

江西理工大学

## 本科毕业设计（论文）

题    目：基于全卷积神经网络的图像分类

专题题目：

学    院：信息工程学院

专    业：通信工程

班    级：通信 142 班

学    号：20143519

学    生：姚美君

指导教师：喻玲娟    职称：副教授

指导教师：            职称：

时间：        年    月    日

## 摘 要

图像分类是计算机视觉中的子领域，是图像检测、图像分割、物体跟踪、行为分析等其他高层视觉任务的基础。在过去，图像分类主要使用统计语言模型和传统的机器学习算法如支持向量机 SVM。随着硬件的发展、数据积累、算法的优化，深度学习在大规模数据上的优势便体现了出来。现代的大多数情景中，卷积神经网络 (CNN) 广泛应用于图像分类，CNN 由交替的卷积层和池化层，以及全连接层构成。本文研究仅有卷积层组成的新架构，即全卷积神经网络，它是对 CNN 结构的一种改进，将其中的池化层和全连接层全部用卷积层替代。进一步地，本文针对公开的数据集 mnist 和 cifar-10，采用全卷积神经网络对数据集进行分类，在时间效率和精确度两个方面，与 CNN 的实验结果进行对比。

**关键词：**计算机视觉；支持向量机；CNN 卷积神经网络；FCN 全卷积神经网络

# ABSTRACT

Image classification is a subfield in computer vision and is the basis for other high-level visual tasks such as image detection, image segmentation, object tracking, and behavior analysis. In the past, image classification mainly used statistical language models and traditional machine learning algorithms such as support vector machine SVM. With the development of hardware, data accumulation, and optimization of algorithms, the advantages of deep learning on large-scale data are reflected. In most modern scenarios, convolutional neural networks (CNNs) are widely used for image classification. CNN consists of alternating convolutional layers and pooled layers, and fully connected layers. In this paper, we study the new architecture consisting of only convolutional layers, that is, full convolutional neural networks. It is an improvement to the CNN structure, in which the pooling layer and the fully connected layer are all replaced by convolutional layers. Further, for the disclosed datasets mnist and cifar-10, the data set is classified using a full convolutional neural network, and compared with the experimental results of CNN in terms of time efficiency and accuracy.

**Keywords:** Computer vision; support vector machine; CNN convolutional neural network; FCN full convolutional neural network

# 目 录

第一章 绪论.....	1
1.1 研究背景与意义.....	1
1.2 国内外研究现状.....	1
1.2.1 计算机视觉发展背景和趋势.....	1
1.2.2 国内研究现状.....	2
1.2.3 国外研究现状.....	3
1.3 图像预处理.....	4
1.4 本文的主要工作.....	5
第二章 卷积神经网络基本原理.....	7
2.1 卷积层.....	7
2.2 池化层.....	13
2.3 全连接层.....	14
2.4 激活函数.....	15
2.5 参数选择.....	18
2.6 本章小结.....	20
第三章 全卷积神经网络的基本原理.....	21
3.1 全卷积神经网络架构.....	21
3.2 卷积层替换池化层.....	23
3.3 卷积层替代全连接层.....	23
3.4 本章小结.....	24
第四章 基于全卷积神经网络的分类.....	25

4.1	mnist 和 cifar-10 数据集介绍.....	25
4.1.1	minst 图像集.....	25
4.1.2	cifar-10 图像集.....	26
4.2	minst 数据集处理结果.....	26
4.3	cifar-10 数据集处理结果.....	29
4.4	本章小结.....	35
第五章	总结与展望.....	36
5.1	论文总结.....	36
5.2	展望.....	36
附 录	.....	38
参考文献	.....	51
外文文献	.....	53
致 谢	.....	63

## 第一章 绪论

### 1.1 研究背景与意义

在研究不同算法在图像分类上的表现前，首先要了解它的意义。为什么需要图像分类，扩大一步讲，为什么需要计算机视觉，计算机视觉到底是什么，可以用来做什么？计算机视觉是包含人工智能和计算机科学两个领域的学科<sup>[1]</sup>。目的是为了电脑可以在视觉上理解世界。是机器智能中重要的组成部分。

计算机视觉的目标是模仿人类的视觉，通过以下三个主要的过程处理数字图像，依次执行来达到目标。

1. 图像采集
2. 图像处理
3. 图像分析和理解

第一步，图像采集是将周围的模拟世界，转为为由 0 和 1 组成的二进制矩阵，也就是数字图像。图像处理是第二步，是从第一步的结果提取出低阶特征<sup>[2]</sup>。比如，边缘检测、语义分割、分类、特征检测和匹配。然后最后才是，图像分析和理解，使用高级算法和第二步产生的图像低级特征，做出决策。例如，图像分类、目标检测、语义分割。见下图 1-1，可知计算机视觉、图像处理、机器视觉这些学科相互交叉，涉及到了机器学习、人工智能等各个领域。

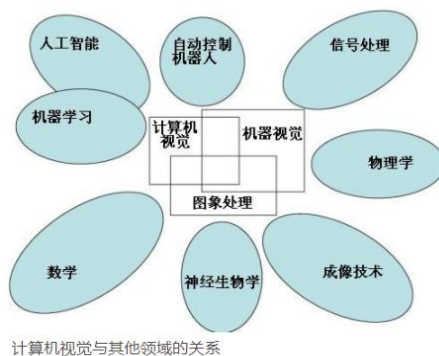


图 1-1 计算机与其他领域关系

### 1.2 国内外研究现状

#### 1.2.1 计算机视觉发展背景和趋势

目前图像处理有几大研究挑战：

1. 同类物体具有不同姿态、光照条件、遮掩、子类别等。如图 1-2、1-3 所示，外形不同，光照颜色和情况也不同。



图 1-2 不同情况下的飞机，外形不同。光照颜色不同



图 1-3 汽车

2. 在特征维度和数据量两个方面，图像数据集的规模不断加大。需要更好的算法来加速模型的训练和精确性。目前，计算机视觉主要应用于以下几类<sup>[3]</sup>：动作识别、增强现实（AR）、自动驾驶、机器人、图像恢复如去噪等等。

### 1.2.2 国内研究现状

ImageNet 数据集中包含有 1500 万由人手工标记的图像，每年都会举办一次 ImageNet 大规模视觉识别挑战赛<sup>[4]</sup>。参赛的代码根据正确率，以分高低。图 1-4 展示的是 ImageNet 官网的图像信息展示。在大赛上的比赛成绩反应了目前的图像处理应用在全世界的最高水平。



图 1-4 Image 图像数据集

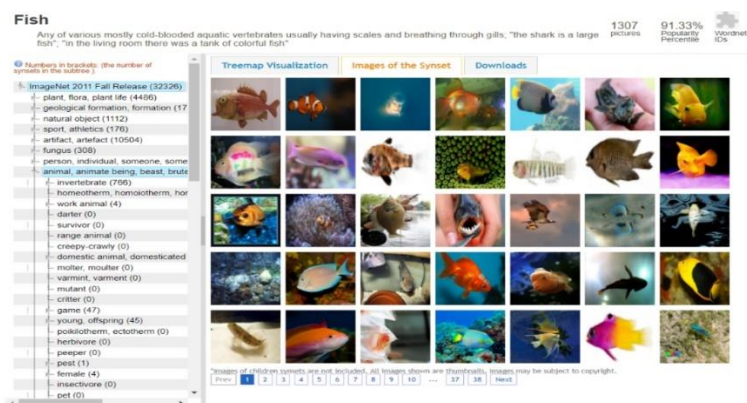


图 1-5 Image 鱼相关图像

2016 年的 ILSVRC，来自中国的团队：Trimps-Soushen（公安部三所），CUImage（商汤和港中文），NUIST（南京信息工程大学，HikVision（海康威视）等团队包揽了各个项目的冠军。国内关于将深度学习应用于图像分类研究成果，在全世界都处于领先的状态。根据下图 1-6 可以发现，无论的图像分类、物体识别、物体检测，计算机的正确率都已经远远超越人类。可以说，计算机视觉在感知方面的问题已经得到了很好的解决。

#### ImageNet Challenge



图 1-6 Image 错误率下降的趋势

### 1.2.3 国外研究现状

1998 年，Yann Lecun 发表了一篇关于卷积神经网络模型（CNN）<sup>[5]</sup> 的论文，确定了 CNN 基础架构的各个组件，之后的算法优化大多在此基础上应用了更加复杂的激活函数和正则化方法<sup>[6]</sup>。2012 年，AlexNet<sup>[7]</sup> 赢得 ImageNet 大赛冠军，图 1-6 展示了 Alex 的经典架构。2013 年，ZFNet 出现。2014 年，CNN 获得 imagenet 冠军。2015 年，ResNet<sup>[8]</sup> 和 FCN<sup>[9]</sup> 提出。2016 年，ResNet，在此基础上他们提出了一种新架构，可以减少整个网络的深度并增加残余网络的宽度。2017 年，SENet 提出。



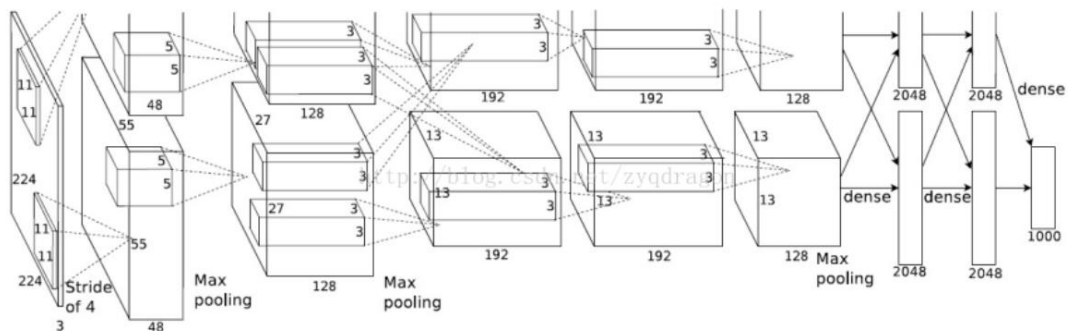


图 1-7 Alex 在 2012 年论文中的卷积核的尺寸

### 1.3 图像预处理

在进行图像分类之前，了解一些关于数字图像的信息。以及，进行模型之前的图像的预处理过程。

## 1. 图像的数学原理

图像的科学定义是：可以用一个定义在平面坐标上的二维函数  $I=f(x, y)$  来表示。其中，定义域  $(x, y)$  是平面坐标，值域  $I=f(x, y)$  是点  $(x, y)$  的亮度或彩色值。一般来说，较高的亮度用较大的函数值表示。灰度图像的值域  $I$  是一个一元标量： $I=\text{greylevel}$ 。彩色图像的值域  $I$  是一个多元向量：如  $I=(r, g, b)$ 。其实，图像本身就是连续的。①对于空间上的连续性，例如是二维成像平面上的点，可以用平面坐标  $(x, y)$  表示<sup>[10]</sup>。②对于时间上的连续性，可以用时刻值  $t$  表示。③对于颜色上的连续性，可以用光的波长  $\lambda$  表示。

因此图像可以表述为：在  $t$  时刻，或成像平面上的点  $(x, y)$  所接收到的波长  $\lambda$  的光能量强度。图像数字化就是把理想图像的各连续量按照某种方式转换为离散数字量的表示。图像数字化的过程直播那个往往会造成少量的信息丢失，并且是无法避免的。图像数字化步骤：①空间坐标离散化（等间隔采样）。②时间上只取离散值（等间隔采样）。③颜色值上取有限个值（动态范围采样）。

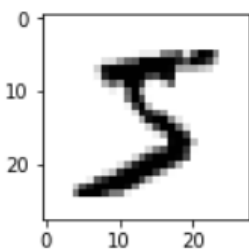


图 1-8 Mnist 数据集数字 5

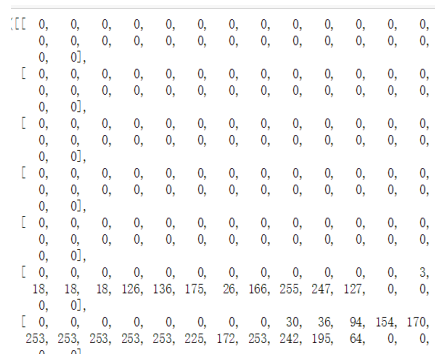


图 1-9 数字 5 像素矩阵部分截图

人看到的图像和计算机看到的图像，有什么区别呢？人眼中的数字 5，如下图 1-8：而在机器看来，图像不过是一堆矩阵。以上图的 mnist 数据集为例，人眼观察到的是 5。通过 python 的 numpy 打印出图像的像素矩阵，见图 1-9。

## 2. 图像表示法：

空间上：静态图像可分为矢量(Vector)图和位图(Bitmap)<sup>[11]</sup>。

颜色：二值图像（只有 0 和 1）、灰度图像（0~255）、彩色图像（rgb 三个通道叠加）<sup>[12]</sup>。

格式：TIFF、JPEG、PGM、PPM、GIF、BMP、PNG、Video formats 等等。

质量：层次、对比度、清晰度（亮度、对比度、尺度大小、细微层次、颜色饱和度）。

## 3. 图像预处理过程：

（1）调整图片尺寸：对于不同尺寸的尺寸，分辨率和距离，可以简单的将每个图片的最大边缩放到一个固定的长度。

①扩充数据（提升性能）。

②旋转：随机旋转 0-360 度（统一）。

③平移：随机平移-10 到 10 个像素点（统一）。

④重新缩放：随机比例因子在 1/1.6 - 1.6 之间（对数均匀）。

⑤镜面翻转：是或否（bernoulli）。

⑥剪切：随机，角度-20° 到 20° 之间（统一）。

⑦拉伸：随机，拉伸系数在 1 / 1.3 和 1.3 之间（对数均匀）。

（2）标准化是改变像素强度值范围的图像处理过程<sup>[13]</sup>，它可以提升模型准确度，并且更快收敛。例如：灰度图像中的所有像素点除以 255，将像素强度的范围标准化到[0, 1]之间。

## （3）Label Onehot encoding 转换（分类和回归算法）

大部分的机器学习算法不能直接处理标签数据。它们要求输入和输出都是数字。但是，如果是整数编码，例如：red: 1、blue: 2、yellow: 3。容易造成混淆，并且在多分类情况下累加时容易出错。而 Onehot 是讲 label 编码为二进制数，在分类不是太多的情况下，表现很好。

## 1.4 本文的主要工作

本文目的在于将 CNN 和 FCN 算法应用于图像分类。比较两者在不同大小的数据集上，时间效率和精确度的效能表现。

本文的各章节的具体安排如下：

第一章：简明地描述本课题的研究背景、意义和现状，以及图像表示法和图像预处理方法。

第二章：卷积神经网络基本原理。

第三章：全卷积神经网络基本原理。

第四章：基于全卷积神经网络的图像分类，并与 CNN 的分类结果作比较。

第五章 总结和展望。

## 第二章 卷积神经网络基本原理

本章介绍了 CNN 的组成原理及其变形算法，讲述 CNN 应用于图像分类的原因和效能。在 CNN 出现之前，图像分类多采用 K-nearest neighbor(k-means)和支持向量机算法。

通常将模式识别分为两个步骤：①特征工程；②根据特征训练图像分类模型。如图 2-1。

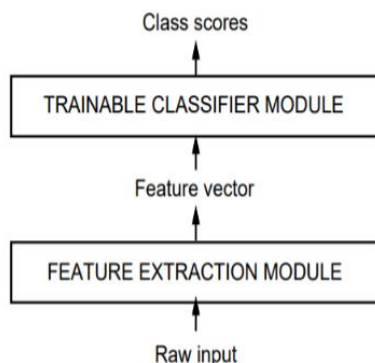


图 2-1 传统的目标检测形式和结构

特征的提取是人工的方式，这个过程取决于个人对于知识的理解。由于特征提取这个过程是人工的，所以识别的精确率很大程度上取决于人工选取的特征集的优劣。不仅如此，对于每个特定的任务，它们的特征集也不一样。并且特征的筛选是一门费时费力的工程。但是，对于神经网络来说，是将整个图像作为输入，由算法本身自动提取特征。

这十年来从机器学习过渡到深度学习，主要是以下三个因素：

（1）以前的计算机的计算速度很慢，这些年硬件的发展符合摩尔定律，计算速度大幅增加，机器可以处理高维度数据并且生成复杂的决策函数。

（2）大数据时代，数据在不管是在维度还是在数据量上都有大量的增加和提升，而深度学习的准确率目前更多的依赖于数据量的增加还不是特征提取。

（3）以前的深度学习依赖于一种蛮力的数值运算，而现在已经存在很多的优化算法，可以让模型更快的收敛，并且准确率也会得到提升。

目前，基本的卷积神经网络架构由一个或多个卷积层和池化层交替组成，经过 `flatten` 压平阵列为二维向量后，再在网络末端连接少量全连接层。通过卷积层和池化层提取图像特征之后，通过连接层计算各特征权重和偏差。

### 2.1 卷积层

卷积层中，会将图像像素矩阵与过滤器 `filter` 进行卷积运算。卷积层在数据

上的意义即相乘和相加<sup>[14]</sup>。卷积运算的目的是检测图像边缘，从中提取不同的特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网路能从低级特征中迭代提取更复杂的特征<sup>[15]</sup>。

