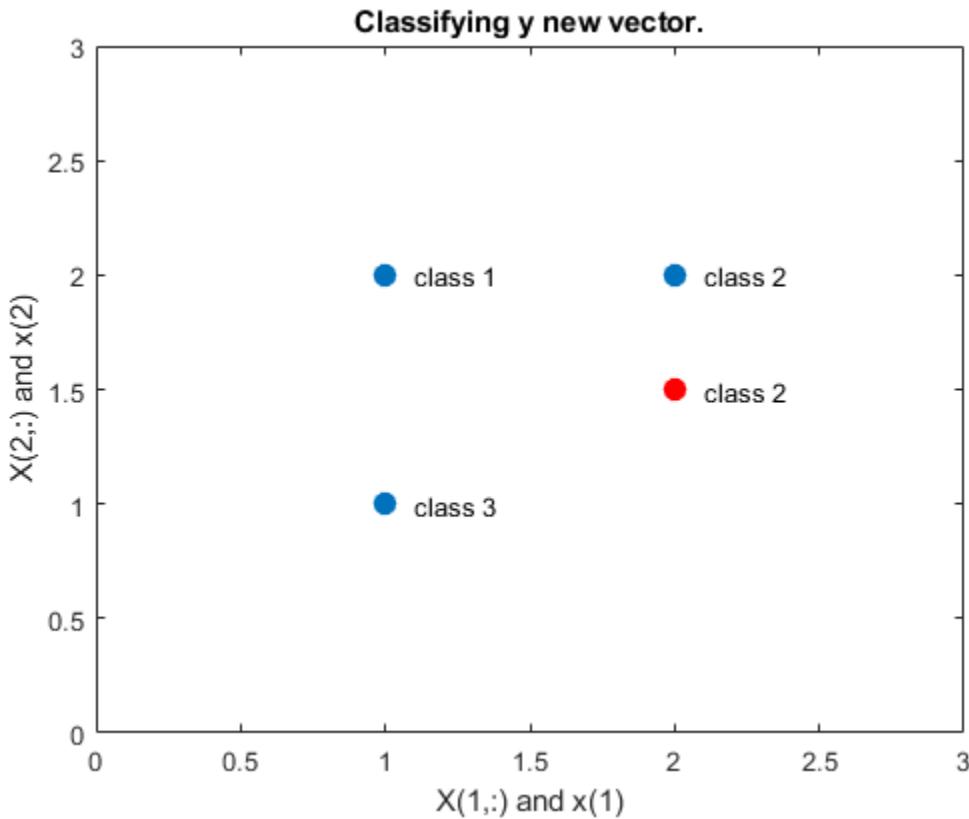


让我们用我们的网络对  $y$  新向量进行分类。

```

x = [2; 1.5];
y = net(x);
ac = vec2ind(y);
hold on
plot(x(1),x(2),'!', 'markersize',30,'color',[1 0 0])
text(x(1)+0.1,x(2),sprintf('class %g',ac))
hold off
title('Classifying y new vector.')
xlabel('X(1,:) and x(1)')
ylabel('X(2,:) and x(2)')

```

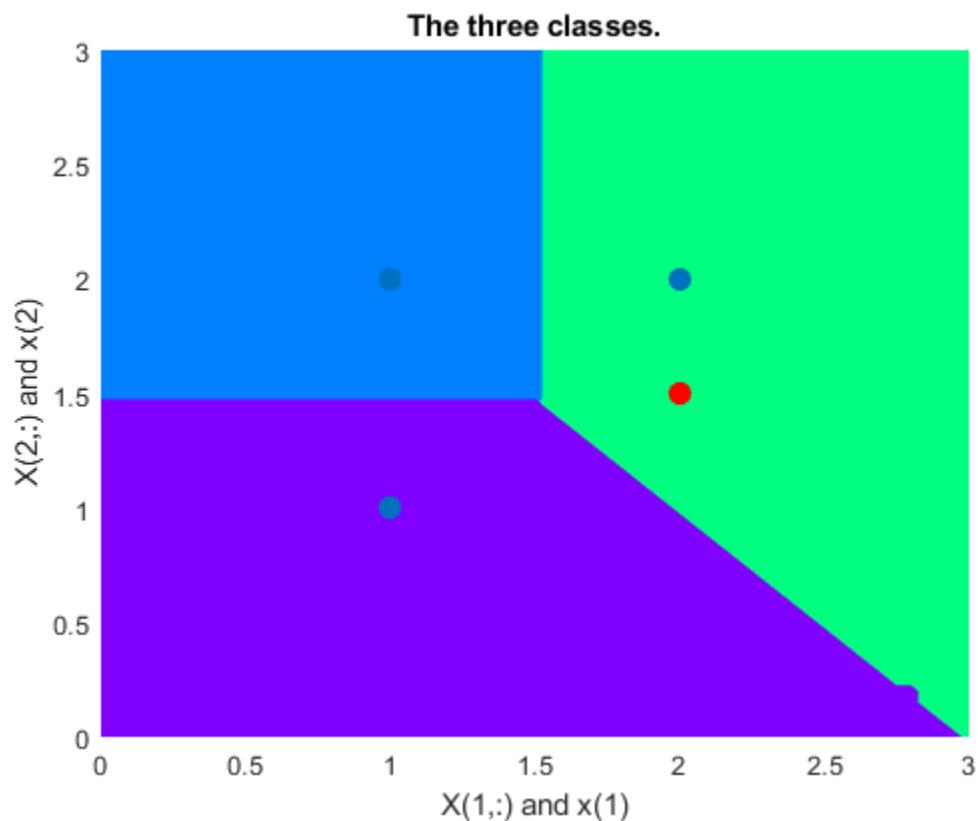


该图显示概率神经网络如何将输入空间分为三个类。

```

x1 = 0:.05:3;
x2 = x1;
[X1,X2] = meshgrid(x1,x2);
xx = [X1(:) X2(:)]';
yy = net(xx);
yy = full(yy);
m = mesh(X1,X2,reshape(yy(1,:),length(x1),length(x2)));
m.FaceColor = [0 0.5 1];
m.LineStyle = 'none';
hold on
m = mesh(X1,X2,reshape(yy(2,:),length(x1),length(x2)));
m.FaceColor = [0 1.0 0.5];
m.LineStyle = 'none';
m = mesh(X1,X2,reshape(yy(3,:),length(x1),length(x2)));
m.FaceColor = [0.5 0 1];
m.LineStyle = 'none';
plot3(X(1,:),X(2,:),[1 1 1]+0.1,'','markersize',30)
plot3(x(1),x(2),1.1,'','markersize',30,'color',[1 0 0])
hold off
view(2)
title('The three classes.')
xlabel('X(1,:)' and 'x(1)')
ylabel('X(2,:)' and 'x(2)')

```



## 学习向量量化

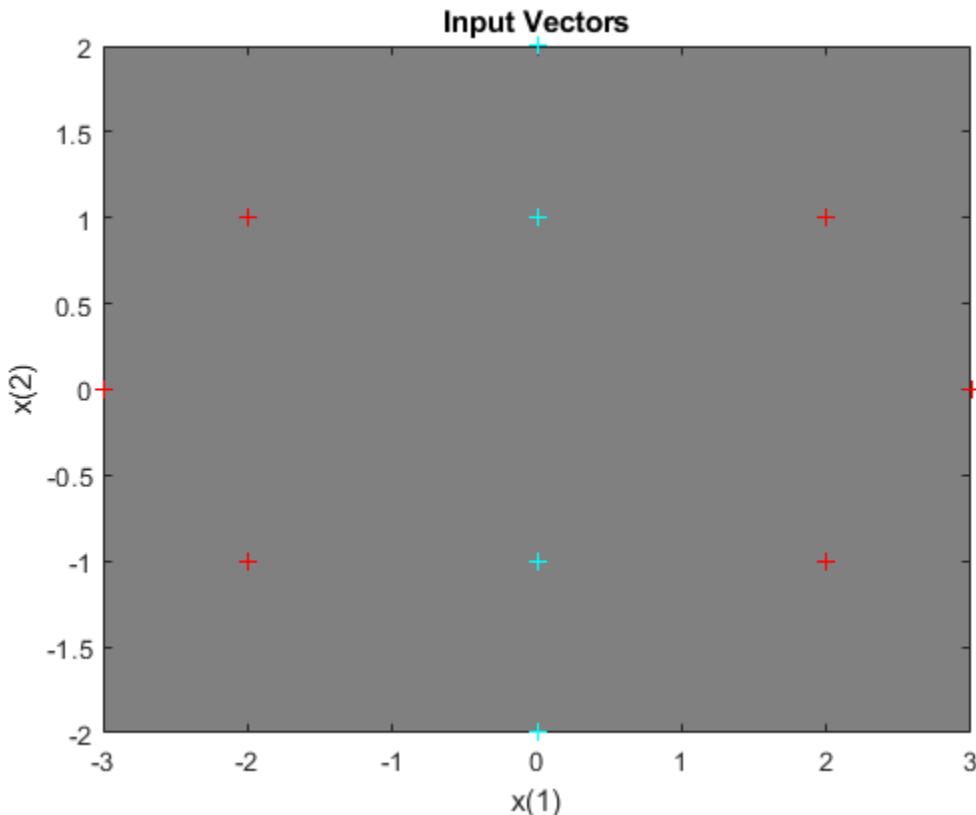
LVQ 网络训练为根据给定目标对输入向量进行分类。

令  $X$  为 10 个 2 元素样本输入向量， $C$  为这些向量所属的类。这些类可以通过 IND2VEC 变换为用作目标  $T$  的向量。

```
x = [-3 -2 -2  0  0  0  0 +2 +2 +3;
      0 +1 -1 +2 +1 -1 -2 +1 -1  0];
c = [1 1 1 2 2 2 2 1 1 1];
t = ind2vec(c);
```

下面绘制了这些数据点。红色 = 第 1 类，青色 = 第 2 类。LVQ 网络表示具有隐藏神经元的向量聚类，并将这些聚类与输出神经元组合在一起以形成期望的类。

```
colormap(hsv);
plotvec(x,c)
title('Input Vectors');
xlabel('x(1)');
ylabel('x(2)');
```

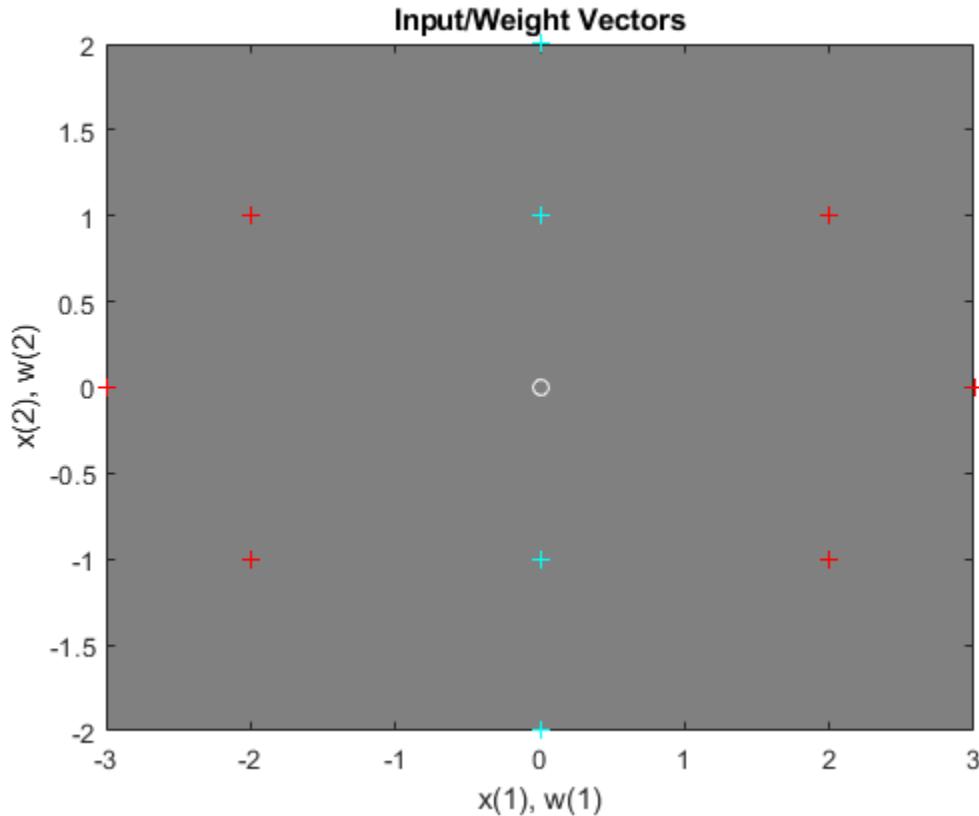


在以下代码中，LVQNET 创建了一个具有四个隐藏神经元的 LVQ 层，学习率为 0.1。然后针对输入  $X$  和目标  $T$  配置网络。（配置通常不是必要步骤，因为 TRAIN 会自动完成配置。）

```
net = lvqnet(4,0.1);
net = configure(net,x,t);
```