(1) 图像卷积过程：

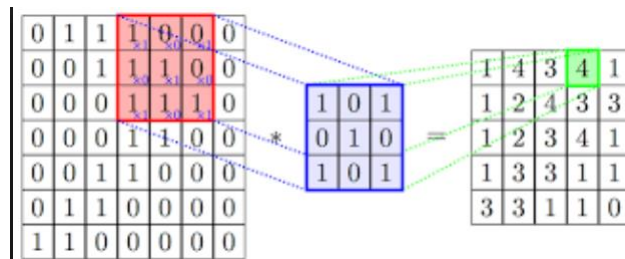


图 2-2 图像卷积过程

卷积的过程将一个自定义大小的过滤器覆盖在图像矩阵上，对应位置相乘，再将所有乘积相加。见图 2-2，蓝色矩阵为过滤器，红色矩阵为图像的某局部位。进行式 2.1 的运算，得到特征矩阵中的数字 4。

$$y(n) = \sum_{i=-\infty}^{\infty} x(i)h(n-i) = x(n) * h(n) \quad (2.1)$$

深度模型中，卷积之后，通常对卷积进行一些非线性处理。加入激活函数如 tanh、relu，使模型也能处理一些复杂的非线性问题，同时也可以使模型更快的收敛。

(2) 卷积层可训练的参数：

①卷积核尺寸 **kernel size**：过滤器 **filter** 的宽和高可以自定义。可以是 2\*2 的矩阵，也可以是 3\*3 的矩阵。如果是灰度图像，则卷积核维度为二维，如果是 RGB 三维图像，则卷积核为三维矩阵。卷积核长宽越小，则捕获的细节就更精细，从另一个角度来讲，卷积核长宽越大，则会错过图片的一些细节。总之，在决定卷积核尺寸的时候，应该先从小 size 开始尝试，慢慢的加大，观察模型精确度的变化，寻找最佳的参数。

②过滤器数量 **filters**：如果是 8 个 **filter**，就可以得到了 8 个输出特征。16 个 **filter** 特征越多，检测到的边缘角度就多，图像的特征就更多更全。训练效果就更好。但是时间成本就会增加。

③卷积的步长 **strides**：在图像像素矩阵上移动几个像素。卷积的步长越大，图像的尺寸缩减的越快，在 FCN 中，可以卷积步长为 2 的卷积层替代 2\*2 的池化层。

④补零 **padding**：补 0 策略，为“valid”，“same”。“valid”代表不补零

的卷积，即对边界数据不处理。“same”代表补零后的卷积。通常情况下，same 比 valid 更好，same 对于各个像素点更具有公平性，若不在矩阵像素周围补零，即采用 padding 为 valid 的方式卷积，则矩阵边缘或角落的像素点可能仅仅只被接触或计算一次。然而中间的像素点，却会被计算多次。进行 same 操作，可以让角落或边缘的像素点也被多次用于计算，充分发挥其作用。

⑤激活函数 Activation function：可以使卷积神经网络对于非线性的数据也有比较好的效果。通常，在卷积或者全连接层之后会，会添加一个激活函数。

⑥偏差 bias：偏置项。

(3) 卷积为什么会有效，参见以下解释：

①从数学的角度演示，垂直边缘检测。以 6\*6 的图像像素矩阵和卷积核为 3\*3 的过滤器，演示卷积的过程：

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

\*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

\*=

图 2-3 卷积过程中的边缘检测

进行式 2.1 的计算，见图 3，假设强度为正数为亮色像素，负数为暗色像素。则图像矩阵为左亮右灰的矩阵。过滤器为左亮中灰右暗的矩阵。卷积得到的特征矩阵，为中部为亮色两边为暗色的特征。此过滤器可以检测到图像的垂直边缘。由于矩阵的尺寸较小，所以检测到的边缘较粗，若在 1000\*1000 的矩阵中，边缘就会很细。运算过程中，矩阵维度和尺寸的变化见式 2.2。

$$\text{Image pixels } 6 \times 6 * \text{filter } n \times n = \text{features } (n - f + 1) * (n - f + 1) \quad (2.2)$$

不同的过滤器可以检测到不同的边缘。尝试使用更多的过滤器，可以捕获到图像中更多的特征。当使用不同的 filter，就可以检测到不同的角度边缘，25 度，45 度，73 度，水平垂直等。卷积是将特征工程从人工到自动化的过程，当图像进入卷积层，模型会使用不同的过滤器，自动计算出不同的特征。

(3) Padding，卷积是否补零

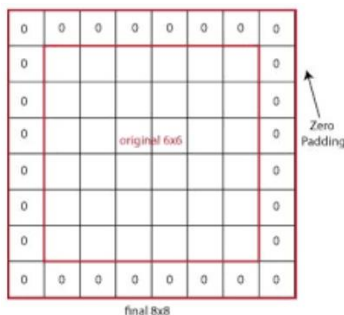


图 2-4 图像矩阵补零

6\*6 的矩阵卷积最后的结果是 4\*4，那么对于结果 4\*4 再进行卷积，只剩下 2\*2 的矩阵。最后只剩下 1\*1 的矩阵，可能会导致特征丢失。另一方面，角落的像素只被一个输出所接触或使用。中间的像素则多次重复运算，则会丢失了图像边缘位置的像素点。在图像像素矩阵的周围补零，则边缘像素点就可以被多次使用，并且多次卷积不会造成大量特征的丢失。

将之前的 6\*6 的像素矩阵，周围补一圈 0，如上图所示。对于 filter 再次进行卷积，得到以下的结果。其中，P 为 padding 的维度。若在矩阵周围补一圈零，则 p 为 1，两圈，则 p 为 2。Padding 的卷积运算满足公式 (2.4)。

$$\text{图像像素矩阵} * \text{filter} = (n + 2p - f + 1) \times (n + 2p - f + 1) \quad (2.3)$$

则  $(7 \times 7) * (3 \times 3) = (6 \times 6)$ 。若希望输出的特征与输入的图像的矩阵尺寸相同，假设输入的图像矩阵的大小为  $n \times n$ ，且卷积运算的计算过程满足公式 (2.4)，应该如何补零？首先，要满足公式 (2.4) 的输出与输入  $n \times n$  相同。见式 (2.5)，见计算过程 (2.6)，(2.7)。

$$(n + 2p - f + 1) \times (n + 2p - f + 1) = n \times n \quad (2.4)$$

$$n + 2p - f + 1 = n \quad (2.5)$$

$$p = (f - 1)/2 \quad (2.6)$$

在  $\text{padding} = (f-1)/2$  的时候，这时候的卷积的输出结果和像素矩阵维度相同。

(4) 卷积步长 (Convolution stride) :

在之前的卷积过程中，默认步长为 1 进行卷积。事实上，还可以做步长为 2 的卷积，见下图 2-9，演示步长为 2 的卷积，绿色方框是第一次卷积计算，黄色是第二次... 蓝色的第三次，由于超出矩阵边缘，所以卷积无效，跳过此次计算。

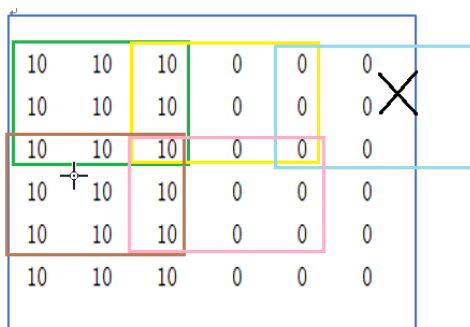


图 2-5 步长为 2 的卷积

如果考虑到卷积步长，卷积结果满足式 (2.7),  $n$  为像素矩阵的长宽， $p$  为 padding 的维度， $f$  为 filter 的长宽， $s$  为步长。

$$\left\lceil \left( \frac{n + 2p - f}{s} + 1 \right) \right\rceil \times \left\lceil \left( \frac{n + 2p - f}{s} + 1 \right) \right\rceil \quad (2.7)$$

上文都是展示的二维矩阵的卷积，也就是灰度值图像，但是在现代生活中，大部分的图像都是 RGB 的彩色图像，由 RGB 三个颜色通道叠加而成。

(5) 演示对于 RGB 图像的卷积：

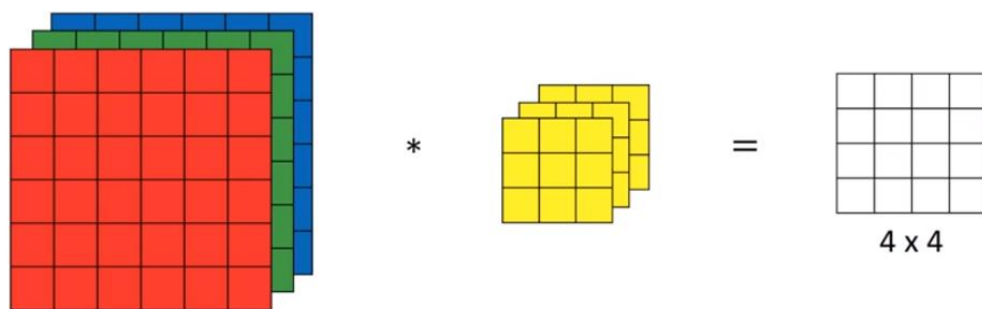
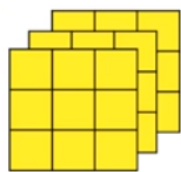


图 2-6 RGB 图像的卷积

左边为  $6 \times 6 \times 3$  的图像像素矩阵， $\text{height}=6$ ,  $\text{width}=6$ ,  $\text{channels}=3$ ,  $\text{channels}$  为 3 代表 RGB 三原色。RGB 是彩色图像的通用表示方法。通过三原色的矩阵叠加，可以产生 256 种强度的颜色图像。因为 Filter 为  $3 \times 3 \times 3$  的过滤器。最后得到的卷积结果为  $4 \times 4$  的矩阵。

可以把像素矩阵想象为一个立方体，filter 想象为一个小的立方体，以步长 1，不补零进行卷积。





(a)



(b)

图 2-7 RGB 将 filter 三维矩阵想象成一个小立方体

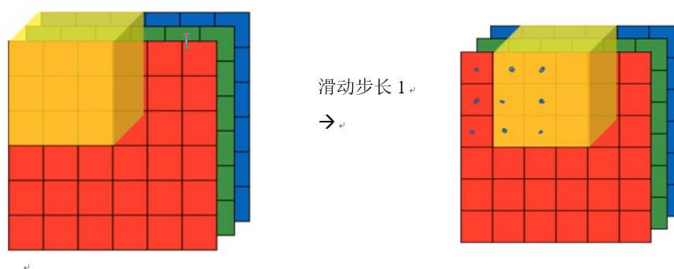


图 2-8 滑动 filter

将 filter 卷积图像的像素矩阵时，一次覆盖，3 的立方，即 27 个数字对应相乘并相加。以步长为 1，不断滑动小立方体，计算卷积。

对于灰度图像操作类似，检测彩色图像边缘，假设，检测红色通道的图像边缘，可以把卷积核设置为图 2-9 所示，将红色通道的过滤器设置为垂直边缘检测过滤器，将绿色通道和蓝色通道的过滤器置空。这样对于 RGB 图像的卷积，只会捕获到红色通道的垂直边缘。

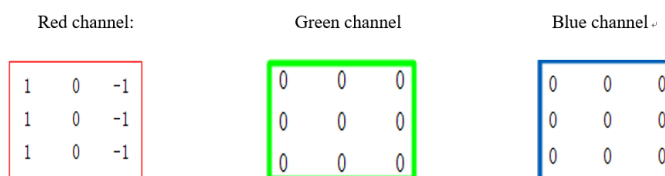


图 2-9 检测红色 channel 的边缘，设置 filter

若不关心是某种颜色的图像边缘，需检测所有颜色的图像垂直边缘，将过滤器设置为图 2-10 所示，将所有通道的过滤器都设置为检测垂直边缘。

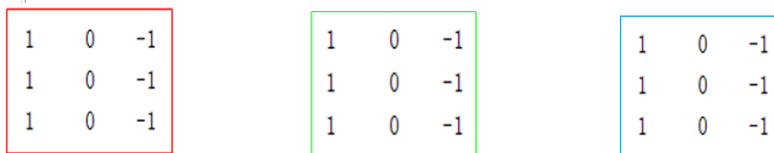


图 2-10 检测三色边缘，设置 filter

## (6) Summary

$$(n \times n \times nc) * (f \times f \times nc)$$

$$= \left( \frac{(n + 2p - f)}{s} \right) + 1 \times \left( \frac{(n + 2p - f)}{s} \right) + 1 \quad (2.8)$$

若将卷积步长补零全部考虑在内，卷积公式表示为式 2.8。N 为图像像素矩阵的长宽，nc 为 channel 通道数（灰度图像为 nc=1，RGB 彩色图像 nc=3，p 为 padding，f 为 filter 的维度，s 为卷积步长，nf 为 filter 的个数）。

## 2.2 池化层

池化有时域池化和空域池化。池化层在很多场景中都有效。它可以减少网络中的参数和计算量。以最常用的最大池化为例，每次选取区域中最大的值，提取出来放在新的矩阵。如果这块区域中有明显特征（矩阵像素值大），则会被提取出来。若区域中没有明显特征，则各个值都很小，提取出来的值也是偏小的。但是，池化的过程，特征矩阵缩小的太快，容易丢失大量的信息。在卷积层的操作中提到过，padding 有 same 和 valid 两种方式，填充 0 或者不填充 0。需要注意的是，在池化过程中，补 0 的操作很少见。在之后的池化层的意义小节中会详细演示池化的步骤。

- （1）池化类型：最大化池化、平均池化。
- （2）池化窗口大小 pool size：池化过滤器的矩阵尺寸。
- （3）池化步长 strides：池化窗口在图像上一次滑动的距离步长。
- （4）补零 padding：补零还是不补零。

池化层可以减小运算量，提高计算速度，同时提高所提取特征的鲁棒性，就即防止过拟合。对下面的矩阵进行 2\*2 的池化，将矩阵划分为 4 个区域，对每个区域取最大值，以 stride=2，以 2\*2 过滤器为例：

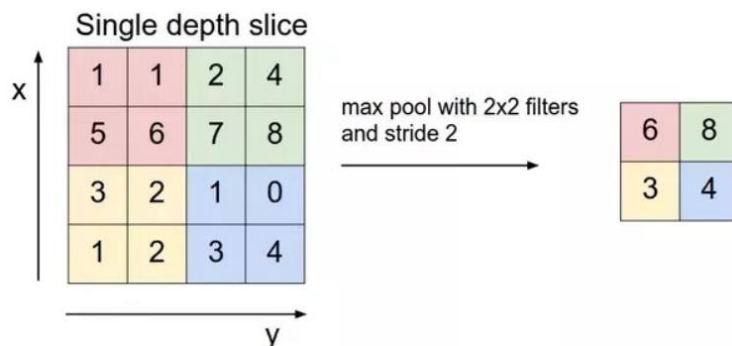


图 2-11 最大池化

图 2-11 演示的就是最大池化。最大池化，池化的超参数有 stride (池化步长)

和 `filter_size` (池化尺寸)。最大化的操作的功能就是只要在任何一个象限内提取到某个特征,它都会被保留在最大池化的输出中。所以最大池化的实际作用就是,如果过滤器提取到某个特征,那么保留其最大值。如果没有提取到这个区域中的特征,可能这个区域本来就不存在某个特征。例如上图的右上角全部都是很小的值,最大值也很小,所以可能本来就没有边缘特征。

池化就一组超参数, `stride` 卷积步长和过滤器的尺寸 `filter_size`, 这组参数不需要学习,在模型建立的时候确定,整个训练过程中就不会被改变。输出结果的维度,和卷积的公式一样,需要注意的是,一般来说,池化层是不需要补零的。如果是多个 `filter` 的池化,池化的结果也是维度也会是:

$$\left[ \left( \left( \frac{n-f}{s} \right) + 1 \right) \right] \times \left[ \left( \left( \frac{n-f}{s} \right) + 1 \right) \right] * nc' \quad (2.9)$$

$nc'$  代表 `filter` 的个数,  $f$  代表 `filter size`,  $s$  代表步长,  $nc'$  代表 `filter` 的个数。除了最大池化以外,还有平均池化 (`average pooling`),但是平均池化用到的很少。

### 2.3 全连接层

全连接层实际上是多层感知器应用于卷积神经网络末端之后别名。感知器是一种线性分类器,通过用一条直线将输入分成两类。输入的特征向量  $x$  与权重  $w$  相乘再与偏差  $b$  相加。 $Y = w * x + b$ 。感知器通过使用权重矩阵  $w$  形成线性组合(也可以对感知器的输出再使用非线性激活函数),可以基于多个输入产生单个输出,现在的全连接层一般和激活函数配合使用,可以产生非线性的决策边界,处理更复杂的非线性分类问题。

多层感知器意味着模型中有多个感知器组成。多层感知器是深度的人工神经网络。它由一个接受信号的输入层,一个产生决策函数或是预测的输出层,两者之间任意数量的隐藏层组成。见图 2-12,多层感知器常应用于监督性学习问题,训练一组带标记的输入数据,对于输入和输出的相关性进行建模。训练过程中为了最小化错误率,涉及到要调整的参数包括权重矩阵  $w$ 、偏差  $b$ 。

在使用最经典的多层感知器进行分类时,首先初始化权重矩阵  $w$ ,定义损失函数  $J$ ,比如均方误差或者交叉熵。模型训练的过程就是最小化损失函数的过程,一般来说,损失函数必须是凸函数,才有最优解。在多层感知器上不断迭代(普遍的是小批量梯度下降),更新  $w$  和  $b$ ,找到使  $J$  最小化的  $w$  和  $b$ 。