按如下方式绘制竞争神经元权重向量。

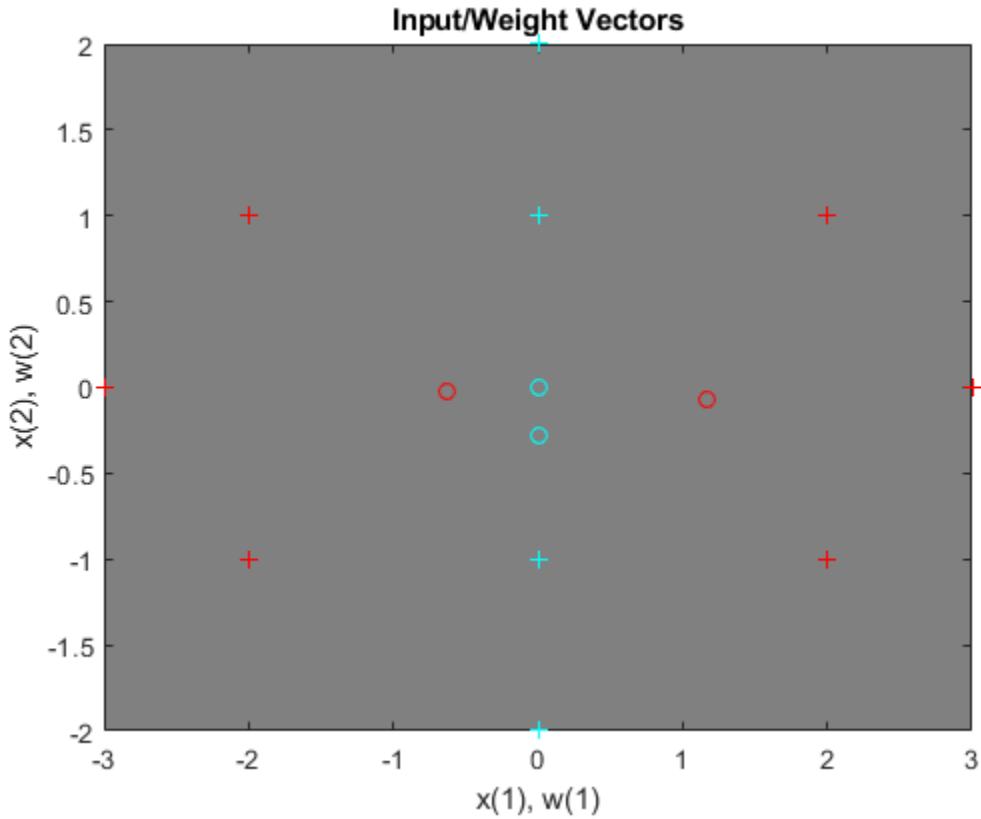
```
hold on
w1 = net.IW{1};
plot(w1(1,1),w1(1,2),'ow')
title('Input/Weight Vectors');
xlabel('x(1), w(1)');
ylabel('x(2), w(2)');
```



要训练网络，首先改写默认的训练轮数，然后训练网络。训练完成后，重新绘制输入向量“+”和竞争神经元的权重向量“o”。红色 = 第1类，青色 = 第2类。

```
net.trainParam.epochs=150;
net=train(net,x,t);

cla;
plotvec(x,c);
hold on;
plotvec(net.IW{1}',vec2ind(net.LW{2}),'o');
```



现在使用 LVQ 网络作为分类器，其中每个神经元都对应于一个不同的类别。提交输入向量  $[0.2; 1]$ 。红色 = 第 1 类，青色 = 第 2 类。

```
x1 = [0.2; 1];
y1 = vec2ind(net(x1))
```

y1 = 2

# 线性预测设计

此示例说明如何设计线性神经元来预测给定最后五个值的时序中的下一个值。

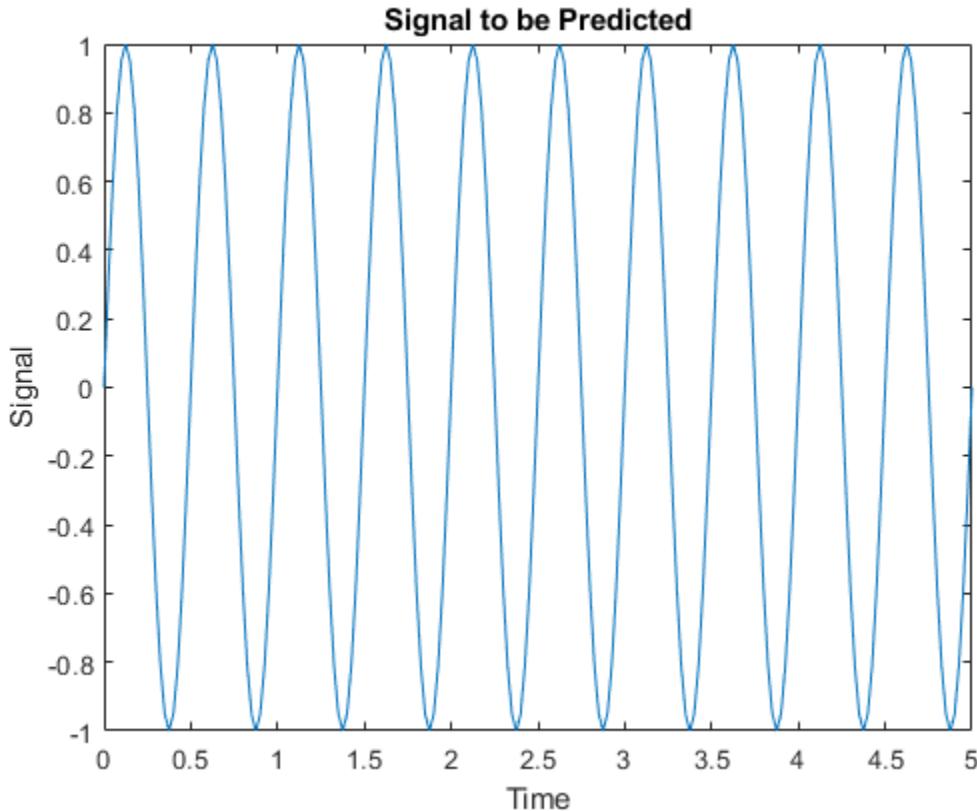
## 定义波形

此处，时间定义为 0 到 5 秒，步长为 1/40 秒。

```
time = 0:0.025:5;
```

我们可以定义关于时间的信号。

```
signal = sin(time*4*pi);
plot(time,signal)
xlabel('Time');
ylabel('Signal');
title('Signal to be Predicted');
```



## 为神经网络设置问题

然后将信号转换为元胞数组。神经网络将时间步表示为一个元胞数组的各列，并将它们与在给定时间的不同样本区分开来，后者用矩阵列表示。

```
signal = con2seq(signal);
```

为了设置此问题，我们将使用信号的前四个值作为初始输入延迟状态，其余的值（最后一个时间步除外）作为输入。

```

Xi = signal(1:4);
X = signal(5:(end-1));
timex = time(5:(end-1));

```

目标现在定义为匹配输入，但前移一个时间步。

```
T = signal(6:end);
```

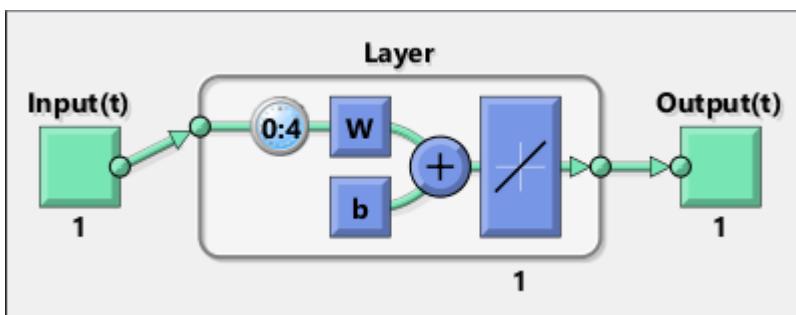
### 设计线性层

函数 **newlind** 现在将设计具有单个神经元的线性层，该层在给定当前值和四个过去值的情况下预测信号的下一个时间步。

```

net = newlind(X,T,Xi);
view(net)

```



### 测试线性层

现在，可以对输入和延迟状态调用网络（就像调用函数一样），以获得它的时间响应。

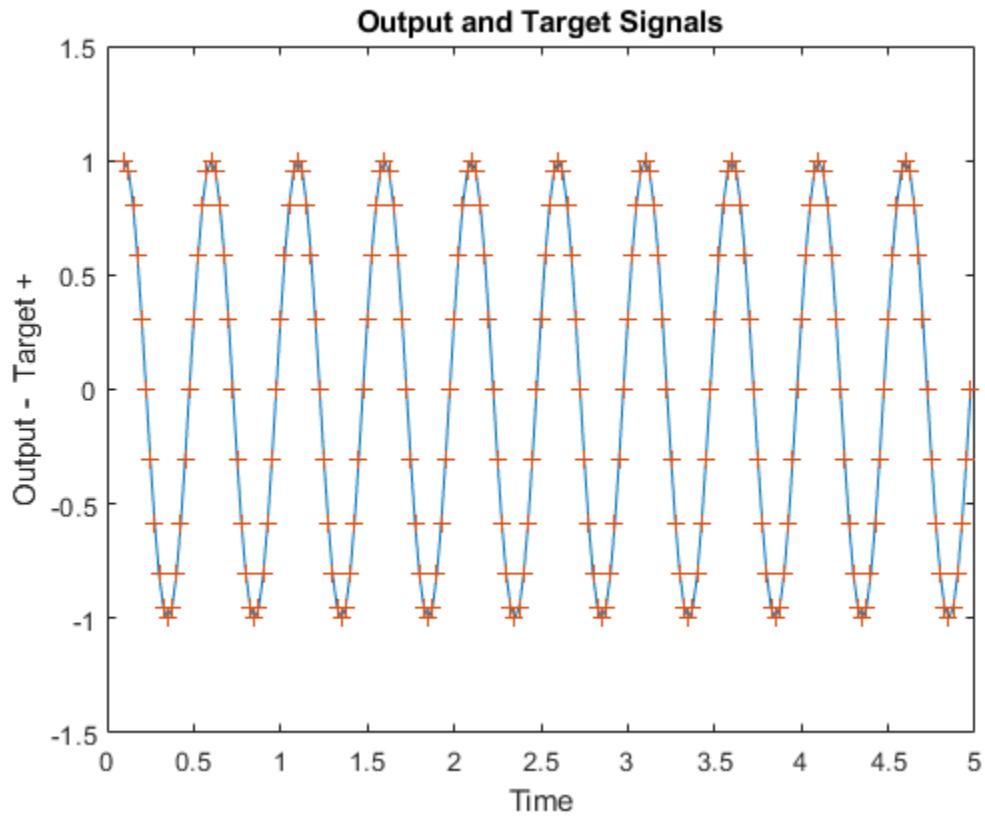
```
Y = net(X,Xi);
```

绘制输出信号与目标。

```

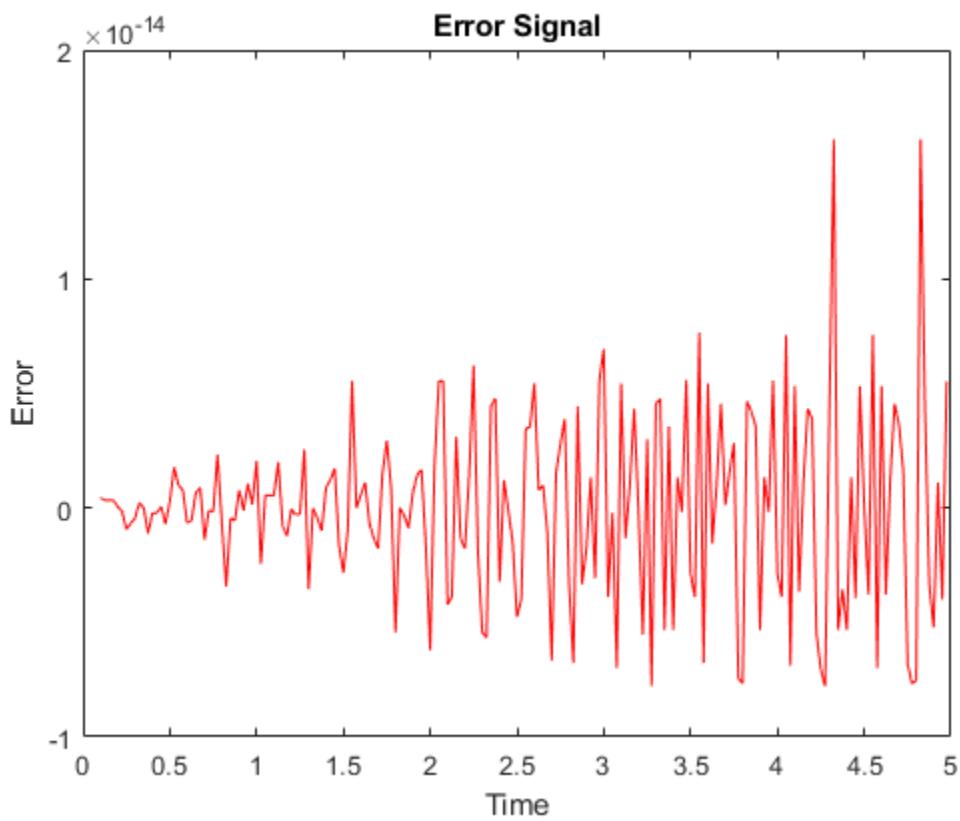
figure
plot(timex,cell2mat(Y),timex,cell2mat(T),'+')
xlabel('Time');
ylabel('Output - Target +');
title('Output and Target Signals');

```



还可以绘制误差。

```
figure  
E = cell2mat(T)-cell2mat(Y);  
plot(timeX,E,'r')  
hold off  
xlabel('Time');  
ylabel('Error');  
title('Error Signal');
```



注意误差有多小！

此示例说明如何设计动态线性网络，它可以根据当前值和过去的值预测信号的下一个值。

## 自适应线性预测

此示例说明自适应线性层如何学习在给定当前值和最后四个值的情况下，预测信号中的下一个值。

要了解如何使用深度学习网络预测时序数据，请参阅“使用深度学习进行时序预测”（第 4-9 页）。

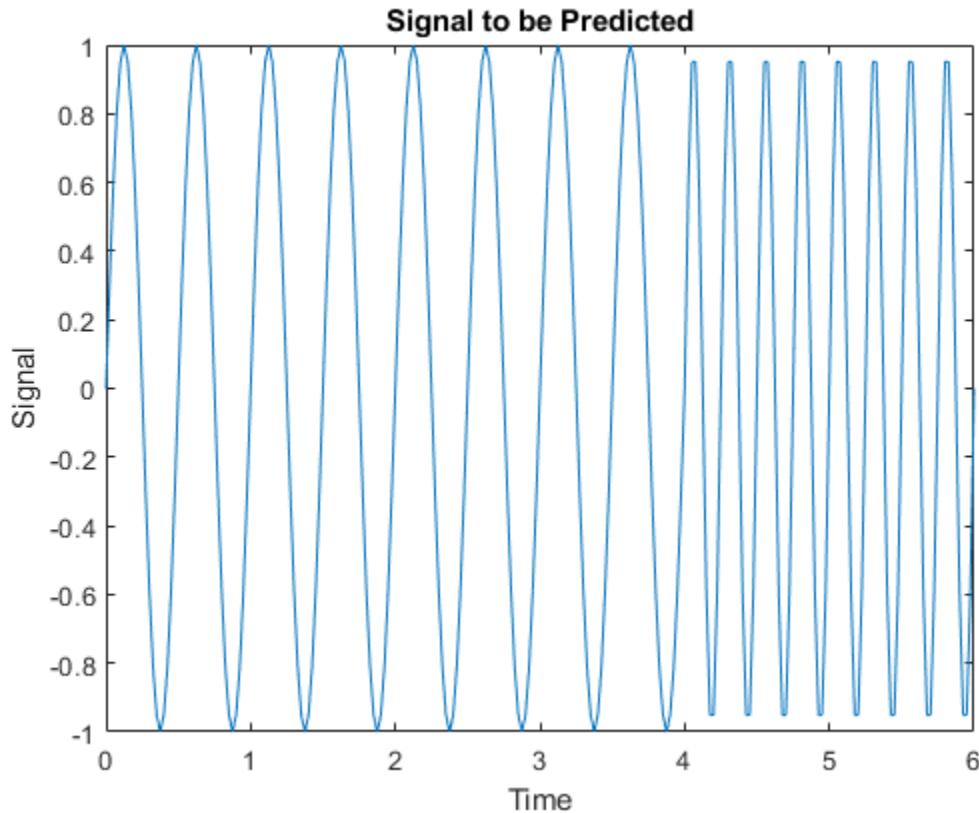
### 定义波形

此处，两个时间段定义为 0 到 6 秒，步长为  $1/40$  秒。

```
time1 = 0:0.025:4; % from 0 to 4 seconds
time2 = 4.025:0.025:6; % from 4 to 6 seconds
time = [time1 time2]; % from 0 to 6 seconds
```

以下信号从一个频率开始，然后转换到另一个频率。

```
signal = [sin(time1*4*pi) sin(time2*8*pi)];
plot(time,signal)
xlabel('Time');
ylabel('Signal');
title('Signal to be Predicted');
```



### 为神经网络设置问题

然后将信号转换为元胞数组。神经网络将时间步表示为一个元胞数组的各列，并将它们与在给定时间的不同样本区分开来，后者用矩阵列表示。

```
signal = con2seq(signal);
```

为了设置此问题，我们将使用信号的前五个值作为初始输入延迟状态，其余的值作为输入。

```
Xi = signal(1:5);
X = signal(6:end);
timex = time(6:end);
```

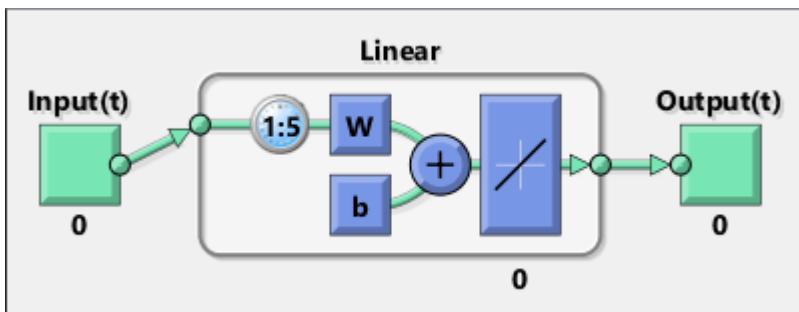
目标现在定义为匹配输入。网络仅使用最后五个值来预测当前输入。

```
T = signal(6:end);
```

### 创建线性层

函数 **linearlayer** 创建一个线性层，该层由一个神经元组成，最后五个输入具有抽头延迟。

```
net = linearlayer(1:5,0.1);
view(net)
```



### 自适应线性层

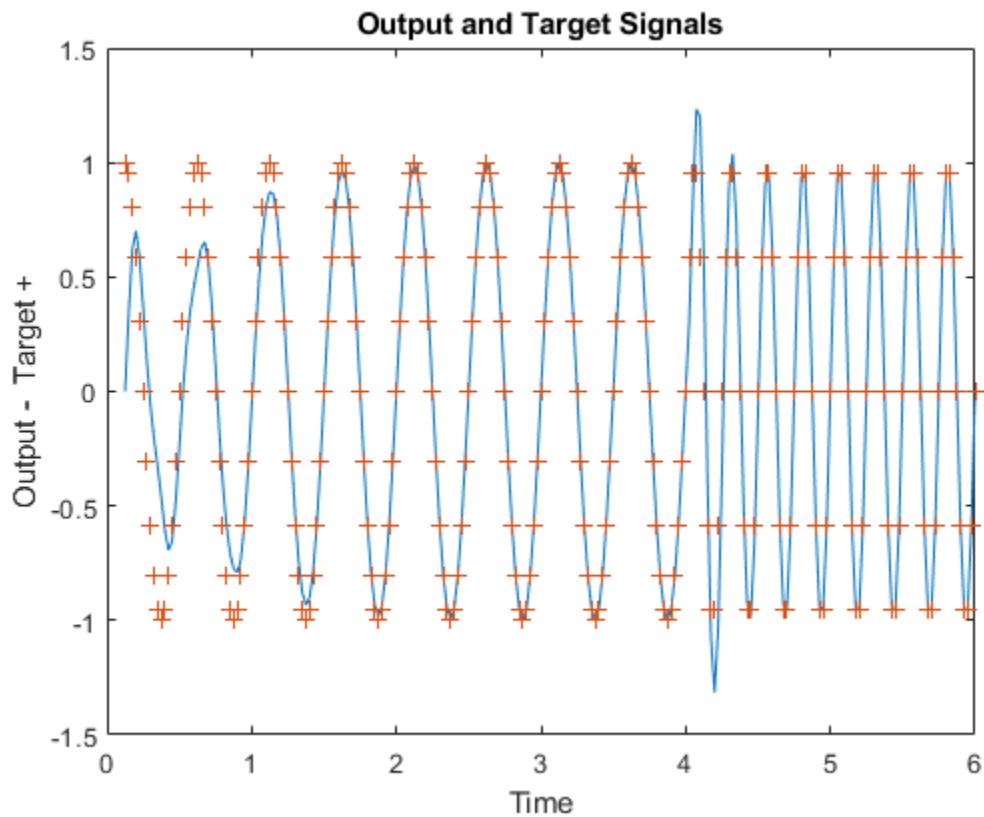
The function \*adapt\* simulates the network on the input, while adjusting its weights and biases after each timestep in response to how closely its output matches the target.

It returns the update networks, its outputs, and its errors.

```
[net,Y] = adapt(net,X,T,Xi);
```

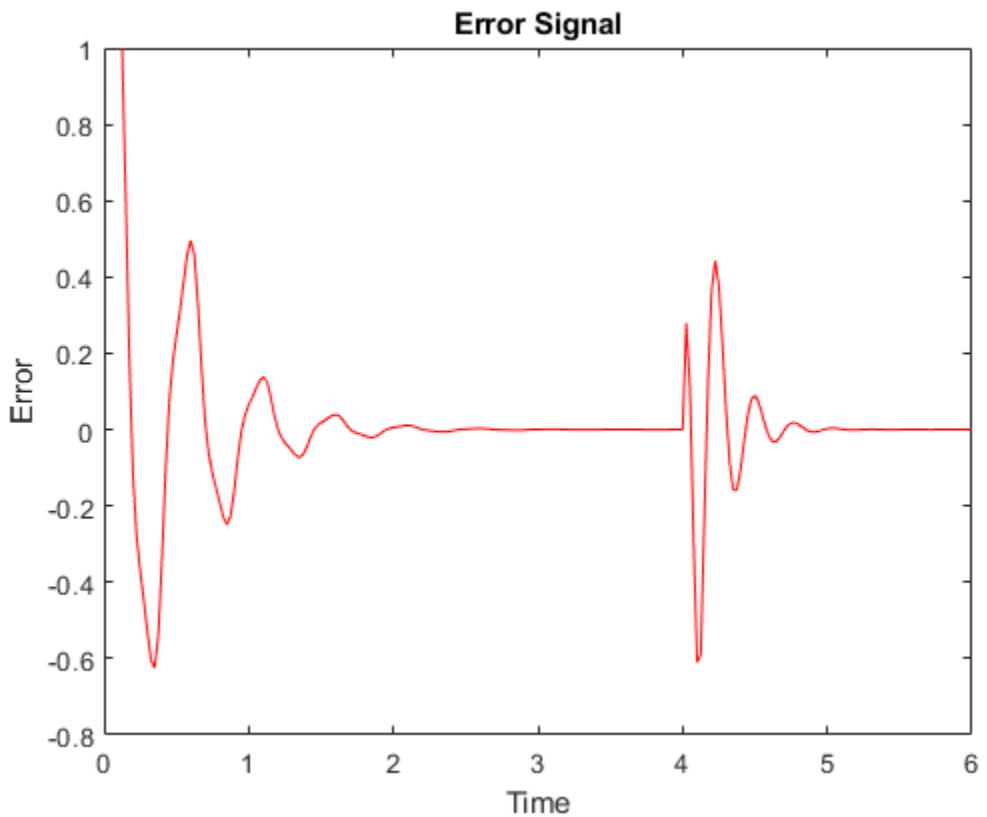
绘制输出信号与目标。

```
figure
plot(timex,cell2mat(Y),timex,cell2mat(T),'+')
xlabel('Time');
ylabel('Output - Target +');
title('Output and Target Signals');
```



还可以绘制误差。

```
figure  
E = cell2mat(T)-cell2mat(Y);  
plot(time,E,'r')  
hold off  
xlabel('Time');  
ylabel('Error');  
title('Error Signal');
```