反向传播算法的目的是使权重和偏差根据误差去调整自身的值,误差可以通过很多种方式评估,均方误差 (RMSE) 也包括在内。目前,通常将多层感知器应

用于卷积神经网络的末尾端，通过若干卷积层和池化层之后，将矩阵压平为一维向量，作为全连接层的输入，最后通过全连接中的输出层输出一维标量，在分类的情况中，会使用 softmax 激活函数，直接输出对应的种类所对应的 one-hot 二进制编码。全连接层最大的特点是，每一层都和上一层的神经元完全连接。

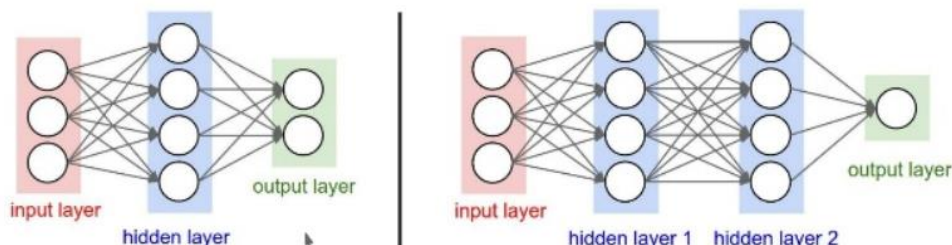


图 2-12 两层隐藏层的全连接层

图 2-12 展示了一层隐藏层的全连接层(左)和两层隐藏层的全连接层(右)。隐藏层的深度越深，捕获的特征更复杂，计算量也大幅提升，模型预测也会更准确。有研究表明，全连接层中需要训练的参数，在整个卷积神经网络中的参数比例为百分之八十左右。全连接中每个神经元都要参与到计算中，所有参数是完全连接的，计算成本也非常大。

## 2.4 激活函数

一般来说，对于卷积层和全连接层的输出使用激活函数<sup>[16]</sup>。Activation function 通常为非线性函数。在网络中加入了激活函数，让神经网络可以处理比较复杂的非线性问题<sup>[17]</sup>。

下面介绍一些常用的激活函数：

### ① Sigmoid<sup>[18]</sup> function

Sigmoid 函数，如下图 2-20：在 X 值-2 到 2 之间，Y 的值是非常陡的。这意味着，在那个区域的 X 值的任何微小变化都会导致 Y 的值发生显著变化<sup>[19]</sup>。这意味着这个函数有把 Y 值带到曲线两端的趋势。对于做分类器考虑它的属性不错，它倾向于将激活带到曲线的任一侧（例如， $x = 2$  以下和  $x = -2$  以下），对预测进行明确区分。激活函数 tanh 的输出（值域）始终在范围 (0, 1) 内。对于 sigmoid 函数的两端接近水平的部分，Y 值对 X 中的变化的响应往往非常小。意味着该区域的梯度将会很小，它引起了一个“渐变消失”的问题。那么当激活到达两边曲线的“近水平”区域附近时会发生什么？梯度很小或已消失（由于极小的值，不能做出重大改变）。网络拒绝进一步学习或者速度非常慢（取决于用例，直到梯度/计算受到浮点值限制的影响）。

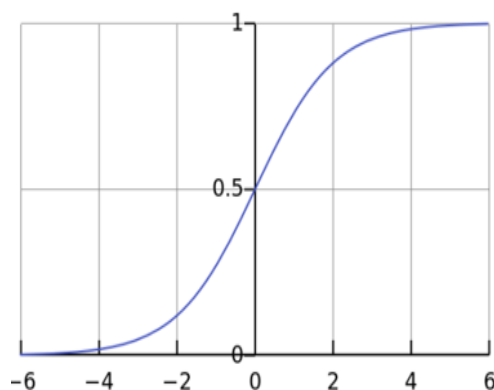


图 2-13 Sigmoid Function

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.10)$$

### ② $\tanh^{[20]}$ Function 双曲正切函数

Tanh 看起来和 sigmoid 函数很像。其实就是缩放版的 sigmoid 函数。它的值域在  $(-1, 1)$  范围内。需要注意的是,  $\tanh$  的梯度比 sigmoid 更强(导数更陡)。在 sigmoid 或  $\tanh$  之间做出决定将取决于对梯度强度的要求。像 sigmoid 一样,  $\tanh$  也有逐渐消失的梯度问题。一般来说,  $\tanh$  函数比 sigmoid 在大多数情况都表现的更好, 除了一种情况, 在进行二元分类的时候, 我们期望的预测的范围在  $(0, 1)$  之间, 而不是  $(-1, 1)$  之间, 在输出层使用 sigmoid 函数。

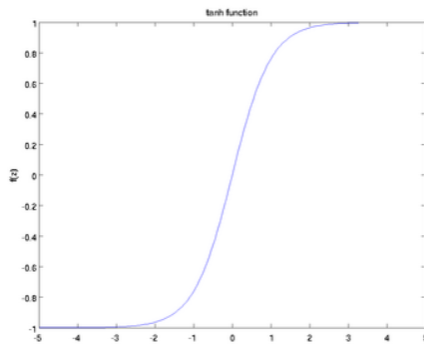


图 2-14 tanh 函数

$$\tanh(x) = \frac{e^x * e^{-x}}{e^x + e^{-x}} \quad (2.11)$$

### ③ reLu 函数

如图 2-22, 若  $x$  是正数, 则给出输出  $x$ , 否则为 0。起初看起来, 它更像是线性函数, 因为它在正轴上是线性的。首先, ReLu 本质上是非线性的, 而且 ReLu

的组合也是非线性的。实际上它是一个很好的逼近器，任何函数都可以用 ReLu 的组合来近似。ReLu 的范围是  $[0, \text{inf}]$ ，这意味着它可能会激活爆炸。

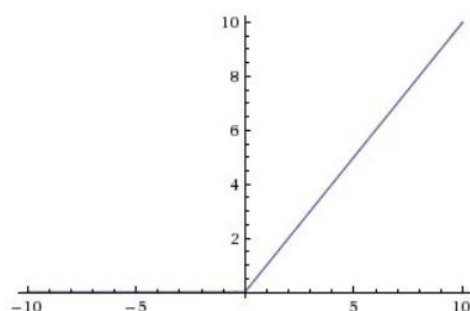


图 2-15 relu 函数

$$\text{relu}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (2.12)$$

另一点需要讨论的是激活的稀疏性。一个有很多神经元的大神经网络，激活是密集的，这很浪费资源。理想情况下，希望网络中的一些神经元不会激活，从而使激活变得稀疏而有效。ReLU 有这个好处。设想一个具有随机初始权值（或归一化）的网络，并且由于 ReLu 的特性（ $x$  的负值输出 0），几乎 50% 的网络会产生 0 激活。这意味着更少的神经元被激活（稀疏激活）并且网络更轻量化。但是没有什么完美的。包括 ReLu。由于 ReLu 的水平线部分（对于负  $x$  的部分），梯度趋向于 0。对于 ReLu 这部分区域的激活，梯度将为 0，因为在下降过程中权重不会被调整。这意味着，那些进入该状态的神经元将停止响应错误输入的变化（仅仅因为梯度为 0，没有任何变化）。这被称为死亡的 ReLu。这个问题会从仅仅几个神经元的错误和不相应，导致网络中相当一部分被动。通过简单地将水平线变为非水平分量，ReLU 中有各种可以减轻这个问题的方法。例如对于  $x < 0$ ,  $y = 0.01x$  将使其稍微倾斜而不是水平线。这是泄漏的 ReLu。还有其他的变化。主要想法是让梯度不为零，并在训练期间最终恢复。由于涉及更简单的数学运算，ReLU 在计算上比 tanh 和 sigmoid 时间花费更小。在设计深度神经网络时，这是一个很好的考虑。

那么，应该选择哪个激活函数呢？应该根据一定的条件，选择更合适的激活函数加速训练的过程。例如，对于二元分类来说，sigmoid 可以很好地工作，因为 sigmoid 的输出在 0 和 1 之间，在大多数情况下，tanh 都可以替代 sigmoid 函数，除了二元分类输出层的激活函数。在大部分情况下或不知道选择什么激活函数时，使用 ReLu 函数，在很多问题上都可以表现的很好。在多分类型神经网络中的最后一层，通常使用 softmax 激活函数。

## 2.5 参数选择

8 核机器下，以 mnist 图像集进行测试和验证：

调整参数：将通过调参，观察程序运行时间，训练集损失函数的大小 loss，训练集精确度 acc，验证集损失函数的大小 val\_loss、验证集精确度的大小 val\_acc 五个方面的结果，来判断该参数对于模型的影响。

本文定义 epochs 为梯度下降的批次，conv layer num 为卷积层的层数，filter 为过滤器的数量，kernel\_size 卷积核的尺寸，pool\_size 为池化窗口的尺寸，padding 代表是否填充零，activation 为激活函数。

（1）验证 epochs，梯度下降的批次对于模型的影响；

条件：batch\_size=128；conv layer num 1；filter=32；kernel\_size=3\*3；pool\_size=(2, 2)；padding=valid；activation=relu

实验结果如表 2-1 所示：

表 2-1 epochs：调参

epochs	Time	Loss	acc	Val loss	Val acc
6	38s * 6	0.0992	0.9711	0.0560	0.9811
12	38s * 12	0.0697	0.9797	0.0445	0.9842
18	38s * 18	0.0577	0.9828	0.0432	0.9747

结果分析：单批次的模型运行时间不变，总时长增加。训练集的损失函数值 loss 不断减小，训练集精度 acc 也在持续增加。验证集损失函数值 val\_loss 也是持续减少，测试集精度 val\_loss 从 epochs=6 到 12，精确度增加，但是在 epochs=18 时，val\_acc 却降低了。且 val\_acc 低于 acc，说明，在 epochs=18 时，模型出现了过拟合。

（2）验证过滤器数 filter number 对于模型的影响

实验结果如表 2-2 所示：

表 2-2 Mnist 改变 filter number

Filter	Time	Loss	acc	Val loss	Val acc
8	22s * 12	0.1496	0.9548	0.0799	0.9753
16	44s * 12	0.0946	0.9715	0.0527	0.9834
32	83s * 12	0.0685	0.9804	0.0439	0.9839

结果分析：可以看到过滤器的数目成倍数增加，模型训练的时间也是成倍增加。从 filter=8 到 16，精确度有较大的提升，但是从 filter=16 到 32，精确度的提升却不大，说明了，过滤器数目的增加，可以获得更多的特征，但是增加到一

定程度上，精确度将不再提升，甚至产生过拟合。因为，输入的图像已经没有更多的有效特征被提取，再继续增加过滤器只会提取一些无效特征。

### （3）验证 padding 补零对于模型的影响

条件：batch\_size=128；epochs=12；卷积层数 1；kernel\_size=3\*3；  
pool\_size=(2, 2)；activation=relu

实验结果如表 2-3 所示：

表 2-3 Mnist 改变 padding

padding	Time	Loss	acc	Val_loss	Val_acc
不补零 valid	38s * 12	0.0753	0.9778	0.0461	0.9837
补零 same	46s * 12	0.0693	0.9788	0.0432	0.9860

实验分析：补零后，精确度有小幅的提高，精确度从 0.9837 上升到 0.9860，模型训练时间略微增加，但是补零的效果比较好，因为如果不补零，可能图像边缘和角落的一些像素值只被触碰或计算一次，而中间区域的像素则被多次计算。补零可以让边缘和角落的像素也被使用。

### （4）验证卷积核大小 kernel\_size 对于模型的影响

条件：batch\_size=128；epochs=12；卷积层数 1；pool\_size=(2, 2)；filter=32；  
padding=valid；activation=relu；activation=relu

实验结果如见表 2-4 所示：

表 2-4 Mnist 改变 kernel size

kernel size	Time	Loss	acc	Val_loss	Val_acc
2 * 2	37s * 12	0.0905	0.9737	0.0574	0.9823
3 * 3	37s * 12	0.0763	0.9773	0.0448	0.9841
6 * 6	37s * 12	0.0538	0.9840	0.0345	0.9888

结果分析：一般来说，kernel size 越大，遗留的特征越多，kernel size 越小，提取的特征就越精细。在 mnist 的数据集的实验结果中，卷积核尺寸变大，精确度上升，但是在普遍情况中，还是使用 3\*3 的卷积核尺寸。

### （5）验证激活函数 activation(对比 sigmoid、tanh、relu)对于模型的影响

条件：batch\_size=128；epochs=12；卷积层数 1；kernel\_size=3\*3；  
pool\_size=(2, 2)；padding=valid；activation=relu

实验结果如表 2-5 所示：



表 2-5 Mnist 改变 activation

Activation	Time	Loss	acc	Val loss	Val acc
Relu	39s * 12	0.0780	0.9768	0.0463	0.9839
Tanh	39s * 12	0.0895	0.9724	0.0627	0.9793
Sigmoid	40s * 12	0.2670	0.9200	0.1882	0.9445

结果分析：可以发现在图像分类上，relu 和 tanh 比 sigmoid 表现的更好。

#### (6) 验证卷积层数对于模型的影响

条件：Batch\_size=128; epochs=12; kernel\_size=3\*3; pool\_size=(2, 2);

padding=valid; activation=relu

实验结果如表 2-6 所示：

表 2-6 Mnist 增加卷积层数

Conv layer num	Time	Loss	acc	Val loss	Val acc
1	38s * 12	0.0706	0.9795	0.0446	0.9849
2	140s * 12	0.0313	0.9906	0.0313	0.9895
3	380s * 12	0.0233	0.9923	0.0253	0.9930

结果分析：从实验结果可以看出，增加卷积层数，精确度在上升。但是也不能盲目的增加卷积层数来提高精确度，因为卷积操作在模型中非常耗时，可以看到增加卷积层数后，时间成本非常大，且多次卷积后，之后的卷积层可能没有更多的特征可以提取，那么之后的卷积都只是无效操作。所以，在实际的训练中，对于卷积的结果进行可视化，观察卷积后的结果中是否还存在有效特征。

## 2.6 本章小结

除了本章讲的卷积、池化层全连接层，在神经网络中，还有一些常用的操作。

**压平 flatten：** Flatten 用来将输入“压平”，即把多维的输入一维化，常用在从卷积层到全连接层的过渡。因为，在预测时，最后的预测输出应该是一个标量。**Flatten** 不影响 batch 的大小。**Dropout：** 为了防止过拟合，在周期性的进行卷积和池化之后，会加入 Dropout 操作，丢弃一些参数。

本章介绍了参数和超参数的概念。CNN 的组成，是由若干层卷积层、池化层交替、以及少量的全连接层构成，本文详细叙述各个层的意义，并验证了其中不同超参数改变对于模型的意义。紧接着，从精确度和效率的平衡性和性价比的角度出发，调整不同的参数和超参数，对 mnist 和 cifar 两个数据集的参数进行调整，达到实验中的最好效果，最后从数学和矩阵的角度，演示了卷积和池化的效果，思考了卷积层和池化层对于 CNN 卷积神经网络模型的意义。在第三章，会基于 FCN 全卷积神经网络，对 mnist 和 cifar 数据集进行讲述。

### 第三章 全卷积神经网络的基本原理

本章主要介绍 FCN 全卷积神经网络，在 CNN 中，模型由卷积层、池化层、以及全连接层构成。卷积层用于检测图像边缘，自动提取图像特征，池化层用于简少参数和计算量，提取重要特征并防止过拟合。而在 FCN 中，会用卷积层替代池化层和全连接层。在这里从精确度和时间效率两方面分析 FCN。

#### 3.1 全卷积神经网络架构

一个全卷积神经网络可以接受任意大小的输入，通过有效的推理和学习产生和输入相同大小的输出<sup>[21]</sup>。在 CNN 中，每个像素点用其周围的对象或区域类别进行标注。但这种方法不管是在速度上还是在精度上都有很大的不足。而在 FCN 中，对于每一个像素进行分类和预测。针对语义分割(识别出物体并分类)训练一个端到端，点对点的网络。

FCN 技术：

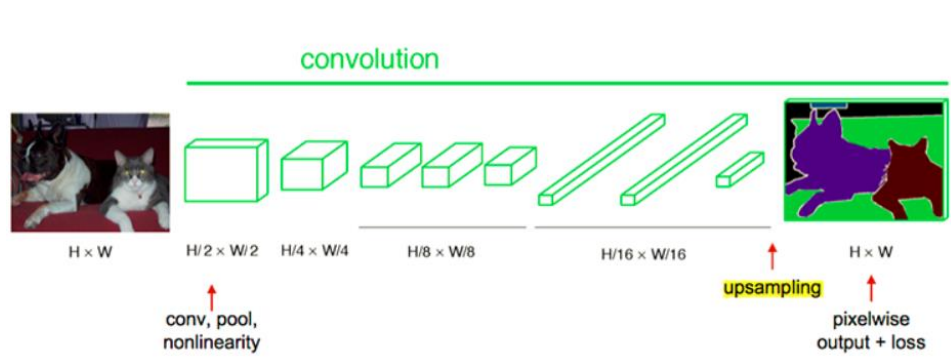


图 3-1 全卷积神经网络结构图

##### 1. 卷积化<sup>[22]</sup>

一般来说，图像分类网络通常在最后都是连接全连接层，进入全连接时，会使用 **flatten**，将矩阵压成一维向量。从而丢失了空间信息，最后训练输出一个标量，也就是分类标签。

而语义分割的输出则需要是一个分割图，且不论尺寸的大小，但是至少是二维的。所以，使用卷积层替代全连接层，也就是卷积化。

##### 2. 上采样<sup>[23]</sup>（upsampling）

在一般的 CNN 结构中，通常使用池化层缩小图片的大小。在 FCN 中，需要得到的是与原图像大小相同的分割图，因此需要对最后一层进行上采样，也称为反卷积(Deconvolution)，也叫做转置卷积。(conv\_transpose)。