请注意除初始误差之外的误差有多小，而且网络会在系统转换开始时和转换后学习系统行为。

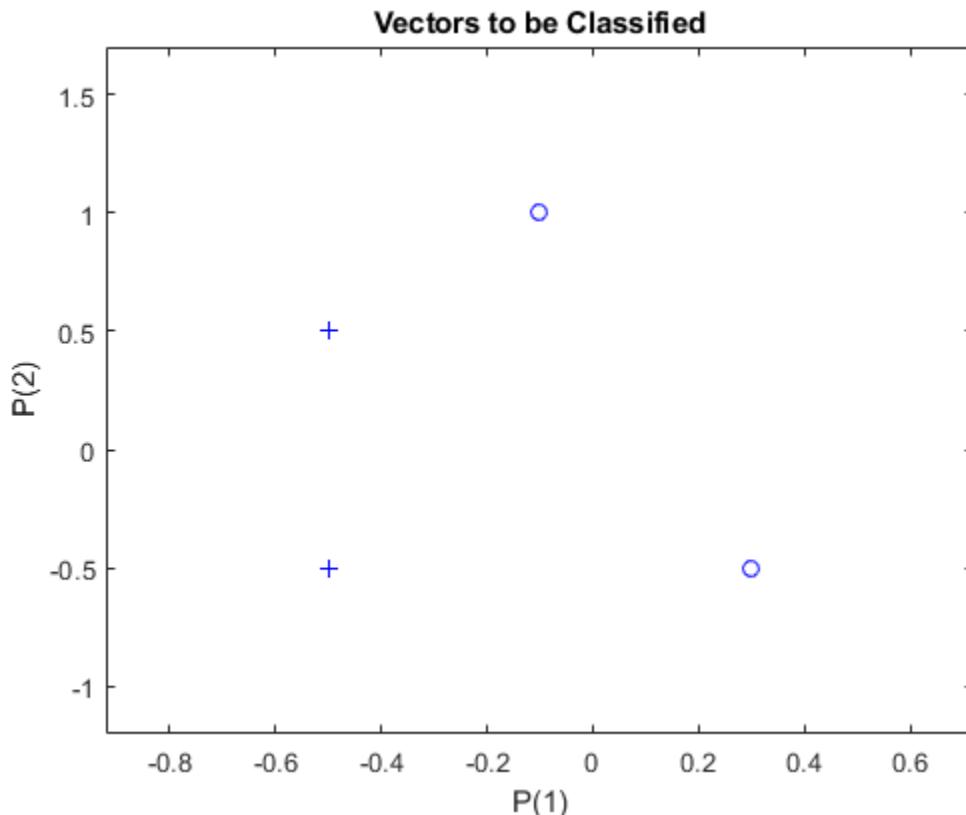
此示例说明如何对自适应线性网络进行仿真，该网络可以根据当前值和过去的值预测信号的下一个值，而不管信号行为如何变化。

## 用双输入感知器分类

双输入硬限制神经元被训练为将四个输入向量划分为两个类别。

$X$  中的四个列向量中的每一个都定义了一个二元素输入向量，行向量  $T$  定义了向量的目标类别。我们可以使用 PLOTPV 绘制这些向量。

```
X = [ -0.5 -0.5 +0.3 -0.1; ...
      -0.5 +0.5 -0.5 +1.0];
T = [1 1 0 0];
plotpv(X,T);
```



感知器必须将  $X$  中的四个输入向量正确分类为由  $T$  定义的两个类别。感知器具有 HARDLIM 神经元。这些神经元能够用一条直线将输入空间分为两个类别（0 和 1）。

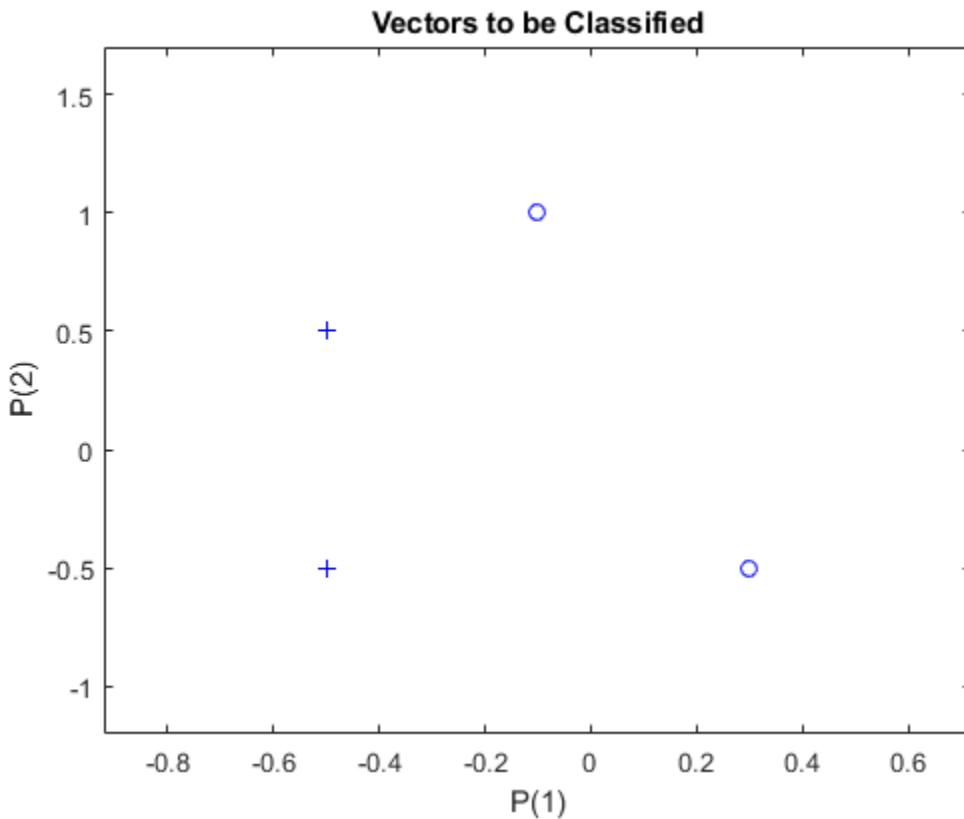
这里 PERCEPTRON 创建了一个具有单个神经元的新神经网络。然后针对数据配置网络，这样我们可以检查其初始权重和偏置值。（通常可以跳过配置步骤，因为 ADAPT 或 TRAIN 会自动完成配置。）

```
net = perceptron;
net = configure(net,X,T);
```

神经元初次尝试分类时，输入向量会被重新绘制。

初始权重设置为零，因此任何输入都会生成相同的输出，而且分类线甚至不会出现在图上。别担心...我们将对它进行训练！

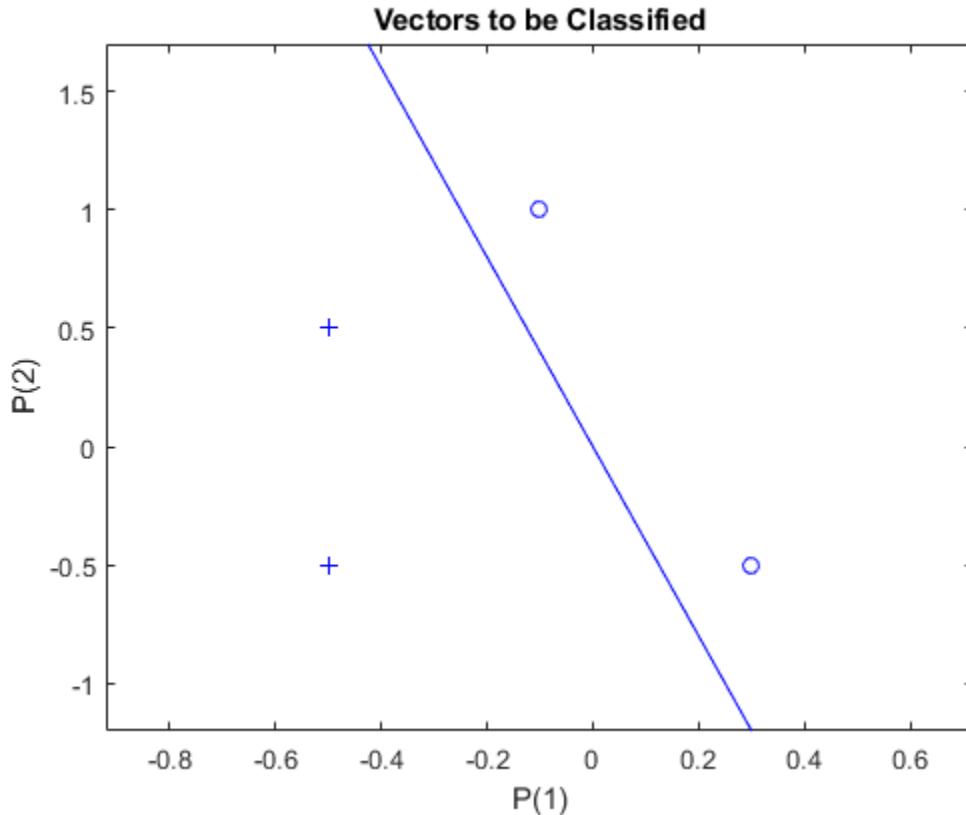
```
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
```



此处，输入数据和目标数据转换为顺序数据（元胞数组，其中每个列指示一个时间步）并复制三次以形成序列 XX 和 TT。

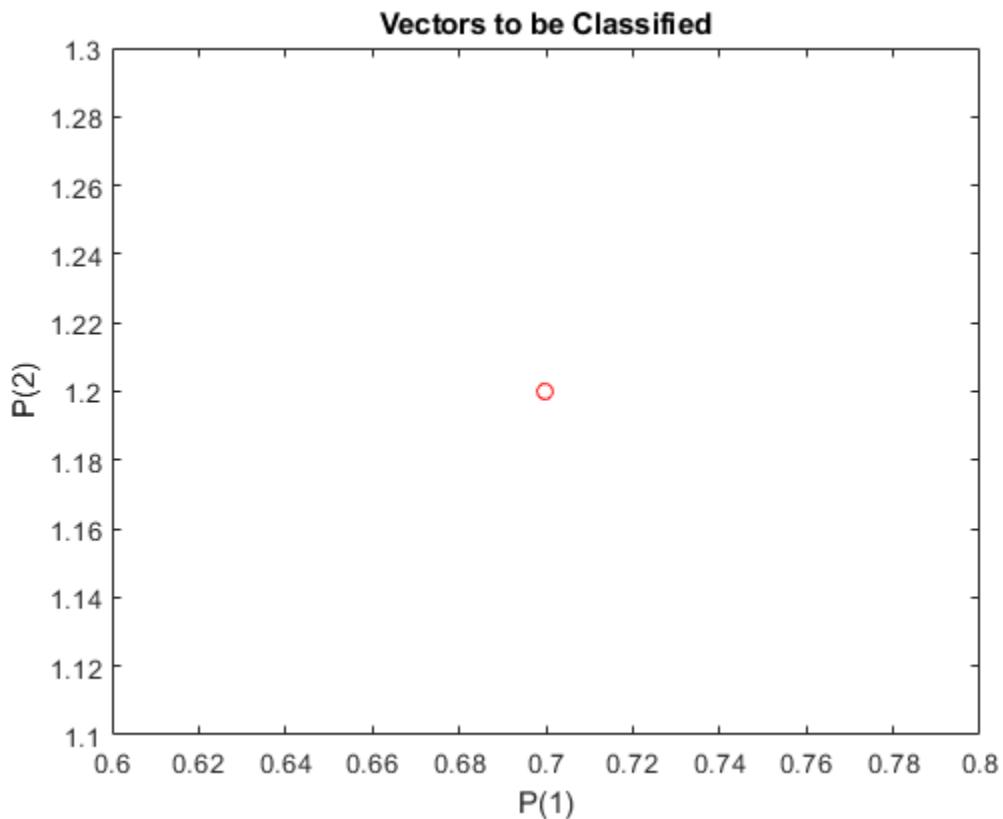
ADAPT 针对序列中的每个时间步更新网络，并返回一个作为更好的分类器执行的新网络对象。

```
XX = repmat(con2seq(X),1,3);
TT = repmat(con2seq(T),1,3);
net = adapt(net,XX,TT);
plotpc(net.IW{1},net.b{1});
```



现在 SIM 用于对任何其他输入向量（如 [0.7; 1.2]）进行分类。此新点及原始训练集的绘图显示了网络的性能。为了将其与训练集区分开来，将其显示为红色。

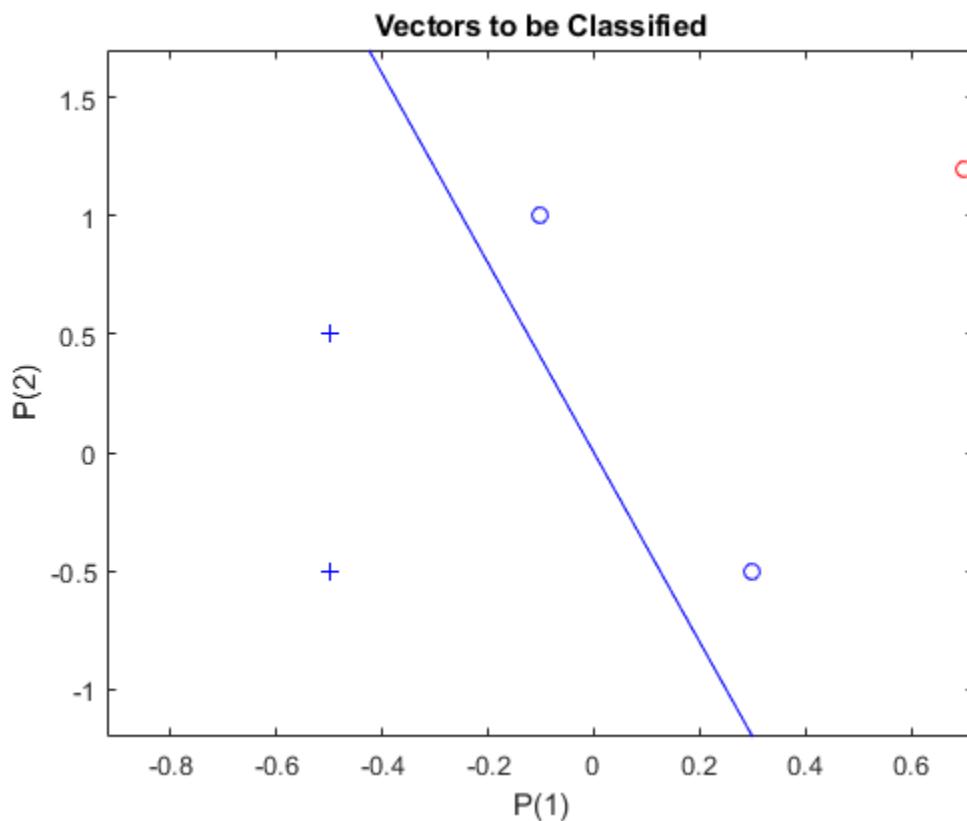
```
x = [0.7; 1.2];
y = net(x);
plotpv(x,y);
point = findobj(gca,'type','line');
point.Color = 'red';
```



开启 "hold"，以便先前的绘图不会被删除，并绘制训练集和分类线。

感知器正确地将我们的新点（红色）分类为类别“零”（用圆圈表示）而不是“一”（用加号表示）。

```
hold on;
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
hold off;
```

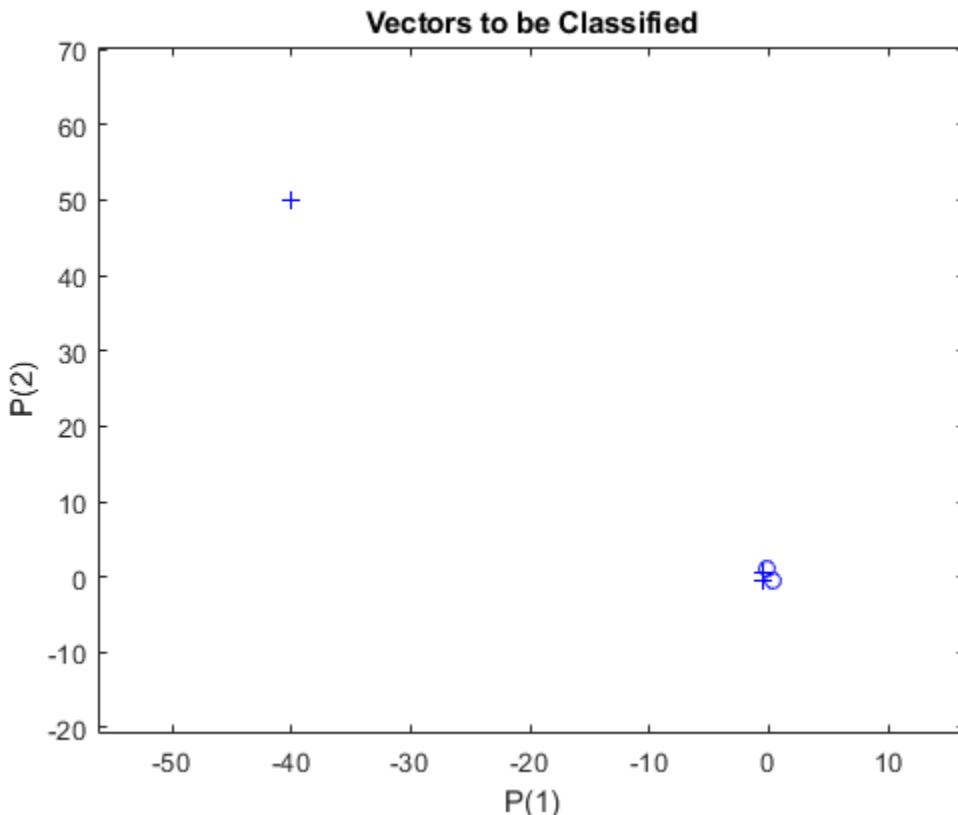


## 离群值输入向量

2 输入硬限制神经元被训练为将 5 个输入向量分类为两个类别。然而，由于 1 个输入向量比所有其他向量大得多，因此训练需要很长时间。

X 中的五个列向量中的每一个都定义了一个 2 元素输入向量，行向量 T 定义了向量的目标类别。使用 PLOTPV 绘制这些向量。

```
X = [-0.5 -0.5 +0.3 -0.1 -40; -0.5 +0.5 -0.5 +1.0 50];
T = [1 1 0 0 1];
plotpv(X,T);
```



请注意，4 个输入向量的幅值远远小于绘图左上角的第五个向量。感知器必须将 X 中的 5 个输入向量正确分类为由 T 定义的两个类别。

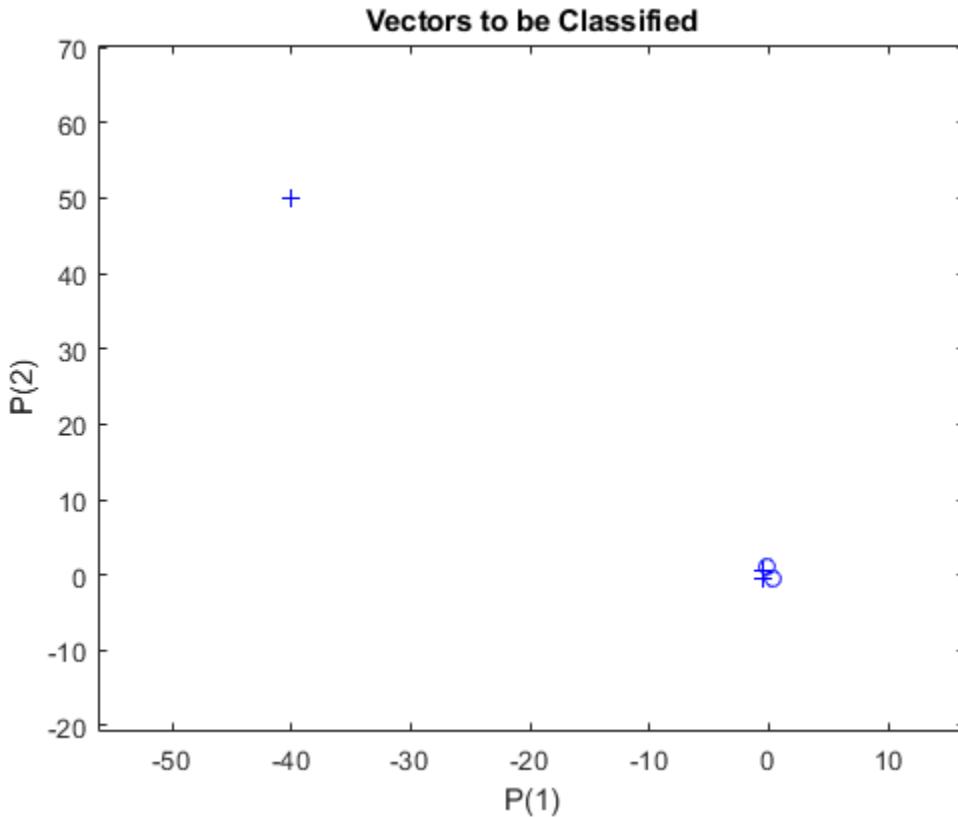
PERCEPTRON 创建一个新网络，然后用输入和目标数据对该网络进行配置，产生其权重和偏置的初始值。  
(配置通常不是必需的，因为它由 ADAPT 和 TRAIN 自动完成。)

```
net = perceptron;
net = configure(net,X,T);
```

将神经元的最初分类尝试添加到绘图中。

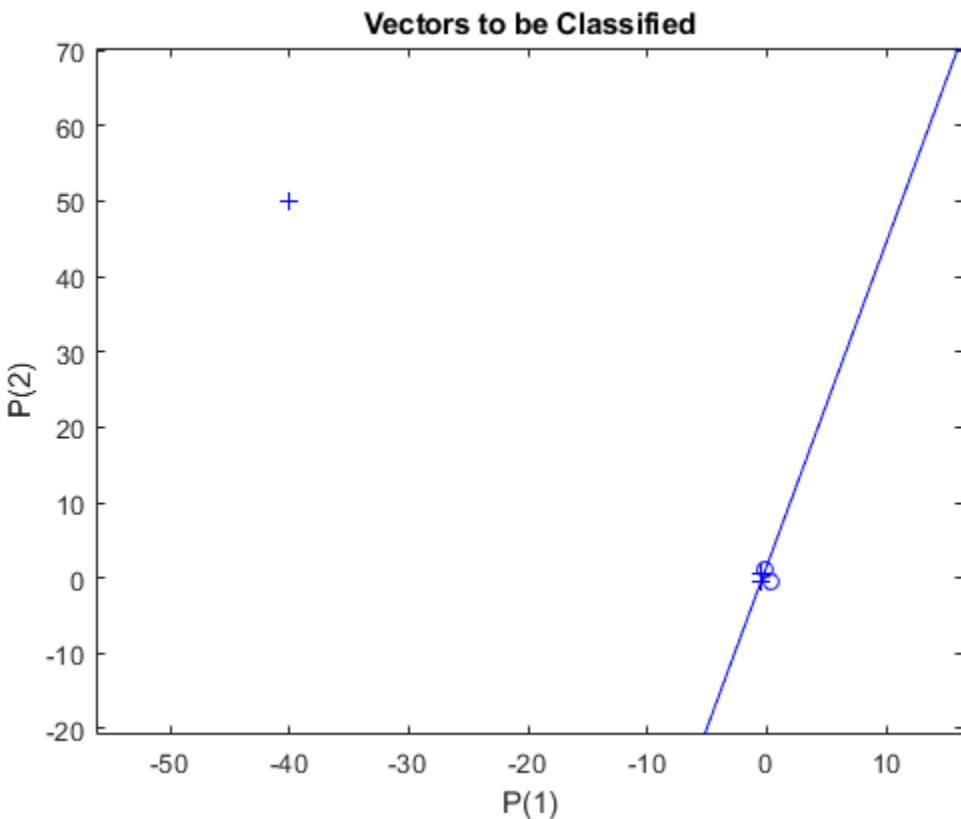
初始权重设置为零，因此任何输入都会生成相同的输出，而且分类线甚至不会出现在图上。别担心...我们将对它进行训练！

```
hold on
linehandle = plotpc(net.IW{1},net.b{1});
```



ADAPT 返回一个新网络对象（它作为更好的分类器执行）、网络输出和误差。此循环会自适应网络并绘制分类线，直到误差为零。

```
E = 1;
while (sse(E))
    [net,Y,E] = adapt(net,X,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
end
```

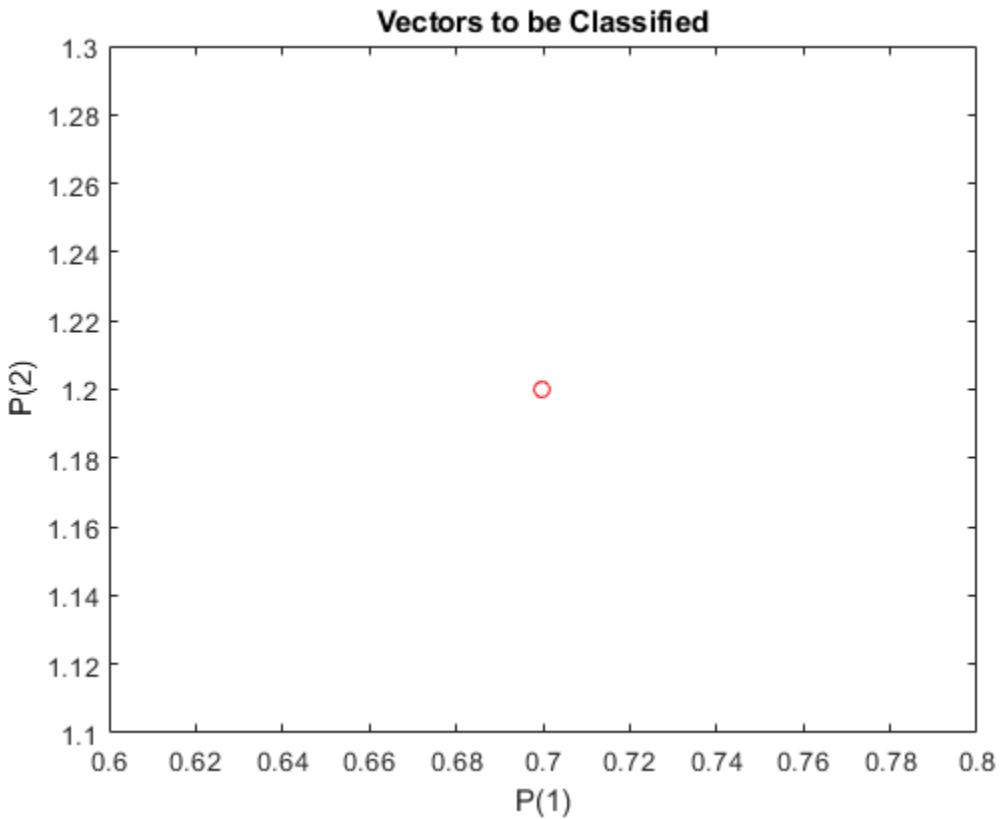


注意感知器需要三遍才能得到正确结果。对于这样一个简单问题来说，这是一段很长的时间。训练时间长的原因是存在离群值向量。尽管训练时间很长，感知器仍能正确学习，并可用于对其他输入进行分类。