在卷积中，每一层的数据都是三维的数组  $h \times w \times d$ ,  $h$  和  $w$  是图像的尺

寸,  $d$  是特征或通道(channel)维度。FCN 的第一层是图片, 像素大小为  $h \times w$ ,  $d$  为颜色通道(channel)。高层中的位置关联的是图像中对应的像素位置, 称作接受域 receptive field。

卷积网络建立在平移不变性上, 它的基础组成(卷积, 池化和激活函数)操作输入的像素区域(filter size 大小的区域)。由于卷积过程中仅与它的领域相关, 所以依赖于像素点在图像矩阵中的相对位置。用  $X_{ij}$  表示在某个特定层上, 数据向量  $(i, j)$  位置的点,  $Y_{ij}$  表示下一个层。

$$Y_{ij} = \text{image pixels matrix} * \text{filter} \quad (3.1)$$

FCN 自动对任何大小的输入进行操作, 并产生相应的空间维度的输出。接下来解释, 如何将一个分类网络通过生成粗略的输出图转换成一个卷积网络。对于像素级的预测, 需要将这些缩略图输出连接回像素。

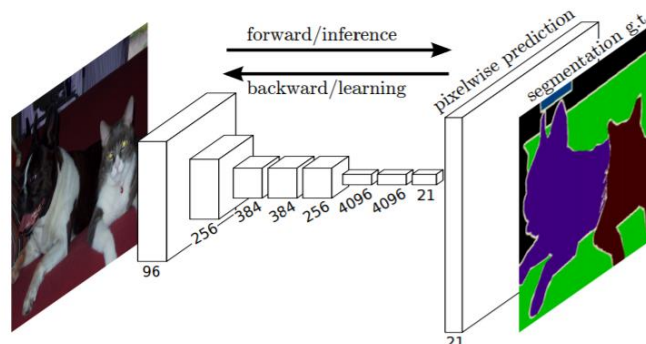


图 3-2 全卷积神经网络结构图及反卷积

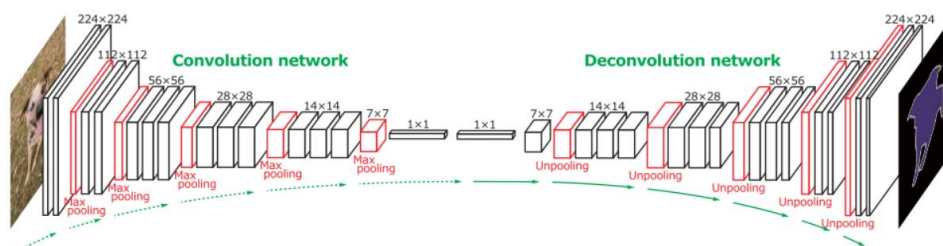
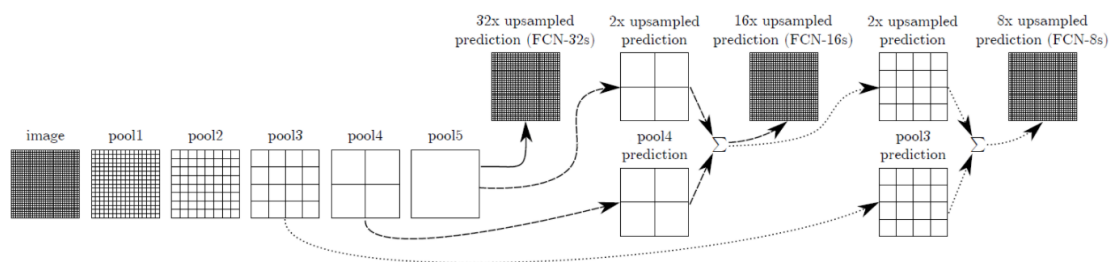


图 3-3 全卷积神经网络结构图 2

### 3. 跳级(skip)结构<sup>[24]</sup>

对 CNN 的结果做处理, 得到了全连接的预测, 得到的分割结果比较粗糙, 所以考虑加入更多前层的细节信息, 也就是把倒数第几层的输出和最后的输出做一个 fusion, 实际上也就是加和。



实验表明，这样的分割结果更细致更准确。在逐层fusion的过程中，做到第三行再往下，结果又会变差，所以作者做到这里就停了。

图 3-4 全卷积神经网络的跳级结构

典型的识别网络，LeNet、AlexNet 表面上都是接受固定长度的输入，并且产生非线性的输出。网络中的全连接层有固定的维度并且会丢掉一些空间坐标。与 CNN 一样，但是由于替换了池化层和全连接层。所以只有卷积层相关参数与梯度下降参数需要调节。简单来说，有梯度下降的批次 epoch、小梯度下降每次下降的 batch\_size、过滤器 filter kernel size、filter kernel numbel、卷积是否补零 padding valid or same、激活函数 activation、卷积层数。

### 3.2 卷积层替换池化层

在第四章，通过实验证明了用卷积步长为 2 的卷积层替代池化层之后，精确并没有损失。池化层在于 CNN 中的主要作用，一个适用于降维，减少训练的参数，避免过拟合，另一个是提取子区域中的主要特征（比如最大池化）。

假设  $10 \times 10$  的二维矩阵，过滤器尺寸为  $3 \times 3$ 。在使用  $2 \times 2$  的 filter 进行步长为 2 的池化时，输出为  $5 \times 5$  的特征矩阵。本文用步长为 2 的卷积层替代时，卷积步长为 2，那么卷积的输出结果是  $((10-3)/2) + 1 = 4$ ，输出维度为  $4 \times 4$  的特征图。相比于  $10 \times 10$  的像素矩阵，步长为 2 的卷积层也做到了降维。并且减少了要训练的参数。

那么，下一个问题是，卷积层是否可以像池化层以下，提取出图像中的重要特征？在 CNN 架构中，卷积层的目标是检测图像边缘，从图像中提取的边缘、角度信息。那么自然而然的，通过实验来观察，再替换池化层之后，模型的精确度如何变化。在第四章，予以证明。

### 3.3 卷积层替代全连接层

全连接层即下一层的每一个神经元都与上一层的全部连接<sup>[25]</sup>，本节定义输入为  $X$ ， $x$  和隐藏层之间有一个权重矩阵，可以把每一个神经元看做一个特征，那么每个特征对最终结果的影响不一样，所以需要有一个权重矩阵，来定义不同特征对于下一层的影响的权重。

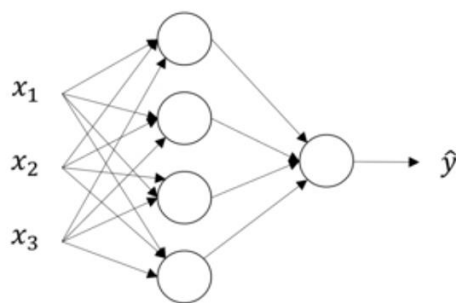


图 3-5 一层隐藏层的全连接层

全连接层的输出结果为： $z = \text{activation\_function}(w'x + b)$

其中， $w'$  为权重矩阵的转置。

全连接层为了让输入层、隐藏层和输出层之间完全连接，它必须要接受固定长度的输入对应神经元的数量。但是卷积层，是在一个大矩阵上依次滑动，所以它可以接受任何大小的输入。

考虑卷积的过程： $\text{activation\_function}(n * f + b)$ 。

可以发现，卷积和全连接的过程都是相乘再相加的过程。

### 3.4 本章小结

本章，首先在原理介绍了 FCN。叙述了全卷积神经网络的框架，是在 CNN 的基础上，改进而来。将 CNN 中的池化层和全连接层由卷积层替代<sup>[26]</sup>。并在原理上，表达了为什么可以被替代。在下一章将通过实验数据进一步论证 FCN 的可行性。

## 第四章 基于全卷积神经网络的分类

### 4.1 mnist 和 cifar-10 数据集介绍

本文使用公开的数据集 mnist 手写数字图像集和 cifar-10 数据集,比较 FCN 和 CNN 在两者上的表现。mnist 包含 60000 个样本的灰度值图像, cifar-10 包含 60000 个样本的 RGB 彩色图像。关于图像集的对比见表 4-1。在图像集大小上, cifar-10 更大,几乎是 mnist 数据集 17 倍左右。在图像包含内容上, cifar-10 是内容更丰富的 RGB 彩图。

表 4-1 Mnist 和 cifar 图像集对比

	Mnist 数据集	Cifar 数据集
Size	11.5MB	175.5MB
颜色格式	灰度图像	RGB 图像
数据内容	手写数字图像	飞机、车等十类图像

#### 4.1.1 minst 图像集

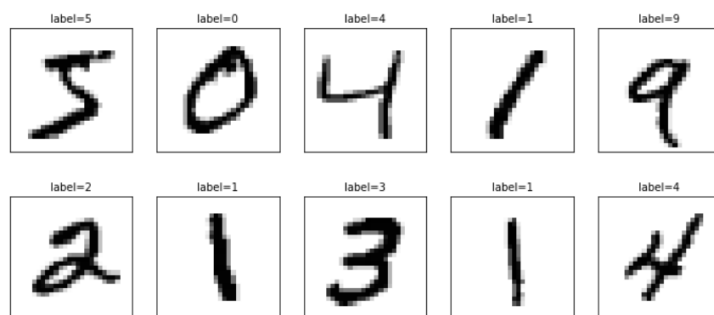


图 4-1 mnist 手写数字训练集的前十笔数据展示

图 4-1 是 mnist 数据集的前十笔数据,上标是关于此图片对应的标记,即真实值。将 mnist 数据集 60000 笔数据按比例划分,训练集样本 50000 笔,测试集样本 10000 笔,见表 2。图片的维度为  $28 * 28 * 1$  的灰度图。

表 4-2 mnist 训练测试样本划分

总数据样本	60000
训练集样本	50000
测试集样本	10000

### 4.1.2 cifar-10 图像集

表 4-3 各个数字对应种类


airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
0	1	2	3	4	5	6	7	8	9
									

图 4-2 cifar-10 训练集前十笔数据展示

为了简化代码和表达，将 cifar-10 十种类型分别对应到 0~10 个数字。对应详情见表 4-3，展示 cifar-10 前十笔数据，根据上标可以知道该图片的标记值及对应编号。同理，将 cifar-10 的 60000 笔数据按比例分类，得到训练集和测试集。cifar-10 数据集的图片大小为  $32 * 32 * 3$  的 RGB 彩色图，见表 4-4。

表 4-4 cifar-10 样本数

总数据样本	60000
训练集样本	50000
测试集样本	10000

### 4.2 mnist 数据集处理结果

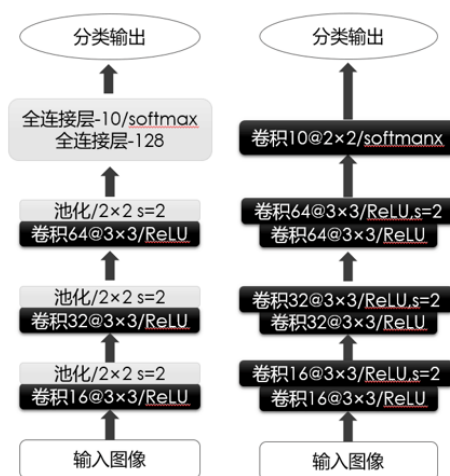


图 4-3 mnist 数据集 CNN(左)和 FCN(右)实现模型

CNN 的模型由图 4-3（左）所示，由三个交替的卷积层和池化层构成，最后连接全连接层，128 个的神经元组成的输入层和隐藏层，10 个的神经元的输出层，并通过 softmax 激活函数。FCN 即在 CNN 的基础上，将步长为 2 的池化层由步长为 2 的卷积层替代，全连接层由一层 10 个过滤器的卷积层（激活函数为 softmax）替代。预测结果前十笔展示，见图 4-4，label 为其标记的真实值，predict 为模型预测值。

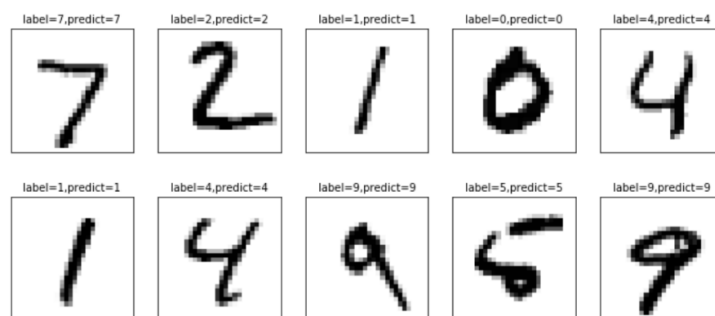


图 4-4 FCN 预测结果部分展示

predict	0	1	2	3	4	5	6	7	8	9
label										
0	979	1	0	0	0	0	0	0	0	0
1	0	1130	3	1	0	0	0	1	0	0
2	0	0	1029	0	0	0	0	3	0	0
3	0	0	0	1005	0	4	0	0	1	0
4	0	0	0	0	975	0	3	0	0	4
5	1	0	0	6	0	884	1	0	0	0
6	8	3	0	1	2	1	941	0	2	0
7	2	3	6	0	0	0	0	1013	1	3
8	3	0	1	1	0	1	0	0	965	3
9	2	0	0	2	3	2	0	1	0	999

图 4-5 FCN mnist 预测结果的混淆矩阵

混淆矩阵，在图像分类中非常有用。如果想要了解，训练的模型中，哪些数字预测的准确率更高，哪些数字在预测中最容易产生混淆。可以使用混淆矩阵。从而，可以判断，预测最容易混淆的数字的原因，是两者在本质上就有相似还是模型自身的问题。从而去解决混淆对象的具体问题。

从上图 4-5 的混淆矩阵可以发现，数字 1，被预测为 1 的数目最高，说明 1 最不容易被混淆。数字 5，被预测成 5 的数目最少，只有 884，并且有 4 笔数据被预测成 3，说明数字 5 容易被误判为数字 3。

其他参数一样的情况下，FCN 和 CNN 的训练过程可视化。CNN mnist 数据集的训练过程：



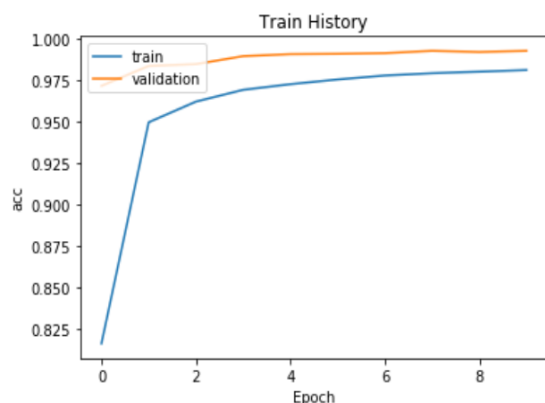


图 4-6 cnn 精确度

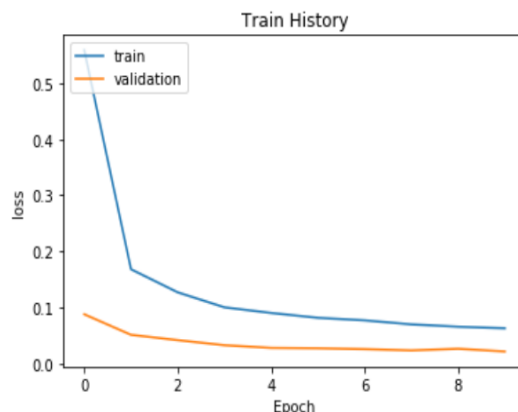


图 4-7 cnn loss 函数

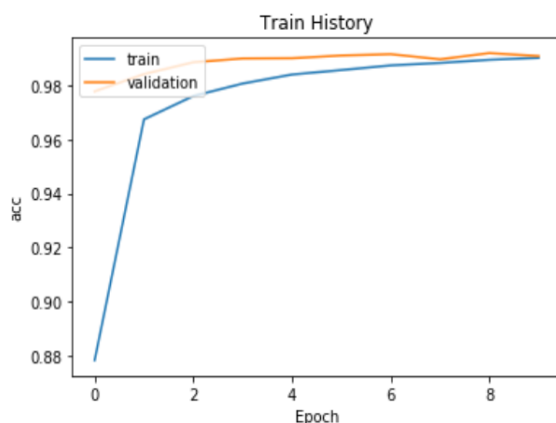


图 4-8 fcn 精确度

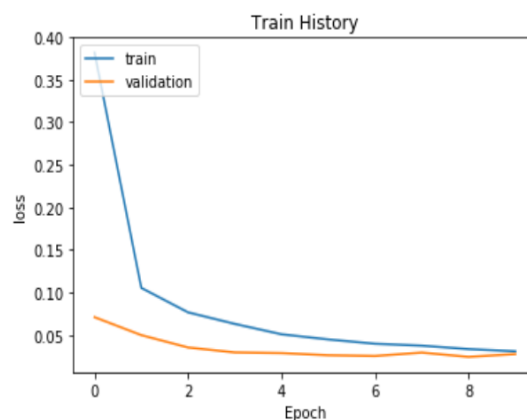


图 4-9 fcn loss 函数

可视化模型的训练过程，图 4-6 可视化 CNN 模型的精确度随训练批次变化过程，可以发现精确度在  $\text{epoch} < 4$  之前，精确度提升很快，之后只有小幅的提高，在  $\text{epoch} > 8$  的区间，精确度几乎并无变化。从而，在目前的条件下，可以知道，单纯的继续增加训练批次，对模型并不能在继续产生较大影响。根据精确度和损失函数的随训练批次的变化曲线，可以在调整模型训练过程中的一些参数，根据不同的参数判断模型如何才能表现的更好。是需要更多次迭代，计算出使损失函数更小更优的权重矩阵  $W$  和偏差  $b$ ，还是增加大量训练批次也无法获得精确度的较大提升，是否需要放弃通过批次提升精确度的方法。