现在，SIM 可用于对任何其他输入向量进行分类。例如，对输入向量  $[0.7; 1.2]$  进行分类。

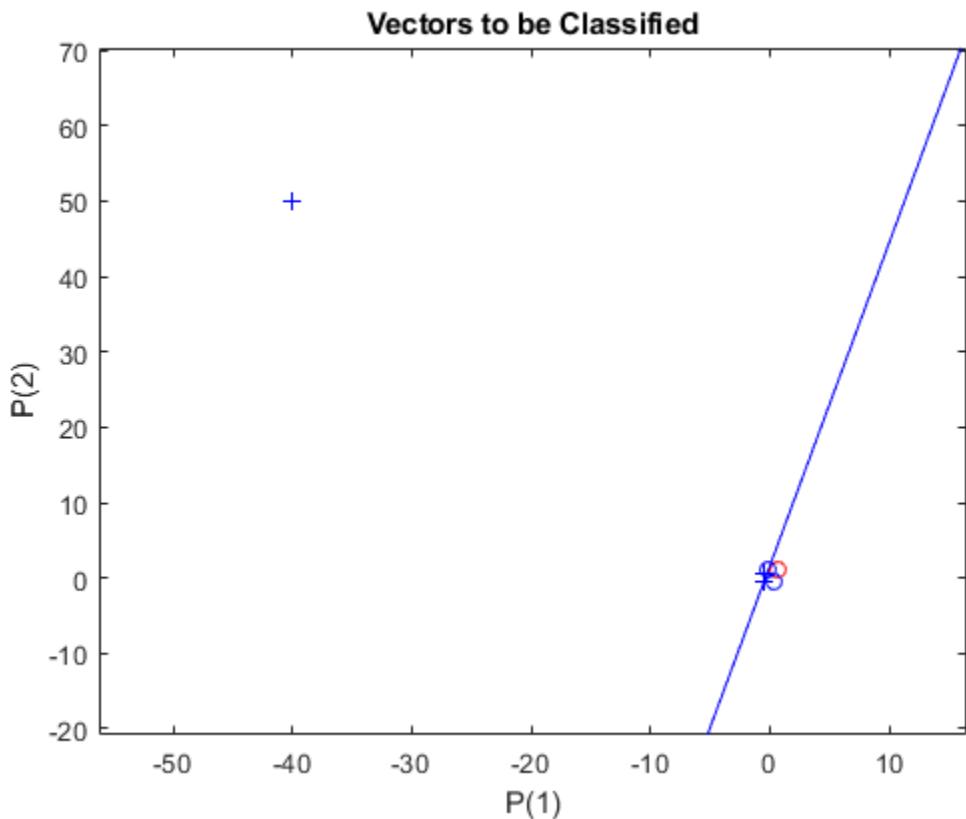
此新点及原始训练集的绘图显示了网络的性能。为了将其与训练集区分开来，将其显示为红色。

```
x = [0.7; 1.2];
y = net(x);
plotpv(x,y);
circle = findobj(gca,'type','line');
circle.Color = 'red';
```



打开 "hold" , 这样之前的绘图不会被擦除。将训练集和分类线添加到绘图中。

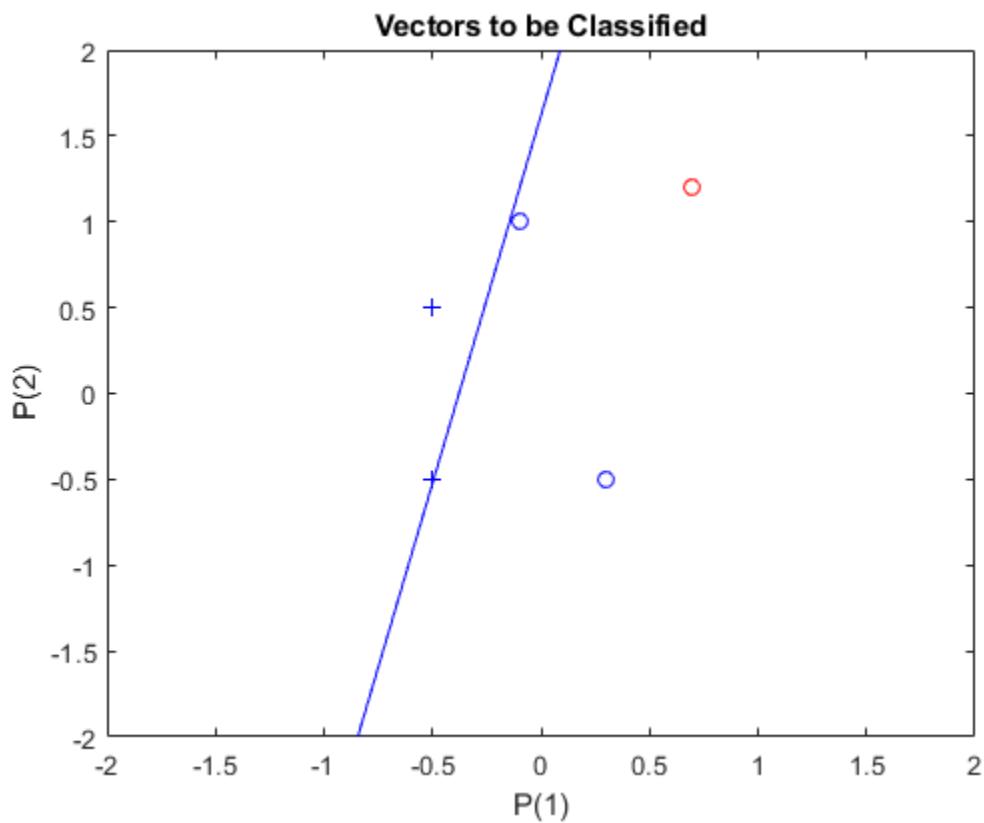
```
hold on;
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
hold off;
```



最后，放大关注的区域。

感知器正确地将我们的新点（红色）分类为类别“零”（用圆圈表示）而不是“一”（用加号表示）。尽管训练时间很长，感知器仍能正确学习。要了解如何减少与离群值向量相关联的训练时间，请参阅“归一化感知器规则”示例。

```
axis([-2 2 -2 2]);
```

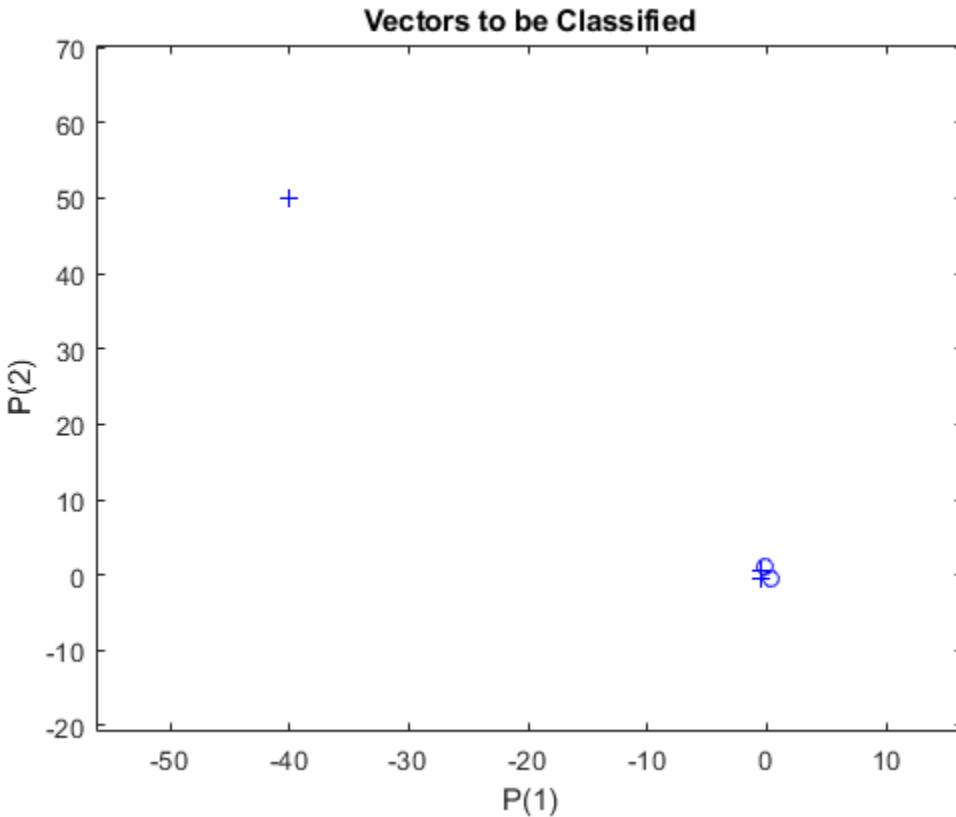


## 归一化感知器规则

2 输入硬限制神经元被训练为将 5 个输入向量分类为两个类别。尽管一个输入向量比其他输入向量大得多，使用 LEARNPN 进行训练还是很快的。

X 中的五个列向量中的每一个都定义了一个 2 元素输入向量，行向量 T 定义了向量的目标类别。使用 PLOTPV 绘制这些向量。

```
X = [-0.5 -0.5 +0.3 -0.1 -40; ...
      -0.5 +0.5 -0.5 +1.0 50];
T = [1 1 0 0 1];
plotpv(X,T);
```



请注意，4 个输入向量的幅值远远小于绘图左上角的第五个向量。感知器必须将 X 中的 5 个输入向量正确分类为由 T 定义的两个类别。

PERCEPTRON 用 LEARPN 规则创建一个新网络，相对于 LEARNP（默认值），该网络对输入向量大小的巨大变化不太敏感。

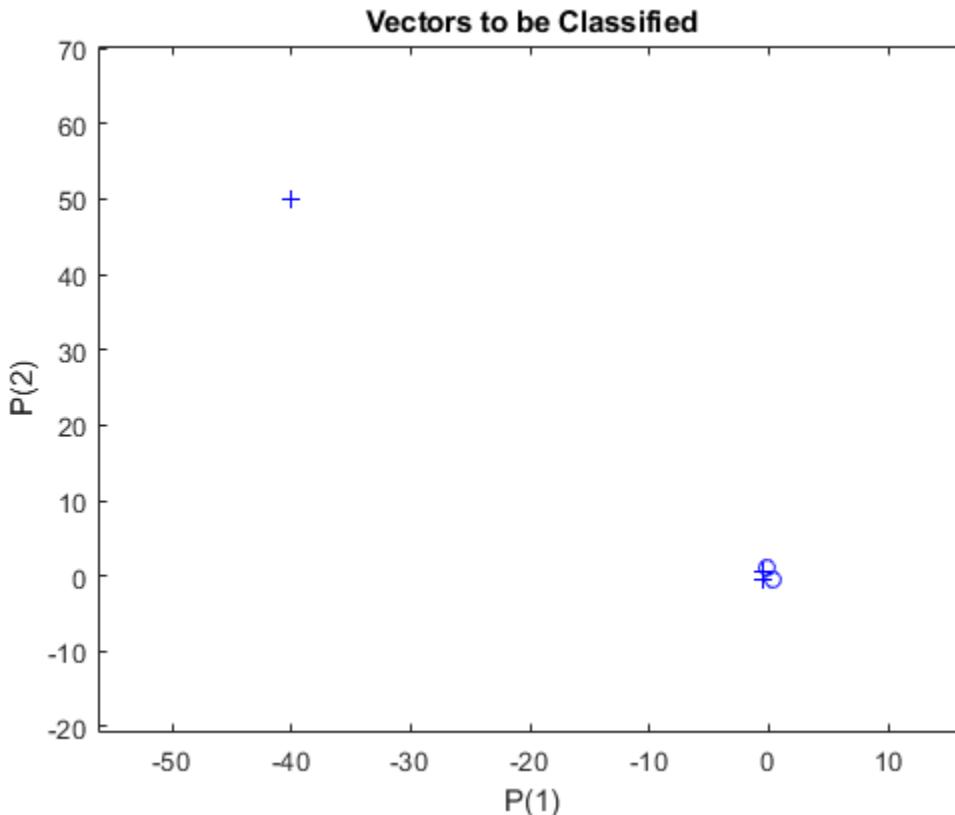
然后用输入数据和目标数据对该网络进行配置，得到其权重和偏置的初始值。（配置通常不是必需的，因为它由 ADAPT 和 TRAIN 自动完成。）

```
net = perceptron('hardlim','learnpn');
net = configure(net,X,T);
```