除此之外，通过对比训练集精确度和测试集精确度，可以判断模型是否出现了过拟合。因为模型根据训练集的数据计算而来，一般来说训练集精确度会高于测试集。但是评估一个模型的好坏关键在于，它对于没有见过的新实例，有如何的表现，也称为模型的泛化能力。若模型对于没有见过的数据，也能做出更好更准确的判断，则称作优秀的模型。根据图 4-6 和图 4-7，发现测试集精确度反而

高于训练集，说明模型没有产生过拟合。图 4-8 和图 4-9，同样，可以看出精确度和损失函数随 epoch 的增加的变化情况。且根据训练集和测试集的精确度对比，可以发现模型存在轻微的过拟合。

根据判断模型是否产生了过拟合，可以增加一些措施，例如使用更适合当前需求的激活函数，丢弃更多比例的参数，增加正则化方法等。

### 4.3 cifar-10 数据集处理结果

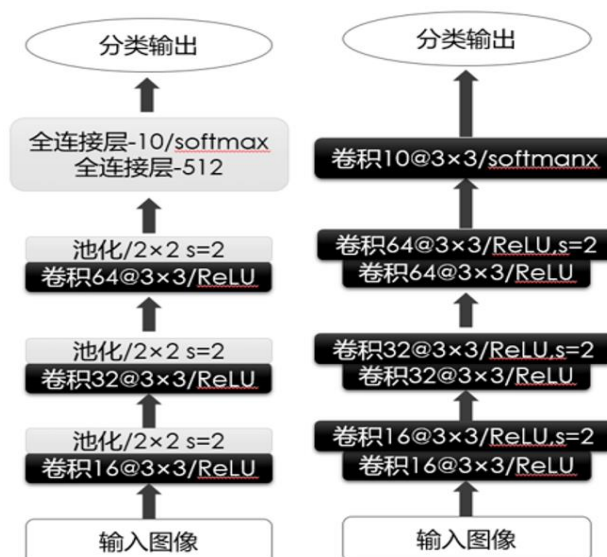


图 4-10 cifar-10 数据集 CNN(左)和 FCN(右)实现模型

CNN 和 FCN 训练模型见图 4-10，将全连接层的神经元增加到了 512 个，因为 cifar-10 是 RGB 彩色图像，且图像包含内容更加丰富，图像包含的特征更多。同样，FCN 模型中用卷积层替代了池化层和全连接层。



图 4-11 cifar-10 预测结果部分展示

cifar-10 数据集前十笔数据预测结果如图 4-11，上标表明了图像的真实值和预测值，如第一行第一列的图像真实值为 cat 猫，预测值为 truck 卡车，预测结

果错误。第一行第四列的图像真实值标记为 airplane 飞机, 预测值也为 airplane 飞机, 则预测正确。

predict	0	1	2	3	4	5	6	7	8	9
label										
0	649	55	1	2	2	4	0	6	93	188
1	18	735	1	0	0	3	0	4	17	222
2	173	46	89	27	2	245	3	82	41	292
3	92	59	3	86	3	226	4	68	49	410
4	43	42	10	16	42	170	3	177	48	449
5	36	33	0	26	0	547	1	61	22	274
6	36	102	7	36	7	79	114	53	104	462
7	23	16	2	8	1	107	0	547	17	279
8	103	70	0	1	0	2	0	6	564	254
9	26	59	0	2	0	1	0	8	8	896

图 4-12 cnn 混淆矩阵

预测结果的混淆矩阵见图 4-12, 可以发现飞机容易被预测为鸟和船, 摩托车易错误预测为青蛙, 狗易错误预测为鸟、猫、蜜蜂等。CNN 和 FCN 在 Cifar-10 数据集训练过程, 精确度和损失函数可视化, 见下图。

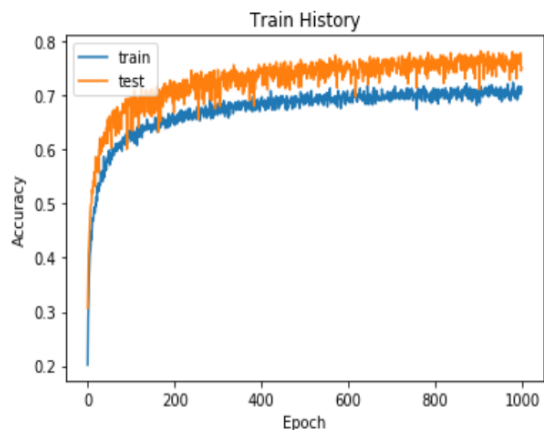


图 4-13 cnn 精确度

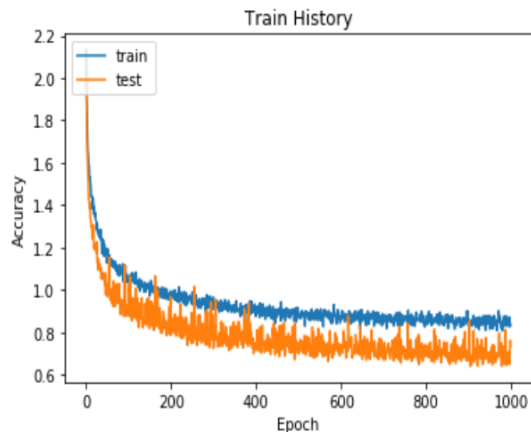


图 4-14 cnn loss function

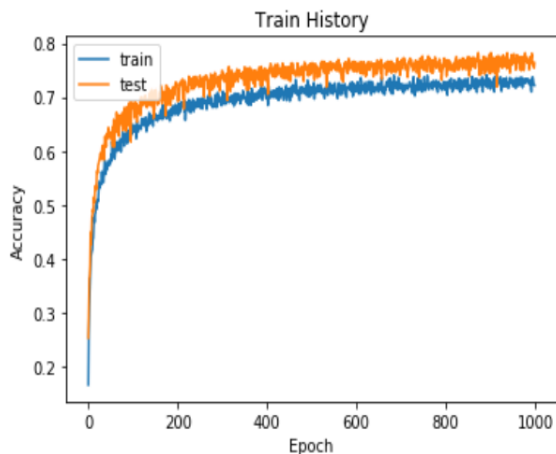


图 4-15 fcnn 精确度

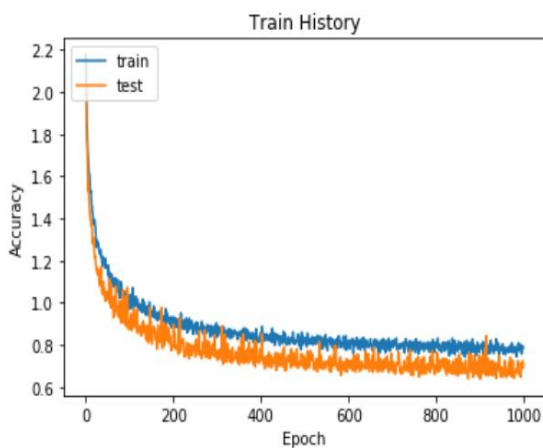


图 4-16 fcnn 精确度

表 4-5 CNN 和 FCN 在 mnist 和 cifar 数据集训练结果比较

algorithm	val_acc	time	dataset	epochs	params
CNN	0.9911	410s	mnist	10	98442
FCN	0.9925	490s	mnist	10	74362
CNN	0.7512	10000s	cifar	1000	553514
FCN	0.7632	8000s	cifar	1000	79184

表 4-5 展示了 CNN 和 FCN 在 mnist 和 cifar-10 数据集上，在时间和精确度以及训练参数的具体数字。

在 mnist 小数据集下，FCN 和 CNN 中的参数数量，没有很大的区别。①在精确度上：FCN 比 CNN 略高一些。②在时间效率上：FCN 参数更少，但是由于没有缺少了池化层的降维，反而运行时更长。并没有体现出参数共享的优势。

在 cifar 这种较大一点的数据集上，CNN 中的参数相比于 FCN 有了一个数量级的增加。此时参数共享的优势体现出来了。①在精确度上：FCN 模型的训练结果在精确度上比 CNN 略高。②在时间效率上：在梯度下降的批次比较大的时候，FCN 训练模型所需时间比 CNN 有大幅减少。

## 1. CNN 和 FCN 模型图和各层参数展示

### (1) Mnist 手写数字图像集对比

表 4-6 CNN model summary

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 16)	0
dropout_1 (Dropout)	(None, 14, 14, 16)	0
conv2d_2 (Conv2D)	(None, 14, 14, 32)	4640
max_pooling2d_2 (MaxPooling2D)	(None, 7, 7, 32)	0
dropout_2 (Dropout)	(None, 7, 7, 32)	0
conv2d_3 (Conv2D)	(None, 7, 7, 64)	18496

max_pooling2d_3 (MaxPooling2)	(None, 3, 3, 64)	0
dropout_3 (Dropout)	(None, 3, 3, 64)	0
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 128)	73856
dropout_4 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

表 4-7 FCN model summary

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 28, 28, 16)	160
conv2d_9 (Conv2D)	(None, 13, 13, 16)	2320
dropout_4 (Dropout)	(None, 13, 13, 16)	0
conv2d_10 (Conv2D)	(None, 13, 13, 32)	4640
conv2d_11 (Conv2D)	(None, 6, 6, 32)	9248
dropout_5 (Dropout)	(None, 6, 6, 32)	0
conv2d_12 (Conv2D)	(None, 6, 6, 64)	18496
conv2d_13 (Conv2D)	(None, 2, 2, 64)	36928
dropout_6 (Dropout)	(None, 2, 2, 64)	0

conv2d_14 (Conv2D)	(None, 1, 1, 10)	2570
flatten_2 (Flatten)	(None, 10)	0
=====		

## (2) Cifar 图像集

表 4-8 CNN model summary

Layer (type)	Output Shape	Param #
=====		
conv2d_5 (Conv2D)	(None, 32, 32, 16)	448
activation_7 (Activation)	(None, 32, 32, 16)	0
max_pooling2d_5 (MaxPooling2	(None, 16, 16, 16)	0
dropout_6 (Dropout)	(None, 16, 16, 16)	0
conv2d_6 (Conv2D)	(None, 16, 16, 32)	4640
activation_8 (Activation)	(None, 16, 16, 32)	0
max_pooling2d_6 (MaxPooling2)	(None, 8, 8, 32)	0
dropout_7 (Dropout)	(None, 8, 8, 32)	0
conv2d_7 (Conv2D)	(None, 8, 8, 64)	18496
activation_9 (Activation)	(None, 8, 8, 64)	0
max_pooling2d_7 (MaxPooling2)	(None, 4, 4, 64)	0
dropout_8 (Dropout)	(None, 4, 4, 64)	0

flatten_2 (Flatten)	(None, 1024)	0
dense_3 (Dense)	(None, 512)	524800
activation_10 (Activation)	(None, 512)	0
dropout_9 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 10)	5130
activation_11 (Activation)	(None, 10)	0
=====		

表 4-9 FCN model summary

Layer (type)	Output Shape	Param #
=====		
conv2d_1 (Conv2D)	(None, 32, 32, 16)	448
conv2d_2 (Conv2D)	(None, 15, 15, 16)	2320
dropout_1 (Dropout)	(None, 15, 15, 16)	0
conv2d_3 (Conv2D)	(None, 15, 15, 32)	4640
conv2d_4 (Conv2D)	(None, 7, 7, 32)	9248
dropout_2 (Dropout)	(None, 7, 7, 32)	0
conv2d_5 (Conv2D)	(None, 7, 7, 64)	18496
conv2d_6 (Conv2D)	(None, 3, 3, 66)	38082
dropout_3 (Dropout)	(None, 3, 3, 66)	0

conv2d_7 (Conv2D)	(None, 1, 1, 10)	5950
flatten_1 (Flatten)	(None, 10)	0

## 2. FCN 和 CNN 应用场景比较和探讨

在实践中,在使用卷积层替代了池化层和全连接层之后。在较小的数据集上:两者的差异并不大,FCN 的精确度比 CNN 略高一些,时间效率上越低一点。但是在较大数据集上,卷积过程中参数共享这个优势体现了出来。FCN 不仅在训练时间比 CNN 少了许多,在精确度上略高于 CNN。所以,得出以下结论:

(1) 如果数据集比较小: CNN 和 FCN 都能胜任

(2) 若数据集较大,选择 FCN 能让你的模型在精确度是时间效率上都更有优势。

## 4.4 本章小结

使用了最简单的 FCN 训练一个 mnist 手写数字集。因为 FCN 只包含了卷积层,所以参数调整和 CNN 中卷积层的调参类似。就没有再次叙述。通过实验发现,在替换之后不仅在精确度上有了提高,并且由于卷积层参数共享的特征,相比于 CNN 的参数有大量的减少。计算速度也得到了提升。

随后对于 CNN 和 FCN 在 mnist 和 cifar 数据集上的实验表现做了一个对比。得出了以下结论:

在数据集比较小的情况下: FCN 和 CNN 的差异并不大。在数据集较大的情况下,FCN 更适合处理较大数据集,在精确度和时间效率上都更有优势。

下一章,对于编程语言和编辑器的选择以及算法的实现和结论做出了一定的总结。



## 第五章 总结与展望

### 5.1 论文总结

关于编程语言和编辑器感想：

为什么使用 python, Python 易于上手, 且包含了大量的内置库。许多可以用于人工智能和机器学习。比如 Tensorflow (高级神经网络库), scikit-learn (用于数据挖掘, 数据分析和机器学习), pylearn2 (比 scikit-learn 更灵活)。对于别的语言, 学生和研究人员在接触 AI 或者 ML 时, 还需先接触这门语言。但是在 python 上, 就不会有这样的状况。所以, 我选择了 python, 将更多的精力放在算法的实现上, 而不是编程语言。

为什么使用 jupyter notebook, (1)jupyter notebook 易于安装, 支持 markdown 和 code 同时编写。(2) 由于 jupyter 便于远程控制, 我可以在本机进行远程服务器上的开发。

关于 CNN 和 FCN 算法总结:

(1) CNN: 通常由卷积层, 池化层和网络尾端的全连接层构成。

(2)FCN: 则是在 CNN 的基础上, 用卷积层替代了其中的池化层和全连接层。实现的是像素级别的分类, 实现了端到端的学习。FCN 常用于语义分割的问题, 也就是 where and what 的问题。

相比于 CNN, FCN 在以下三方面更棒:

(1) 输入图片尺寸: 如果网络中没有全连接层, 则可以接受几乎任何尺寸的图像。因为只有全连接层需要特定大小的输入。

(2) 像素空间信息: 全连接层通常会造成大量的空间信息丢失。因为每一个神经元都与上一层相连, 这种架构不能用于分割。

(3) 计算成本: 由于卷积层要训练的参数更少。所以计算成本更低。

(4) 精确度: 相比于 CNN, FCN 在大数据集上的精确度略高一些。

关于 CNN 和 FCN 图像分类的应用场景总结: FCN 在大数据集上, 相比于 CNN 在时间和精确度上都更有优势。

### 5.2 展望

在本文中, 仅仅使用了最简单的 CNN 和 FCN 结构作为比较。而并没有找到两者让数据集正确率最优化的解决方案。而且, cifar-10 并不算很大的数据集。由于条件和机器有限, 只选择了 cifar-10 和 mnist。

在未来学习的日子中, 将不断学习, 善于总结, 找到更多的调参经验和方法。并深入理解各种神经网络的各类架构, 以及相关的数学原理。

不断学习, 不断进步。将神经网络用于更多的领域, 而不仅仅是图像分类,

将网络应用于时间序列，目标检测等各个领域。

## 附 录

部分代码如下：

CNN mnist 代码实现：

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import os

batch_size = 128
num_classes = 10
epochs = 10
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_mnist_trained_model_epoch.h5'

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) =
mnist.load_data('/home/lhadmin/imageclassify/app/mnist.npz')

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(16, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape,
                 padding='same',
                 ))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
               optimizer=keras.optimizers.Adam(),
               metrics=['accuracy'])
print(model.summary())
train_history = model.fit(x_train, y_train,
                          batch_size=batch_size,
                          epochs=epochs,
                          verbose=1,
                          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)

# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

FCN mnsit 代码实现：

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import os

batch_size = 128
num_classes = 10
epochs = 10
```

```

save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_mnist_fcn_trained_model_code.h5'

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train,          y_train),          (x_test,          y_test)          =
mnist.load_data('/home/lhadmin/imageclassify/app/mnist.npz')

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(16, kernel_size=(3, 3),
                activation='relu',

```

```
        input_shape=input_shape,
        padding='same',
    ))
model.add(Conv2D(16, kernel_size=(3,3), activation='relu', strides=2))
model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu', strides=2))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu', strides=2))
model.add(Dropout(0.25))

model.add(Conv2D(10, (2, 2), activation='softmax'))
model.add(Flatten())

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
print(model.summary())
train_history = model.fit(x_train, y_train,
                        batch_size=batch_size,
                        epochs=epochs,
                        verbose=1,
                        validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)

# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

CNN 在 cifar 数据集上的实现:

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os

batch_size = 32
num_classes = 10
epochs = 1000
data_augmentation = True
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_cifar10_trained_model_epoch_1000.h5'

# The data, split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(16, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
```



```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])

print(model.summary())
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```

if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False, # set input mean to 0 over the dataset
        samplewise_center=False, # set each sample mean to 0
        featurewise_std_normalization=False, # divide inputs by std of the dataset
        samplewise_std_normalization=False, # divide each input by its std
        zca_whitening=False, # apply ZCA whitening
        rotation_range=0, # randomly rotate images in the range (degrees, 0 to 180)
        width_shift_range=0.1, # randomly shift images horizontally (fraction of total
width)
        height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
        horizontal_flip=True, # randomly flip images
        vertical_flip=False) # randomly flip images

    # Compute quantities required for feature-wise normalization
    # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_train)

    # Fit the model on the batches generated by datagen.flow().steps_per_epoch
    train_history = model.fit_generator(datagen.flow(x_train, y_train,
                                                    batch_size=batch_size),
                                       steps_per_epoch=128,
                                       epochs=epochs,
                                       validation_data=(x_test, y_test),
                                       workers=4)

```

```
# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)
```

```
# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

FCN 在 cifar 数据集上的实现：

```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
import os

batch_size = 32
num_classes = 10
epochs = 1000
data_augmentation = True
num_predictions = 20
save_dir = os.path.join(os.getcwd(), 'saved_models')
model_name = 'keras_cifar10_trained_model_fcn_epochs_1000.h5'

# The data, split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
```

```
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(16, (3, 3), padding='same',
                 input_shape=x_train.shape[1:],
                 activation='relu',
                 ))
model.add(Conv2D(16, (3, 3), strides=2, activation='relu'))
model.add(Dropout(0.25))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(32, (3, 3), strides=2, activation='relu'))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(66, (3, 3), strides=2, activation='relu'))
model.add(Dropout(0.25))

model.add(Conv2D(10, (3,3), activation='softmax'))

model.add(Flatten())

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

```
print(model.summary())

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

if not data_augmentation:
    print('Not using data augmentation.')
    model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=(x_test, y_test),
              shuffle=True)
else:
    print('Using real-time data augmentation.')
    # This will do preprocessing and realtime data augmentation:
    datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
        width_shift_range=0.1,  # randomly shift images horizontally (fraction of total
width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images

    # Compute quantities required for feature-wise normalization
    # (std, mean, and principal components if ZCA whitening is applied).
    datagen.fit(x_train)
```

```

# Fit the model on the batches generated by datagen.flow().steps_per_epoch
train_history = model.fit_generator(datagen.flow(x_train, y_train,
                                                batch_size=batch_size),
                                   steps_per_epoch=128,
                                   epochs=epochs,
                                   validation_data=(x_test, y_test),
                                   workers=4)

# Save model and weights
if not os.path.isdir(save_dir):
    os.makedirs(save_dir)
model_path = os.path.join(save_dir, model_name)
model.save(model_path)
print('Saved trained model at %s ' % model_path)

# Score trained model.
scores = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])

加载训练完成的模型：

from keras.models import load_model
from keras.datasets import mnist

# 本机路径
(x_train,y_train),(x_test,y_test)= mnist.load_data('/home/lhadmin/imageclassify/app/mnist.npz')
model=load_model('/home/lhadmin/imageclassify/app/saved_models/keras_mnist_fcn_trained_
model_code.h5')
训练过程可视化：
import matplotlib.pyplot as plt
def show_train_history(train_history,train,validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train History')

```

```
plt.ylabel(train)
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
show_train_history(train_history, 'acc', 'val_acc')
show_train_history(train_history, 'loss', 'val_loss')
```

预测结果可视化：

```
import matplotlib.pyplot as plt
def plot_images_labels_prediction(images, labels, prediction,
                                   idx, num=10):
```

```
    fig = plt.gcf()
    fig.set_size_inches(12, 14)
    if num > 25: num = 25
    for i in range(0, num):
        ax = plt.subplot(5, 5, 1+i)
        ax.imshow(images[idx], cmap='binary')
        title = "label=" + str(labels[idx])
        if len(prediction) > 0:
            title += ", predict=" + str(prediction[idx])

        ax.set_title(title, fontsize=10)
        ax.set_xticks([]); ax.set_yticks([])
        idx += 1
    plt.show()
```

```
prediction = model.predict_classes(x_test.reshape(x_test.shape[0], 28, 28, 1))
plot_images_labels_prediction(x_test, y_test, prediction, idx=0)
```

查看混淆矩阵：