将神经元的最初分类尝试添加到绘图中。

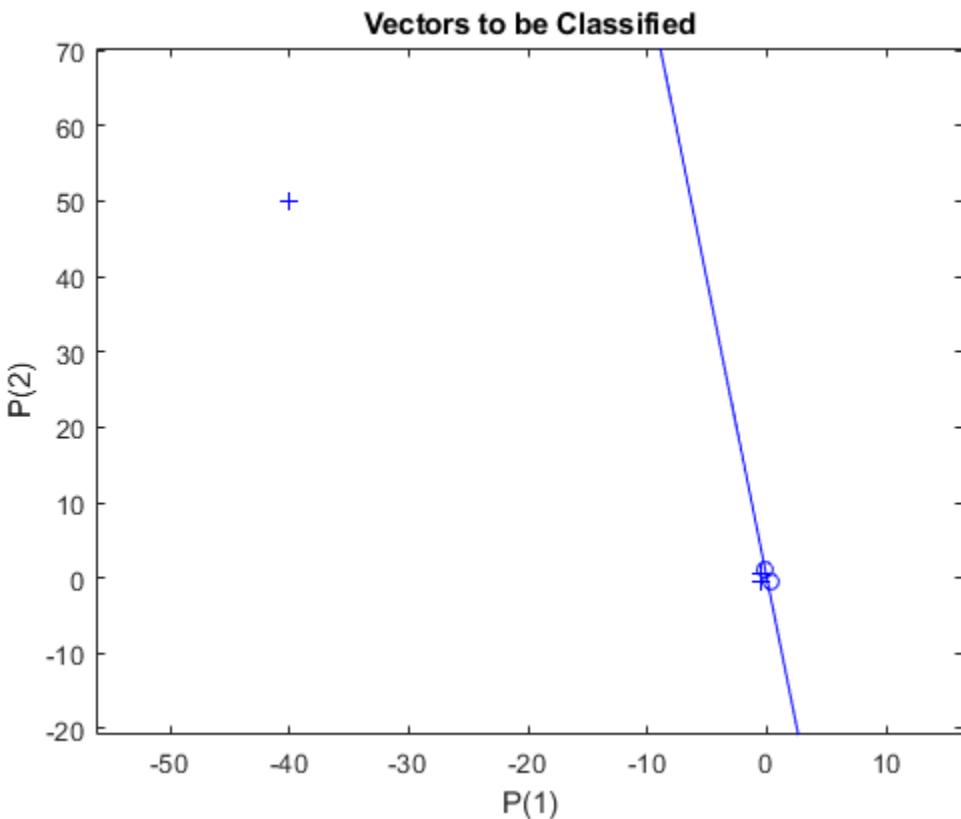
初始权重设置为零，因此任何输入都会生成相同的输出，而且分类线甚至不会出现在图上。别担心...我们将对它进行训练！

```
hold on
linehandle = plotpc(net.IW{1},net.b{1});
```



ADAPT 返回一个新网络对象（它作为更好的分类器执行）、网络输出和误差。此循环允许网络自适应，绘制分类线，并继续进行直到误差为零。

```
E = 1;
while (sse(E))
    [net,Y,E] = adapt(net,X,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle);
    drawnow;
end
```

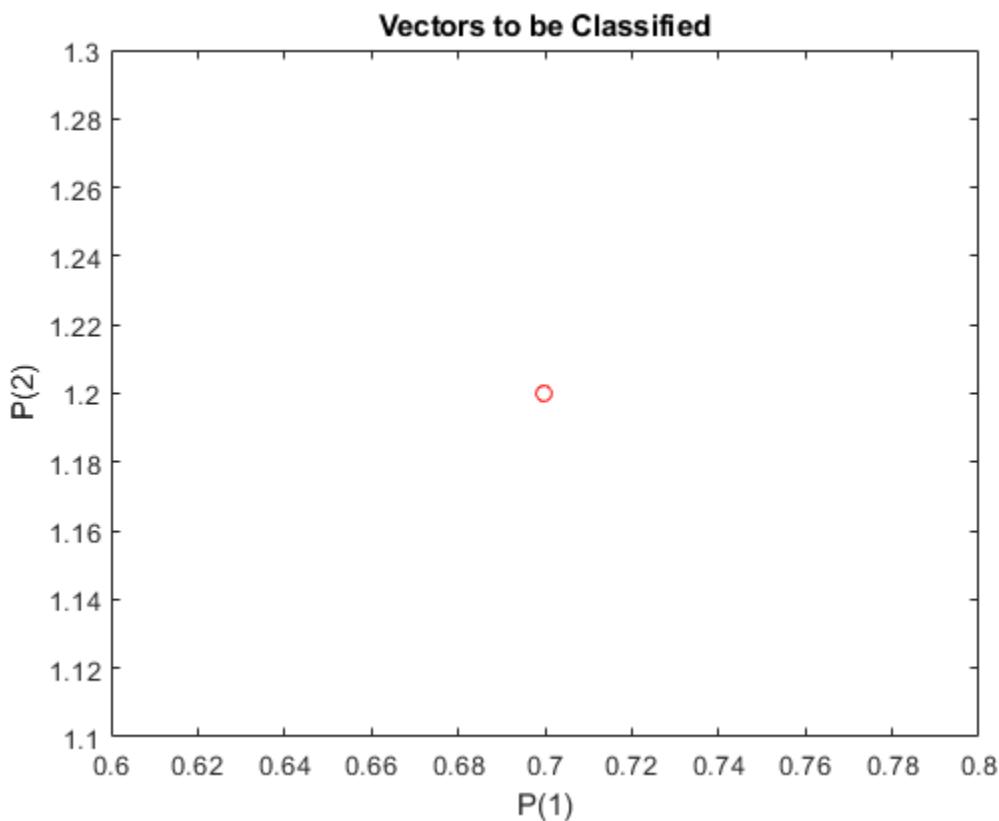


请注意，用 LEARNP 进行训练只需要 3 轮，而用 LEARNPN 求解同样的问题需要 32 轮。因此，当输入向量大小有巨大变化时，LEARNPN 的表现优于 LEARNP。

现在，SIM 可用于对任何其他输入向量进行分类。例如，对输入向量 [0.7; 1.2] 进行分类。

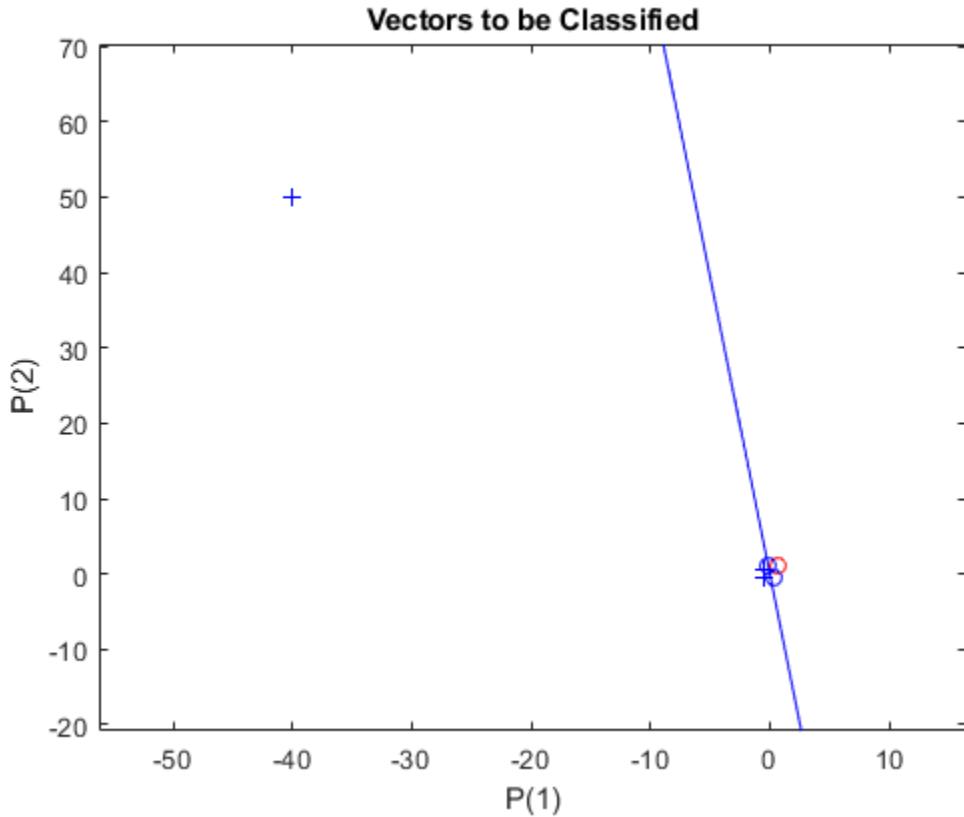
此新点及原始训练集的绘图显示了网络的性能。为了将其与训练集区分开来，将其显示为红色。

```
x = [0.7; 1.2];
y = net(x);
plotpv(x,y);
circle = findobj(gca,'type','line');
circle.Color = 'red';
```



打开 "hold" , 这样之前的绘图不会被擦除。将训练集和分类线添加到绘图中。

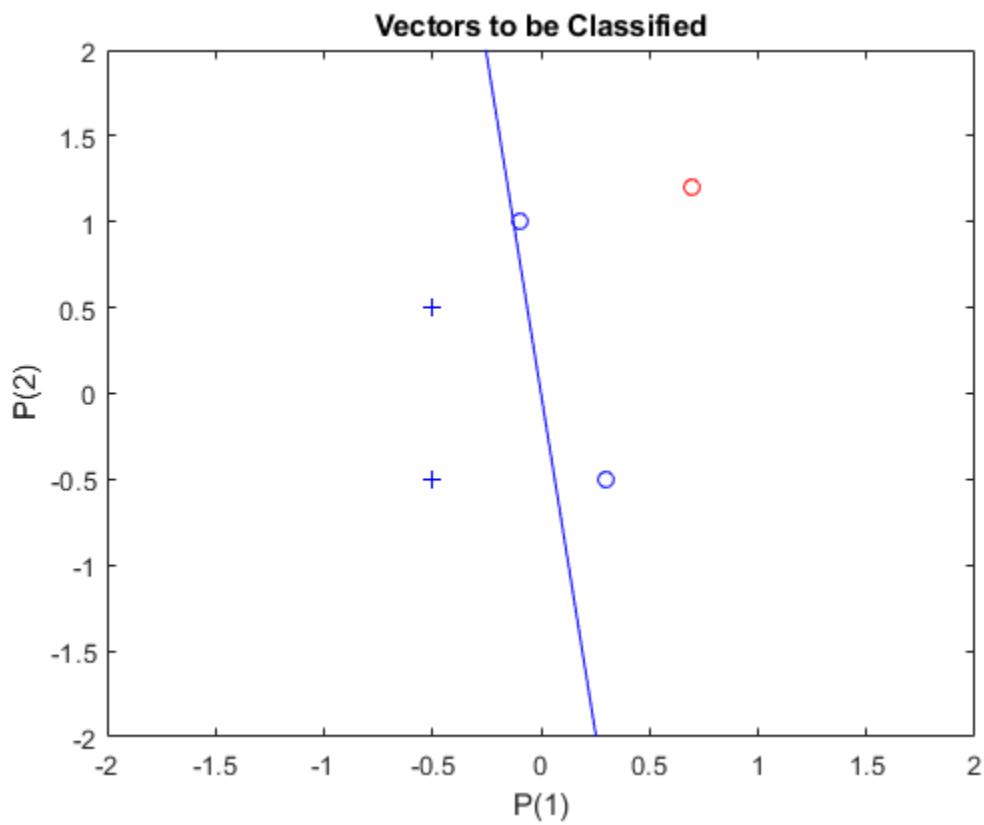
```
hold on;
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
hold off;
```



最后，放大关注的区域。

感知器正确地将我们的新点（红色）分类为类别“零”（用圆圈表示）而不是“一”（用加号表示）。尽管存在离群值，感知器仍能在短得多的时间内正确学习（与“离群值输入向量”示例相比）。

```
axis([-2 2 -2 2]);
```