```
import pandas as pd
```

```
pd.crosstab(y_test, prediction, rownames=['label'], colnames=['predict'])
```

## 参考文献

- [1] Forsyth D A, Ponce J. Computer vision: a modern approach[M]. Prentice Hall Professional Technical Reference, 2002.
- [2] 关雪梅. 图像特征提取技术研究[J]. 绥化学院学报, 2017, 37(2): 158-160.
- [3] 邵泽明. 计算机视觉伺服跟踪控制系统[D]. 南京: 南京航空航天大学机械工程系, 2003.
- [4] 蔡芷茵, 高炜, 俞祝良, 等. 基于三元组卷积神经网络的图像检索[J]. 西安邮电大学学报, 2016, 21(6): 60-64.
- [5] 梁晔, 刘宏哲. 基于视觉注意力机制的图像检索研究[J]. 北京联合大学学报(自然科学版), 2010, 24(1): 30-35.
- [6] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.
- [7] Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.
- [8] Szegedy C, Ioffe S, Vanhoucke V, et al. Inception-v4, inception-resnet and the impact of residual connections on learning[C]//AAAI. 2017, 4: 12.
- [9] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 3431-3440.
- [10] 吴正文. 卷积神经网络在图像分类中的应用研究[D]. 成都: 电子科技大学, 2015.
- [11] 康牧. 图像处理中几个关键算法的研究[D]. 西安电子科技大学, 2009.
- [12] 罗婷婷. 两种改进的彩色图像灰度化算法研究[D]. 浙江工商大学, 2014.
- [13] 周艳, 艾斯, 卡尔. 基于四边边缘匹配的图像信息隐秘法[J]. 电脑与信息技术, 2007, 15(1): 39-42.
- [14] Lawrence S, Giles C L, Tsoi A C, et al. Face recognition: A convolutional neural-network approach[J]. IEEE transactions on neural networks, 1997, 8(1): 98-113.
- [15] 阳哲. 卷积神经网络在印章编号识别中的应用[J]. 现代计算机: 上下旬, 2016 (3): 47-50.
- [16] Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.
- [17] Specht D F. Probabilistic neural networks[J]. Neural networks, 1990, 3(1): 109-118.
- [18] Cybenko G. Approximation by superpositions of a sigmoidal function[J]. Mathematics of control, signals and systems, 1989, 2(4): 303-314.



- [19] Kalman B L, Kwasny S C. Why tanh: choosing a sigmoidal function[C]//Neural Networks, 1992. IJCNN. , International Joint Conference on. IEEE, 1992, 4: 578-581.
- [20] Agostinelli F, Hoffman M, Sadowski P, et al. Learning activation functions to improve deep neural networks[J]. arXiv preprint arXiv:1412.6830, 2014.
- [21] Dai J, Li Y, He K, et al. R-fcn: Object detection via region-based fully convolutional networks[C]//Advances in neural information processing systems. 2016: 379-387.
- [22] Tutorial C C, Long J, Shelhamer E. Fully Convolutional Networks[J].
- [23] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation[C]//International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015: 234-241.
- [24] LeCun Y, Bengio Y. Convolutional networks for images, speech, and time series[J]. The handbook of brain theory and neural networks, 1995, 3361(10): 1995.
- [25] Huang G, Liu Z, Weinberger K Q, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, 1(2): 3.
- [26] Abdel-Hamid O, Mohamed A, Jiang H, et al. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition[C]//Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. IEEE, 2012: 4277-4280.

## 外文文献

# FULLY CONVOLUTIONAL MULTI-CLASS MULTIPLE INSTANCE LEARNING

## Abstract

Multiple instance learning (MIL) can reduce the need for costly annotation in tasks such as semantic segmentation by weakening the required degree of supervision. We propose a novel MIL formulation of multi-class semantic segmentation learning by a fully convolutional network. In this setting, we seek to learn a semantic segmentation model from just weak image-level labels. The model is trained end-to-end to jointly optimize the representation while disambiguating the pixel-image label assignment. Fully convolutional training accepts inputs of any size, does not need object proposal pre-processing, and offers a pixelwise loss map for selecting latent instances. Our multi-class MIL loss exploits the further supervision given by images with multiple labels. We evaluate this approach through preliminary experiments on the PASCAL VOC segmentation challenge.

## 1. Introduction

Convolutional networks (convnets) are achieving state-of-the-art performance on many computer vision tasks but require costly supervision. Following the ILSVRC12-winning image classifier of Krizhevsky et al. (2012), progress on detection (Girshick et al., 2014) and segmentation (Long et al., 2014) demonstrates that convnets can likewise address local tasks with structured output.

Most deep learning methods for these tasks rely on strongly annotated data that is highly timeconsuming to collect. Learning from weak supervision, though hard, would sidestep the annotation cost to scale up learning to available image-level labels.

In this work, we propose a novel framework for multiple instance learning (MIL) with a fully convolutional network (FCN). The task is to learn pixel-level semantic segmentation from weak imagelevel labels that only signal the presence or absence of an object. Images that are not centered on the labeled object or contain multiple objects make the problem more difficult. The insight of this work is to drive the joint learning of the convnet representation and pixel classifier by multiple instance

learning. Fully convolutional training learns the model end-to-end at each pixel. To learn the segmentation model from image labels, we cast each image as a bag of pixel-level-instances and define a pixelwise, multi-class adaptation of MIL for the loss.

MIL can reduce the need for bounding box annotations (Cinbis et al., 2014; Song et al., 2014), but it is rarely attempted for segmentation. Oquab et al. (2014) improve image classification by inferring latent object location, but do not evaluate the localization. Hoffman et al. (2014) train by MIL fine-tuning but rely on bounding box supervision and proposals for representation learning. Most MIL problems are framed as max-margin learning (Andrews et al., 2002; Felzenszwalb et al., 2010), while other approaches use boosting (Ali & Saenko, 2014) or Noisy-OR models (Heckerman, 2013). These approaches are limited by (1) fixed representations and (2) sensitivity to initial hypotheses of the latent instance-level labels. We aim to counter both shortcomings by simultaneously learning the representation to maximize the most confident inferred instances. We incorporate multi-class annotations by making multi-class inferences for each image. When an image / bag contains multiple classes the competition of pixelwise models help to better infer the latent instance-level classes.

We investigate the following ideas and carry out preliminary experiments to these ends:

- We perform MIL jointly with end-to-end representation learning in a fully convolutional network. This eliminates the need to instantiate instance-label hypotheses. FCN learning and inference can process images of different sizes without warping or object proposal pre-processing. This makes training simple and fast.
- We propose a multi-class pixel-level loss inspired by the binary MIL scenario. This tries to maximize the classification score based on each pixel-instance, while simultaneously taking advantage of inter-class competition in narrowing down the instance hypotheses.
- We target the under-studied problem of weakly supervised image segmentation. Our belief is that pixel-level consistency cues are helpful in disambiguating object presence. In this way weak segmentation can incorporate more image structure than bounding boxes.

## 2. Fully Convolutional MIL

A fully convolutional network (FCN) is a model designed for spatial prediction problems. Every layer in an FCN computes a local operation on relative spatial coordinates. In this way, an FCN can take an input of any size and produce an output of corresponding dimensions.

For weakly supervised MIL learning, the FCN allows for the efficient selection of training instances. The FCN predicts an output map for all pixels, and has a corresponding loss map for all pixels. This loss map can be masked, re-weighted, or otherwise manipulated to choose and select instances for computing the loss and back-propagation for learning.

We use the VGG 16-layer net (Simonyan & Zisserman, 2014) and cast it into fully convolutional form as suggested in Long et al. (2014) for semantic segmentation by replacing fully connected layers with corresponding convolutions. The network is fine-tuned from the pre-trained ILSVRC classifier weights i.e. pre-trained to predict image-level labels. We then experiment with and without initializing the last layer weights i.e. the classifier layer. These initializations, without MIL finetuning, act as the baselines (row 1 and 2 in Table). If there is no image-level pretraining, the model quickly converges to all background. Semantic segmentation requires a background class but the classification task has none; we simply zero initialize the background classifier weights.

### 3. MULTI-CLASS MIL LOSS

We define a multi-class MIL loss as the multi-class logistic loss computed at maximum predictions.

This selection is enabled by the output map produced by FCN i.e. for an image of any size, the FCN outputs a heat-map for each class (including background) of corresponding size. We identify the max scoring pixel in the coarse heat-maps of classes present in image and background. The loss is then only computed on these coarse points, and is back propagated through the network. The alternating optimization in the binary MIL problem inspires this ignoring of the loss at nonmaximally scoring points. The background class is analogous to the negative instances by competing against the positive object classes. Let the input image be  $I$ , its label set be  $\mathcal{L}_I$  (including background label) and  $\hat{p}_l(x,y)$  be the output heat-map for the  $l$ th label at location  $(x,y)$ . The loss is defined as:

$$(x_l, y_l) = \arg \max_{\forall (x,y)} \hat{p}_l(x, y) \quad \forall l \in \mathcal{L}_I$$

$$= \frac{-1}{|\mathcal{L}_I|} \sum_{l \in \mathcal{L}_I} \log \hat{p}_l(x_l, y_l)$$

Ignoring the loss at all non-maximally scoring points is key to avoid biasing the learning of the FCN to the background. Simultaneous training exploits multi-label images through inter-class confusion to help refine the intra-class pixel accuracy. At inference time, the MIL-FCN takes the top class prediction at every point in the coarse prediction and bilinearly interpolates to image resolution to obtain a pixelwise segmentation.

#### 4. EXPERIMENTS

All results are on the PASCAL VOC segmentation challenge. We train and validate on the VOC

2011 train augmented by Hariharan et al. (2011) and val sets then evaluate on the completely heldout VOC 2012 test set. The evaluation metric is intersection over union (IU), and is defined per class as the percentage of pixels in the intersection of ground truth segmentation mask, and the predicted mask out of the number of pixels in their union. The MIL-FCN model is initialized from the 16-layer VGG ILSVRC14 classifier (Simonyan & Zisserman, 2014) then fine-tuned by the MIL loss. Long et al. (2014) fine-tune from all but the output layer, as they have access to complete supervision. In our setting however, transferring the output layer parameters for the classes common to both PASCAL and ILSVRC improves results. Including these classifier parameters helps prevent degenerate solutions of predicting all background. We train our model with a learning rate 0.0001, momentum 0.9 and weight decay 0.0005. The training is quick and the network converges in less than 10,000 iterations.

Table 1: Results on PASCAL VOC 2011 segmentation validation and 2012 test data. Fine-tuning with the MIL loss achieves 96% relative improvement over the baseline.

Approach	mean IU (VOC2011 val)	mean IU (VOC2012 test)
Baseline (no classifier)	3.52%	Baseline (with classifier) 13.11%
MIL-FCN	25.05%	25.66%
Oracle (supervised)	59.43%	63.80%

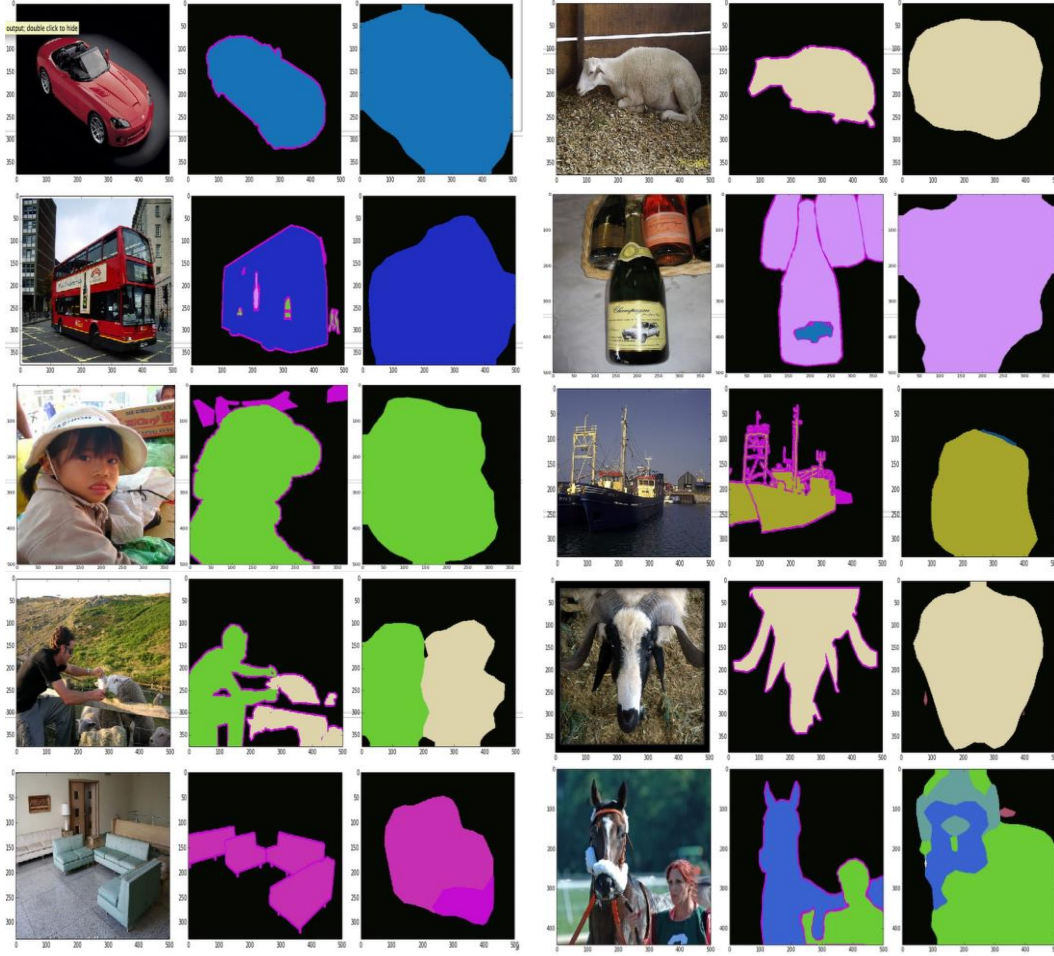


Figure 1: Sample images from PASCAL VOC 2011 val-segmentation data. Each row shows input (left), ground truth (center) and MIL-FCN output (right).

Table 1 shows quantitative intersection-over-union (IU) scores while example outputs from MILFCN are shown in Figure 1. MIL-FCN achieves 96% relative improvement over the baseline results when the classifier is fine-tuned from the common classes. These are preliminary but encouraging results.