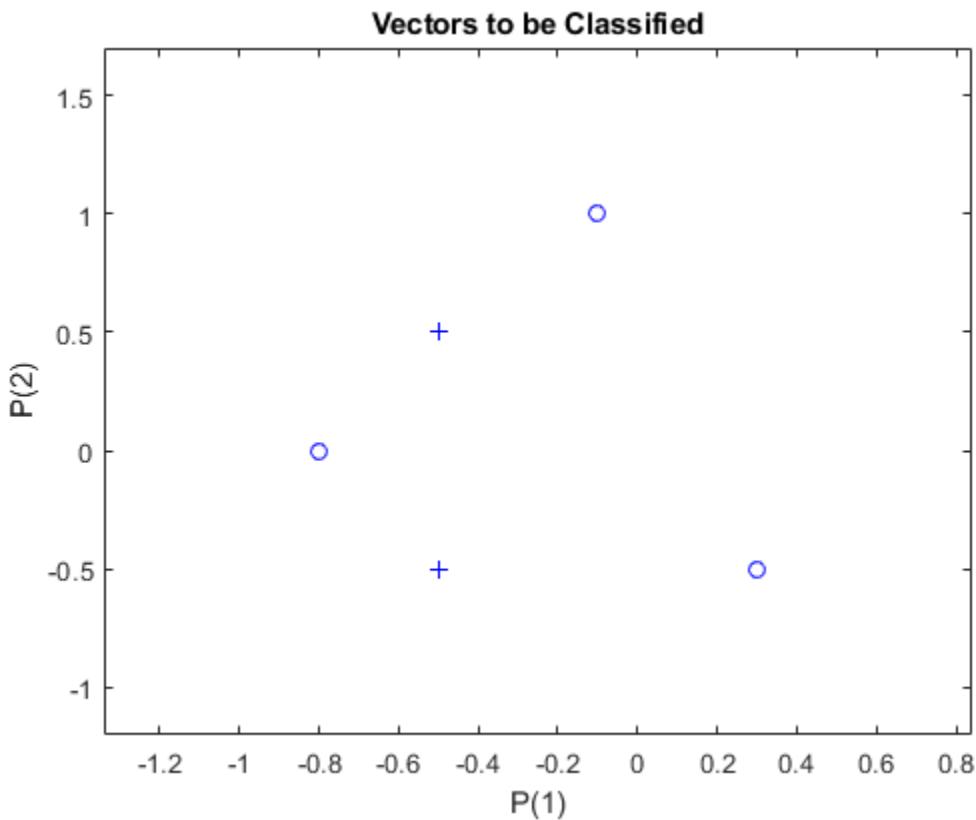


## 线性不可分向量

2 输入硬限制神经元无法对 5 个输入向量正确分类，因为它们是线性不可分的。

$X$  中的五个列向量中的每一个都定义了一个 2 元素输入向量，行向量  $T$  定义了向量的目标类别。使用 PLOTPV 绘制这些向量。

```
X = [ -0.5 -0.5 +0.3 -0.1 -0.8; ...
       -0.5 +0.5 -0.5 +1.0 +0.0 ];
T = [1 1 0 0 0];
plotpv(X,T);
```



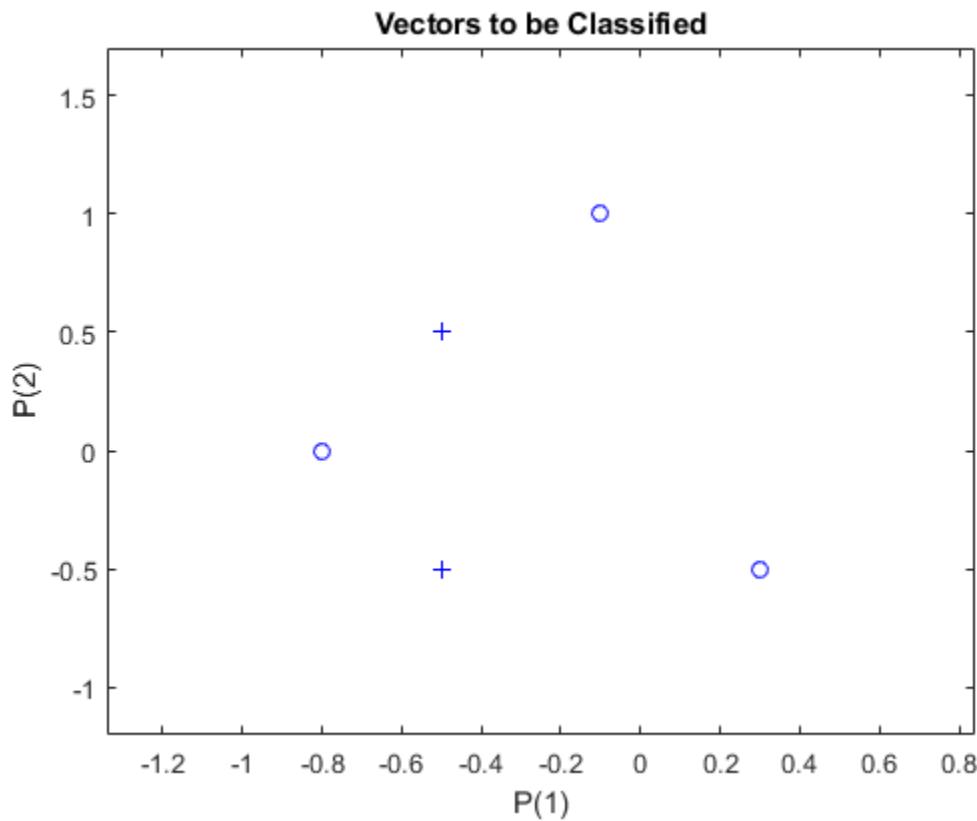
感知器必须将  $X$  中的 5 个输入向量正确分类为由  $T$  定义的两个类别。因为这两种输入向量无法由一条直线分隔，感知器将无法进行上述分类。

在此处创建和配置初始感知器。（配置步骤通常是可选的，因为它由 ADAPT 和 TRAIN 自动执行。）

```
net = perceptron;
net = configure(net,X,T);
```

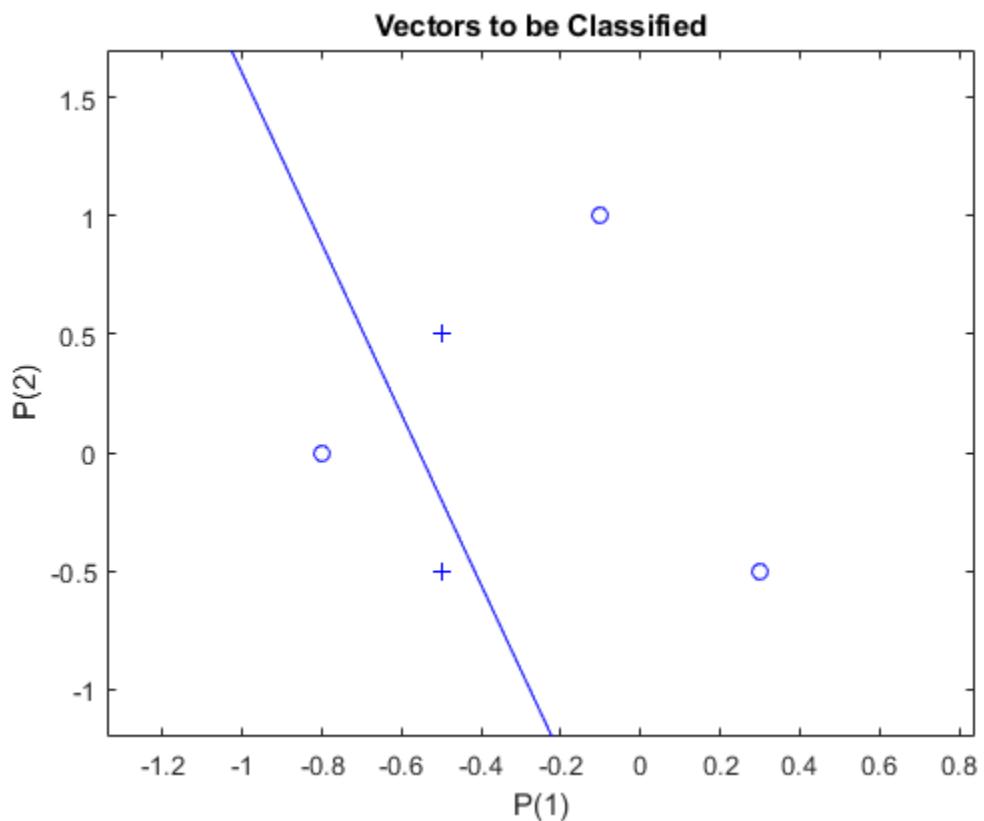
将神经元的最初分类尝试添加到绘图中。初始权重设置为零，因此任何输入都会生成相同的输出，而且分类线甚至不会出现在图上。

```
hold on
plotpv(X,T);
linehandle = plotpc(net.IW{1},net.b{1});
```



ADAPT 在基于输入和目标数据进行学习后，返回一个新网络、输出和误差。此循环允许网络重复自适应、绘制分类线，并在 25 次迭代后停止。

```
for a = 1:25
    [net,Y,E] = adapt(net,X,T);
    linehandle = plotpc(net.IW{1},net.b{1},linehandle); drawnow;
end;
```



请注意，永远不会得到零误差。尽管经过训练，感知器尚未成为可接受的分类器。感知器的根本限制是只能对线性可分的数据进行分类。

# 浅层神经网络参考书目

---



# 数学表示法



# 用于 Simulink 环境的神经网络模块

## 神经网络 Simulink 模块库

### 本节内容

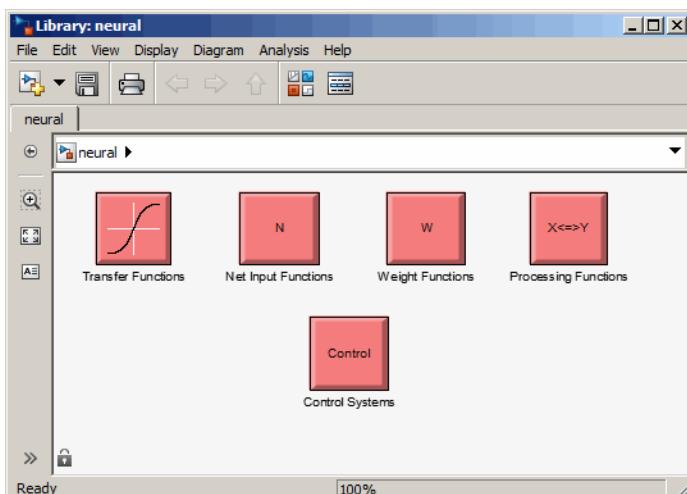
- “传递函数模块” (第 B-2 页)
- “净输入模块” (第 B-3 页)
- “权重模块” (第 B-3 页)
- “处理模块” (第 B-3 页)

Deep Learning Toolbox 产品提供一组模块，您可以通过 Simulink 软件使用这些模块构建神经网络，或使用函数 `gensim` 生成您使用 MATLAB 软件创建的任何网络的 Simulink 版本。

使用以下命令打开 Deep Learning Toolbox 模块库：

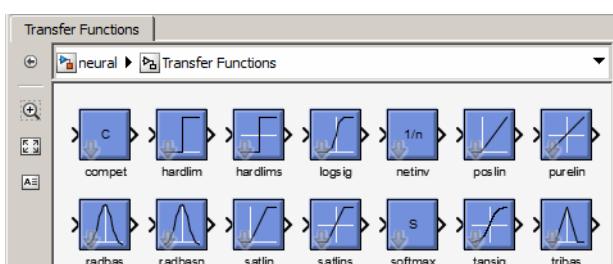
```
neural
```

这将打开一个包含五个模块的库窗口。其中每个模块都包含附加模块。



### 传递函数模块

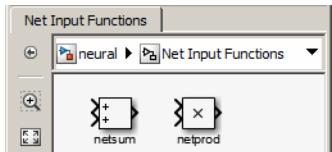
双击 Neural library 窗口中的 Transfer Functions 模块，以打开一个包含多个传递函数模块的窗口。



其中每个模块都接受一个净输入向量，并生成一个对应的输出向量，其维度与输入向量相同。

## 净输入模块

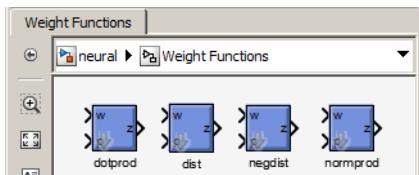
双击 Neural library 窗口中的 Net Input Functions 模块，以打开一个包含两个净输入函数模块的窗口。



其中每个模块都接受任意数量的加权输入向量、加权层输出向量和偏置向量，并返回一个净输入向量。

## 权重模块

双击 Neural library 窗口中的 Weight Functions 模块，打开一个包含三个权重函数模块的窗口。



其中每个模块都接受一个神经元的权重向量，并将其应用于一个输入向量（或层输出向量），以获得神经元的加权输入值。

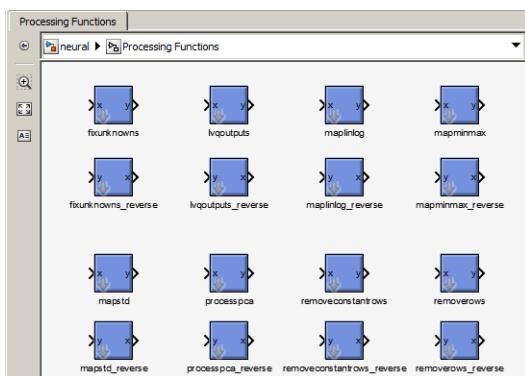
特别要注意，这些模块要求神经元的权重向量定义为列向量。这是因为 Simulink 信号可以为列向量，但不能为矩阵或行向量。

还需要注意，由于存在此限制，您必须创建 S 权重函数模块（每行一个）来实现要输入到包含 S 神经元的层的权重矩阵。

这与另外两种模块形成对比。每层只需要一个净输入函数模块和一个传递函数模块。

## 处理模块

双击 Neural library 窗口中的 Processing Functions 模块，以打开一个包含处理模块及其对应的反向处理模块的窗口。



其中每个模块都可用于预处理输入和后处理输出。



# 代码说明