## 5. DISCUSSION

We propose a novel model of joint multiple instance and representation learning with a multi-class pixelwise loss inspired by binary MIL. This model is learned end-to-end as a fully convolutional network for the task of weakly supervised semantic segmentation. It precludes the need for any kind of proposal or instance hypothesis mechanisms. Inference is fast ( $\approx 1/5$  sec).

These results are encouraging, and can be improved further. Currently, the coarse output is merely interpolated; conditional random field regularization or super-pixel (Achanta et al., 2012) projection could refine the predictions. These grouping methods

could likewise drive learning by selecting whole segments instead of single points for MIL training. Moreover, controlling convnet learning by manipulating the loss map could have further uses such as encouraging consistency across images for co-segmentation or hard negative mining.

## 全卷积神经网络多分类多实例学习

**摘要：**多重实例学习（MIL）可减少对诸如语义分割等任务的注释的需求，从而削弱所需的监督程度。我们提出了一种全新的卷积网络的多类语义分割学习的 MIL 公式。在这种情况下，我们试图从弱图像级标签中学习语义分割模型。该模型是端对端训练以共同优化表示，同时消除像素图像标签分配的歧义。完全卷积训练接受任意大小的输入，不需要对象提议预处理，并提供用于选择潜在实例的按像素丢失图。我们的多级 MIL 损失利用了多标签图像的进一步监督。我们通过 PASCAL VOC 分割挑战的初步实验来评估此方法。

### 1. 介绍

卷积网络（convnets）在许多计算机视觉任务上实现了最先进的性能，但需要昂贵的监督。继 Krizhevsky 等人的 ILSVRC12 获奖图像分类器之后，（2012 年），检测进展（Girshick 等，2014）和细分（Long 等，2014）表明，convnets 同样可以用结构化输出来处理本地任务。

这些任务的大多数深度学习方法都依赖于强大的注释数据，这些数据非常耗时收集。从弱监管中学习虽然很难，但会避开注释成本，以扩大对可用图像级标签的学习。在这项工作中，我们提出了一个全卷积网络（FCN）的多实例学习（MIL）的新框架。其任务是从弱图像级标签中学习像素级语义分割，这些标签只表示对象是否存在。不在标注对象上的图像或包含多个对象的图像使问题更加困难。这项工作的见解是通过多实例学习来推动闭合表示和像素分类器的联合学习。完全卷积训练在每个像素端到端学习模型。为了从图像标签中学习分割模型，我们将每个图像作为一包像素级实例进行投射，并定义一个像素级的多级适应 MIL 的损失。

MIL 可以降低对边界框注释的需求（Cinbis 等，2014；Song 等，2014），但很少尝试分割。Oquab 等人（2014）通过推断潜在物体的位置来改善图像分类，但不评估本地化。霍夫曼等人（2014 年）通过 MIL 微调训练，但依靠包围盒监督和代表性学习提案。大多数 MIL 问题都被认为是最大利润率学习（Andrews et al.，2002；Felzenszwalb et al.，2010），而其他方法使用提升（Ali&Saenko，2014）或 Noisy-OR 模型（Heckerman，2013）。这些方法受限于（1）固定表示和（2）对潜在实例级别标签的初始假设的敏感性。我们旨在通过同时学习表示来最大限度地利用最自信的推断实例来克服两个缺点。我们通过对每个图像进行多级推理来整合多级注释。当图像/包包含多个类时，像素模型的竞争有助于更好地推断潜在的实例级类。



我们调查以下想法并为此进行初步实验：

- 我们在完全卷积网络中与端到端表示学习一起执行 MIL。这消除了实例化实例标签假设的需要。FCN 学习和推理可以处理不同大小的图像，而无需翘曲或对象提议预处理。这使得训练简单快捷。
- 我们提出了受二进制 MIL 情景启发的多级像素级损失。这试图最大化基于每个像素实例的分类分数，同时利用阶级间的竞争来缩小实例假设。
- 我们针对弱监督图像分割的研究不足问题。我们的信念是，像素级一致性线索有助于消除对象存在。通过这种方式，弱分割可以包含比边框更多的图像结构。

## 2. 全卷积 MIL

完全卷积网络（FCN）是为空间预测问题设计的模型。FCN 中的每一层计算相对空间坐标上的本地操作。通过这种方式，FCN 可以接受任何尺寸的输入并产生相应尺寸的输出。

对于弱监督的 MIL 学习，FCN 允许有效地选择训练实例。FCN 预测所有像素的输出映射，并且具有针对所有像素的相应损失映射。可以对该损失图进行掩蔽，重新加权或以其他方式操纵以选择和选择用于计算学习的损失和反向传播的实例。

我们使用 VGG 16 层网络（Simonyan & Zisserman, 2014），并将其转换为完全卷积形式，如 Long 等人所建议的。（2014）通过用相应的卷积替换完全连接的层来进行语义分割。该网络根据预先训练的 ILSVRC 分类器权重进行微调，即预先训练以预测图像级别标签。然后，我们尝试并且不初始化最后的层权重，即分类器层。这些初始化没有 MIL 微调，充当基线（表 1 中的第 1 行和第 2 行）。如果没有图像级预训练，模型会快速收敛到所有背景。语义分割需要背景类，但分类任务没有；我们只是初始化背景分类器权重。

## 3. 多分类 MIL 损失

我们将多级 MIL 损失定义为在最大预测下计算的多级逻辑损失。该选择由 FCN 产生的输出图启用，即对于任何尺寸的图像，FCN 为相应尺寸的每个类（包括背景）输出热图。我们在图像和背景中存在的类的粗热图中确定最大得分像素。然后只计算这些粗点的损失，并通过网络反向传播。二元 MIL 问题中的交替优化激发了这种忽略非最大得分点的损失。背景类通过与积极的对象类竞争而类似于负面的例子。设输入图像为  $I$ ，其标签集为  $LI$ （包括背景标签）， $p_l(x, y)$  为位置  $(x, y)$  处第  $l$  个标签的输出热图。损失定义为：

$$(x_l, y_l) = \arg \max_{\forall (x, y)} \hat{p}_l(x, y) \quad \forall l \in \mathcal{L}_I$$

$$= \frac{-1}{|\mathcal{L}_I|} \sum_{l \in \mathcal{L}_I} \log \hat{p}_l(x_l, y_l)$$

忽略所有非最大得分点的损失是避免将 FCN 的学习偏向于背景的关键。同步训练通过类间混淆利用多标签图像来帮助改进类内像素精度。在推断时间，MIL-FCN 在粗略预测的每个点处进行顶级预测，并对图像分辨率进行双线性插值以获得像素分割。

#### 4. 实验

所有结果都基于 PASCAL VOC 分割挑战。我们对 VOC 进行培训和验证。2011 年列车由 Hariharan 等人加强。（2011）和 val 设置，然后评估完全保留的 VOC 2012 测试集。评估度量是联合交集（IU），并且每个类定义为地面真实分段蒙版交集中像素的百分比，以及联合中像素数量中的预测蒙版。MIL-FCN 模型从 16 层 VGG ILSVRC14 分类器（Simonyan & Zisserman, 2014）初始化，然后通过 MIL 损失进行微调。Long 等人（2014 年）从除输出层以外的所有层面进行微调，因为他们可以获得完整的监督。然而，在我们的设置中，为 PASCAL 和 ILSVRC 共同的类传输输出层参数可以改善结果。包括这些分类器参数有助于防止预测所有背景的退化解决方案。我们训练模型的学习率为 0.0001，动量为 0.9，体重衰减为 0.0005。训练速度快，网络收敛少于 10,000 次迭代。

表 1 关于 PASCAL VOC 2011 分割验证和 2012 测试数据的结果。使用 MIL 损耗进行微调可以比基准线提高 96%

Approach	mean IU (VOC2011 val)	mean IU (VOC2012 test)
Baseline (no classifier)	3.52%	Baseline (with classifier) 13.11% 13.09%
MIL-FCN	25.05%	25.66%
Oracle (supervised)	59.43%	63.80%

图 1 来自 PASCAL VOC 2011 val-segmentation 数据的样本图像。每行显示输入（左），地面实况（中心）和 MIL-FCN 输出（右）

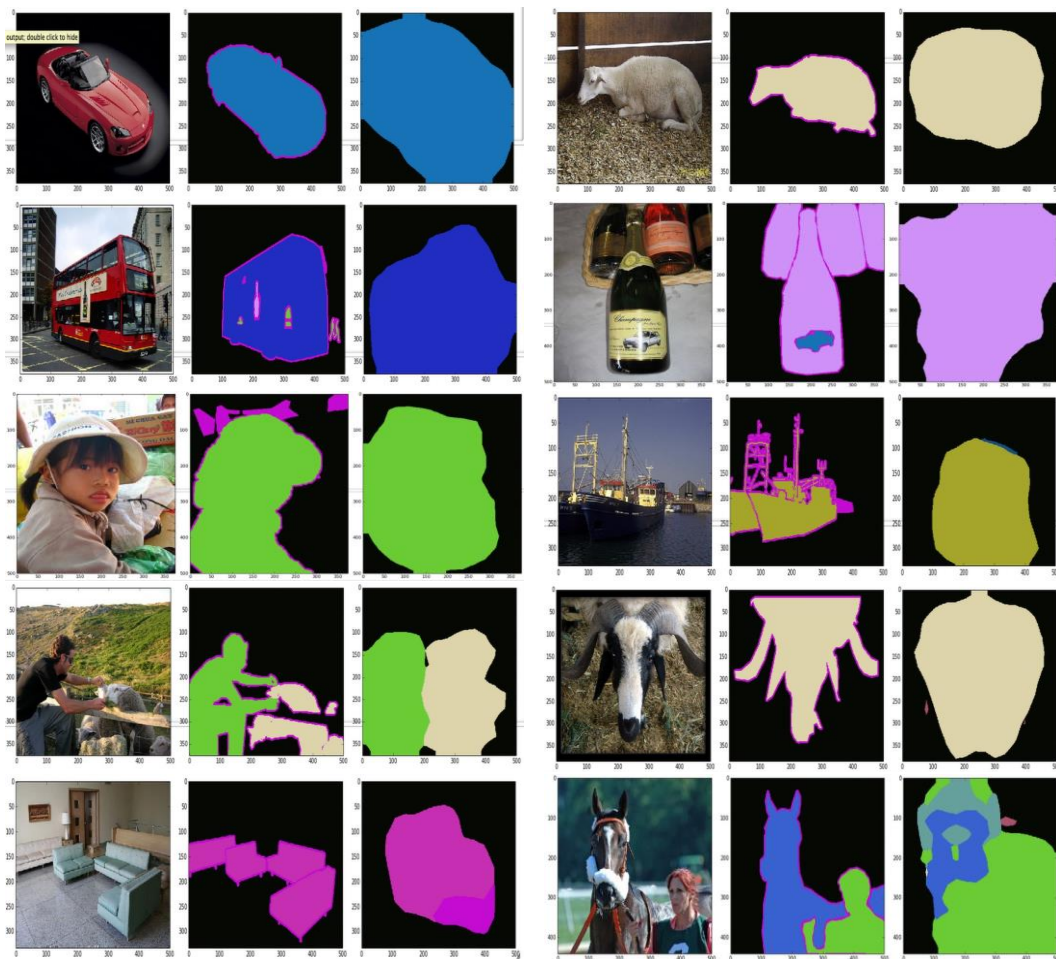


表 1 显示了定量的交叉联盟（IU）得分，而来自 MILFCN 的示例输出显示在图 1 中。当分类器从常用类别进行微调时，MIL-FCN 相对于基线结果实现 96% 的相对改进。这些都是初步但令人鼓舞的结果

## 5. 讨论

我们提出了一种新颖的联合多实例和表示学习模型，其具有受二进制 MIL 启发的多级像素损失。该模型作为完全卷积网络端到端地学习，用于弱监督语义分割的任务。它排除了任何形式的提案或实例假设机制的需要。推论很快（ $\approx 1/5$  秒）。

这些结果令人鼓舞，并且可以进一步改善。目前，粗略输出仅仅是内插的；条件随机场正则化或超像素（Achanta 等，2012）预测可以改进预测。这些分组方法同样可以通过为 MIL 训练选择整个分段而不是单个分数来推动学习。此外，通过操纵损失地图来控制小环境学习可能会有其他用途，例如鼓励图像之间的一致性分析或硬性负面挖掘的一致性。

## 致 谢

光阴荏苒，岁月如梭，转眼之间毕业设计就已到尾声，同时大学四年的生活也即将结束四年的大学生活让我受益良多。回想这几个月的设计历程，期间我得到了许多的关怀和帮助，现在要向他们表示我最诚挚的谢意。

首先向我的指导老师喻玲娟老师致以最诚挚的谢意，感谢老师在繁忙的工作中挤出时间审查，仔细审查论文，并提出了许多宝贵的意见，老师严谨的治学态度影响着我，让我获益匪浅。在整个设计期间，老师给了我很大的帮助，每一项任务都凝聚着老师辛勤与汗水，在此向老师表达我最诚挚的敬意。

在其次我要感谢父母的养育之恩，论文的完成离不开他们的大力支持，在这里道一声爸爸妈妈您们辛苦了。最后，要感谢江西理工大学，我的母校，给我提供了如此难得的学习机会和优越的学习条件。

# 基于全卷积神经网络的图像分类

姚美君<sup>1a</sup>

(1. 江西理工大学, 信息工程学院, 江西 赣州 341000)

**摘要:** 卷积神经网络(CNN)广泛应用于图像分类, CNN 由交替的卷积层和池化层, 以及全连接层构成。本文研究仅有卷积层组成的新架构, 即全卷积神经网络, 它是对 CNN 结构的一种改进, 将其中的池化层和全连接层全部用卷积层替代。进一步地, 本文针对公开的数据集 mnist 和 cifar-10, 采用全卷积神经网络对数据集进行分类, 在时间效率和精确度两个方面, 与 CNN 的实验结果进行对比。

**关键词:** CNN 卷积神经网络; FCN 全卷积神经网络; mnist; cifar-10

**Abstract:** Convolutional neural networks (CNN) are widely used for image classification. CNN consists of alternating convolutional layers and pooled layers, as well as fully connected layers. In this paper, we study the new architecture consisting of only convolutional layers, that is, full convolutional neural networks. It is an improvement to the CNN structure, in which the pooling layer and the fully connected layer are all replaced by convolutional layers. Further, for the disclosed datasets mnist and cifar-10, the data set is classified using a full convolutional neural network, and compared with the experimental results of CNN in terms of time efficiency and accuracy.

**Key words:** CNN convolutional neural network; FCN full convolutional neural network; mnist; cifar-10

## 0 前言

现代的大多数情景中, 几乎都是使用卷积神经网络 CNN 用于图像分类, CNN: 由交替的卷积层和池化层, 网络末尾少量的全连接层构成。全卷积神经网络 (All Convolutional Network, A-CNN) 是对 CNN 结构的一种改进, 将其中的池化层和全连接层全部用卷积层替代。针对公开的数据集 mnist 和 cifar-10, 提出基于 FCN 的图片分类方法, 并与 CNN 的实验结果进行对比。

## 1 卷积神经网络基本原理

### 1.1 卷积层

卷积层在数据上的意义即相乘和相加<sup>[1]</sup>。卷积运算的目的是检测图像边缘, 从中提取不同的特征, 第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级, 更多层的网路能从低级特征中迭代提取更复杂的特征<sup>[2]</sup>。

卷积层可训练的参数: ①kernel\_size: 卷积核尺寸大小, 卷积核长宽越小, 则捕获的细节就更精细。②filters 卷积核的数目:

filter 越多,检测到的边缘角度就多,训练效果就更好。但是时间成本就会增加。③strides: 卷积的步长,在图像像素矩阵上移动几个像素。卷积的步长越大,图像的尺寸缩减的越快, ④padding: 阵列补 0 ⑤Activation function: 激活函数,可以使卷积神经网络对于非线性的数据也有比较好的效果。⑥bias: 偏置项。

假设  $n$  为像素矩阵的长宽,  $p$  为 padding 的维度,  $f$  为 filter 的长宽,  $s$  为步长

## 1.2 池化层

池化层可以减少网络中的参数和计算量。以最常用的最大池化为例,每次选取区域中最大的值,提取出来放在新的矩阵。如果这块区域中有明显特征(矩阵像素值大),则会被提取出来。若区域中没有明显特征,则各个值都很小,提取出来的值也是偏小的。但是,池化的过程,特征矩阵缩小的太快,容易丢失大量的信息。

池化层中的可选参数有(1)池化类型: 最大化池化、平均池化(2) pool\_size: 池化窗口大小,池化过滤器的矩阵尺寸。(3) strides: 池化步长。例如设 2 将会使得输出 shape 为输入的一半。(4) padding: 补零还是不补零。池化层可以减小运算量,提高计算速度,同时提高所提取特征的鲁棒性,也可以防止过拟合。

## 1.3 全连接层

全连接层实际上是多层感知器应用于卷积神经网络末端之后别名。感知器是一种线性分类器,通过用一条直线将输入分成两类。输入的特征向量  $x$  与权重  $w$  相乘再与偏差  $b$  相加。 $Y = w * x + b$ 。感知器通过使用权重矩阵  $w$  形成线性组合(也可以对感知

器的输出再使用非线性激活函数),可以基于多个输入产生单个输出,现在的全连接层一般和激活函数配合使用,可以产生非线性的决策边界,处理更复杂的非线性分类问题。

多层感知器意味着模型中有多个感知器组成。多层感知器是深度的人工神经网络。它由一个接受信号的输入层,一个产生决策函数或是预测的输出层,两者之间任意数量的隐藏层。在使用最经典的多层感知器进行分类时,首先初始化权重矩阵  $w$ ,定义损失函数  $J$ ,比如平方损失函数或者交叉熵。模型训练的过程就是最小化损失函数的过程,一般来说,损失函数必须是凸函数,才有最优解。在多层感知器上不断迭代(普遍的是小批量梯度下降),更新  $w$  和  $b$ ,找到使  $J$  最小化的  $w$  和  $b$ 。全连接层最大的特点是,每一层都和上一层的神经元完全连接。

## 1.4 激活函数

一般来说,对于卷积层和全连接层的输出使用激活函数。Activation function 通常为非线性函数。在网络中加入了激活函数,让神经网络可以处理比较复杂的非线性问题<sup>[3]</sup>。

常见的激活函数有 sigmoid<sup>[4]</sup>、tanh<sup>[5]</sup>、relu<sup>[6]</sup>、softmax。那么,应该选择哪个激活函数呢?应该根据一定的条件,选择更合适的激活函数加速训练的过程。例如,对于二元分类来说,sigmoid 可以很好地工作,因为 sigmoid 的输出在 0 和 1 之间,在大多数情况下,tanh 都可以替代 sigmoid 函数,除了二元分类输出层的激活函数。在大部分情况下或不知道选择什么激活函数时,使用 ReLu 函数,在很多问题上都可以表现的很好。在多类型神经网络中的最后一层,通

常使用 softmax 激活函数。

FCN 中，会用卷积层替代池化层和全连接层<sup>[7]</sup>

2 全卷积神经网络的基本原理

2.1 全卷积神经网络架构

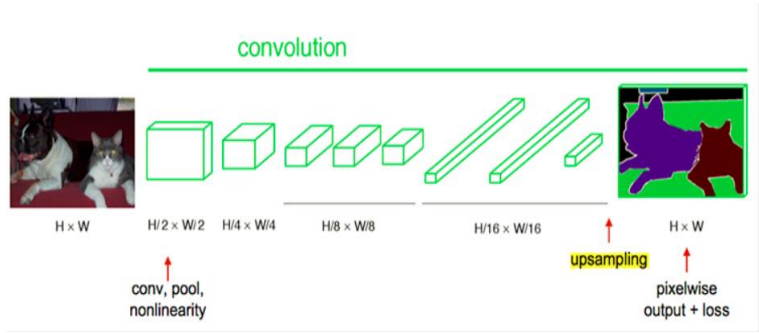


图 3 全卷积神经网络结构

2.2 卷积层替换池化层

池化层在于 CNN 中的主要作用，一个适用于降维，减少训练的参数，避免过拟合，另一个是提取子区域中的主要特征（比如最大池化）。假设  $10 \times 10$  的二维矩阵，过滤器尺寸为  $3 \times 3$ 。在使用  $2 \times 2$  的 filter 进行步长为 2 的池化时，输出为  $5 \times 5$  的特征矩阵。本文用步长为 2 的卷积层替代时，卷积步长为 2，那么卷积的输出结果是  $((10-3)/2) + 1 = 4$ ，输出维度为  $4 \times 4$  的特征图。相比于  $10 \times 10$  的像素矩阵，步长为 2 的卷积层也做到了降维。并且减少了要训练的参数。

2.3 卷积层替代全连接层

全连接层即下一层的每一个神经元都与上一层的全部连接<sup>[8]</sup>，本节定义输入为  $X$ ， $x$  和隐藏层之间有一个权重矩阵，可以把每一个神经元看作一个特征，那么每个特征对最终结果的影响不一样，所以需要有一个权重

矩阵<sup>[9]</sup>，来定义不同特征对于下一层的影响的权重。全连接层的输出结果为： $z = \text{activation\_function}(w'x + b)$  其中， $w'$  为权重矩阵的转置。全连接层为了让输入层、隐藏层和输出层之间完全连接，它必须要接受固定长度的输入对应神经元的数量。但是卷积层，是在一个大矩阵上依次滑动，所以塔可以接受任何大小的输入。考虑卷积的过程： $\text{activation\_function}(n * f + b)$ 。可以发现，卷积和全连接的过程都是相乘再相加的过程。

3 基于全卷积神经网络的分类

3.1 mnist 和 cifar-10 数据集介绍

对比 mnist 手写数字集合 cifar-10，见表 1。可以看到 cifar-10 的图像尺寸大小远远大于 mnist，并且 mnist 是灰度图像，RGB 是红绿蓝三色颜色通道的叠加。

表 1 Mnist 和 cifar 图像集对比

	Mnist	Cifar-10
Size	11.5MB	175.5MB
颜色格式	灰度图像	RGB 图像
数据内容	手写数字图像	飞机、车等十类图像



### (1) mnist 图像集

mnist 数据集，前十笔数据展示，上标是关于此图片对应的标记，即真实值。将 mnist 数据集 60000 笔数据按比例划分，训练集样本 50000 笔，测试集样本 10000 笔，见表 2。图片的维度为  $28 * 28 * 1$  的灰度图。

### (2) cifar-10 图像集

为了简化代码和表达，将 cifar-10 十种类型分别对应到 0~10 个数字。展示 cifar-10 前十笔数据，根据上标可以知道该图片的标记值及对应编号。同理，将 cifar-10 的 60000 笔数据按比例分类，得到训练集和测试集。cifar-10 数据集的图片大小为  $32 * 32 * 3$  的 RGB 彩色图。

## 3.2 mnist 数据集处理结果

CNN 的模型由图 2（左）所示，由三个交替的卷积层和池化层构成，最后连接全连接层，128 个的神经元组成的输入层和隐藏层，10 个的神经元的输出层，并通过 softmax 激活函数。FCN 即在 CNN 的基础上，将步长为 2 的池化层由步长为 2 的卷积层替代，全连接层由一层 10 个过滤器的卷积层（激活函数为 softmax）替代。

可视化模型的训练过程，图 5 可视化 CNN 模型的精确度随训练批次变化过程，可以发现精确度在 epoch<4 之前，精确度提升很快，之后只有小幅的提高，在 epoch>8 的区间，精确度几乎并无变化。从而，在目前的条件下，可以知道，单纯的继续增加训练批次，对模型并不能在继续产生较大影响。根据精确度和损失函数的随训练批次的变化曲线，可以在调整模型训练过程中的一些参数，根据不同的参数判断模型如何才能表

现的更好。是需要更多次迭代，计算出使损失函数更小更优的权重矩阵  $W$  和偏差  $b$ ，还是增加大量训练批次也无法获得精确度的较大提升，是否需要放弃通过批次提升精确度的方法。

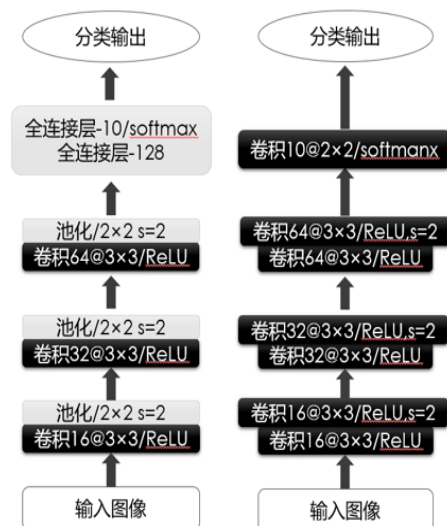


图 4 mnist CNN（左）和 FCN 模型（右）

除此之外，通过对比训练集精确度和测试集精确度，可以判断模型是否出现了过拟合。因为模型根据训练集的数据计算而来，一般来说训练集精确度会高于测试集。但是评估一个模型的好坏关键在于，它对于没有见过的新实例，有如何的表现，也称为模型的泛化能力。若模型对于没有见过的数据，也能做出更好更准确的判断，则称作优秀的模型。根据图 6 发现测试集精确度反而高于训练集，说明模型没有产生过拟合。图 7 和图 8，同样，可以看出精确度和损失函数随 epoch 的增加的变化情况。且根据训练集和测试集的精确度对比，可以发现模型存在轻微过拟合。

根据判断模型是否产生了过拟合，可以增加一些措施，例如使用更适合当前需求的激活函数，丢弃更多比例的参数，增加正则化方法等。



predict	0	1	2	3	4	5	6	7	8	9
label										
0	979	1	0	0	0	0	0	0	0	0
1	0	1130	3	1	0	0	0	1	0	0
2	0	0	1029	0	0	0	0	3	0	0
3	0	0	0	1005	0	4	0	0	1	0
4	0	0	0	0	975	0	3	0	0	4
5	1	0	0	6	0	884	1	0	0	0
6	8	3	0	1	2	1	941	0	2	0
7	2	3	6	0	0	0	0	1013	1	3
8	3	0	1	1	0	1	0	0	965	3
9	2	0	0	2	3	2	0	1	0	999

图 5 mnist 混淆矩阵

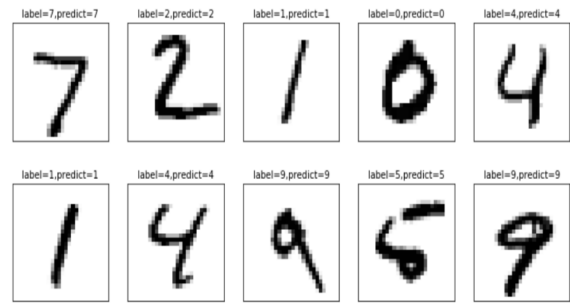


图 6 mnist 预测结果展示

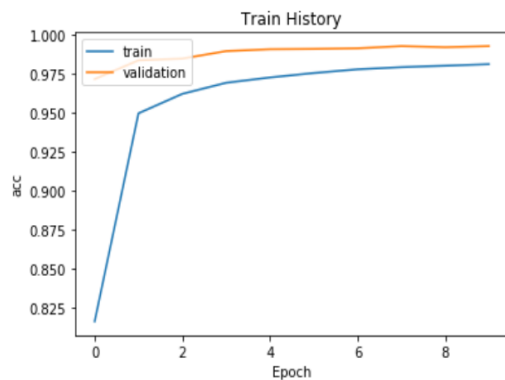


图 5 cnn 精度

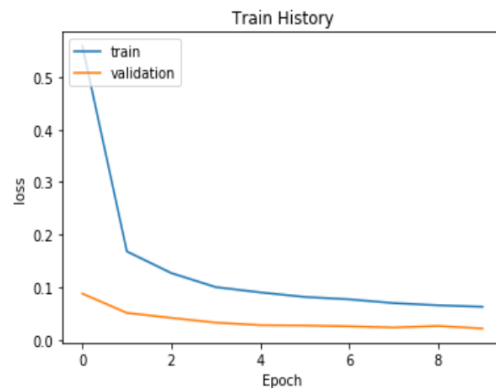


图 6 cnn loss 函数

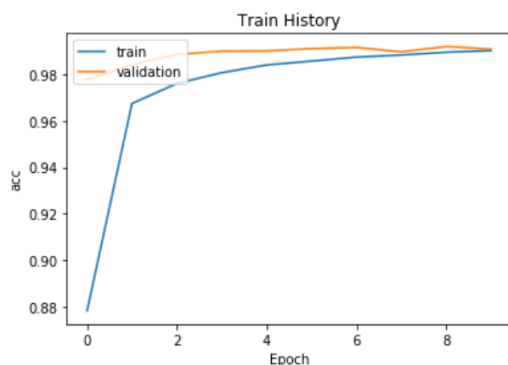


图 7 fcnn 精度

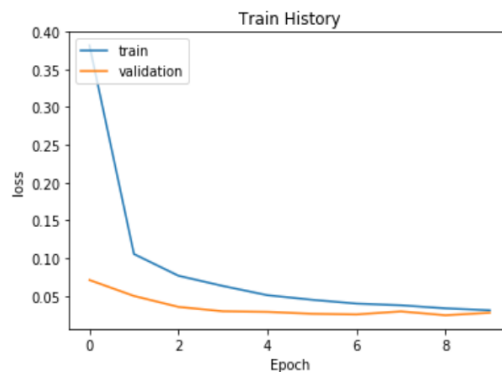


图 8 fcnn loss 函数

### 4.3 cifar-10 数据集处理结果



图 9 cifar CNN (左) 和 FCN 模型 (右)

CNN 和 FCN 训练模型见图 9，将全连接层的神经元增加到了 512 个，因为 cifar-10 是 RGB 彩色图像，且图像包含内容更加丰富，图像包含的特征更多。同样，FCN 模型中用卷积层替代了池化层和全连接层。

cifar-10 数据集前十笔数据预测结果如图 11，上标表明了图像的真实值和预测值，如第一行第一列的图像真实值为 cat 猫，预测值为 truck 卡车，预测结果错误。第一行第

四列的图像真实值标记为 airplane 飞机，预测值也为 airplane 飞机，则预测正确。

表 2 CNN 与 FCN 实验数据

algorithm	val_acc	time	dataset	epochs	params
CNN	0.9911	410s	mnist	10	98442
FCN	0.9925	490s	mnist	10	74362
CNN	0.7512	10000s	cifar	1000	553514
FCN	0.7632	8000s	cifar	1000	79184

预测结果的混淆矩阵见图 10,可以发现飞机容易被预测为鸟和船,摩托车易错误预测为青蛙,狗易错误预测为鸟、猫、蜜蜂等。CNN 和 FCN 在 Cifar-10 数据集训练过程,精确度和损失函数可视化,见下图。

表 2 展示了 CNN 和 FCN 在 mnist 和 cifar-10 数据集上,在时间和精确度以及训练参数的具体数字。

在 mnist 小数据集下,FCN 和 CNN 中的参数数量,没有很大的区别。①在精确度上: FCN 比 CNN 略高一些。②在时间效率上: FCN 参数更少,但是由于没有缺少了池化层的降维,反而运行时更长。并没有体现出参数共享的优势。

在 cifar 这种较大一点的数据集上,CNN 中的参数相比于 FCN 有了一个数量级的增加。此时参数共享的优势体现出来了。①在精确度上: FCN 模型的训练结果在精确度上比

CNN 略高。②在时间效率上: 在梯度下降的批次比较大的时候,FCN 训练模型所需时间比 CNN 有大幅减少。

predict label	0	1	2	3	4	5	6	7	8	9
0	649	55	1	2	2	4	0	6	93	188
1	18	735	1	0	0	3	0	4	17	222
2	173	46	89	27	2	245	3	82	41	292
3	92	59	3	86	3	226	4	68	49	410
4	43	42	10	16	42	170	3	177	48	449
5	36	33	0	26	0	547	1	61	22	274
6	36	102	7	36	7	79	114	53	104	462
7	23	16	2	8	1	107	0	547	17	279
8	103	70	0	1	0	2	0	6	564	254
9	26	59	0	2	0	1	0	8	8	896

图 10 cifar-10 混淆矩阵



图 11 cifar-10 预测结果展示

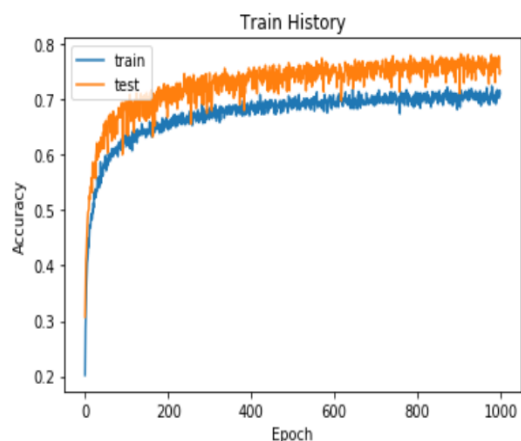


图 12 CNN 精确度

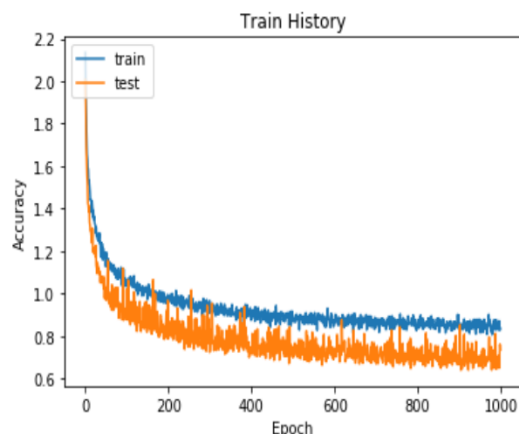


图 13 CNN 损失函数

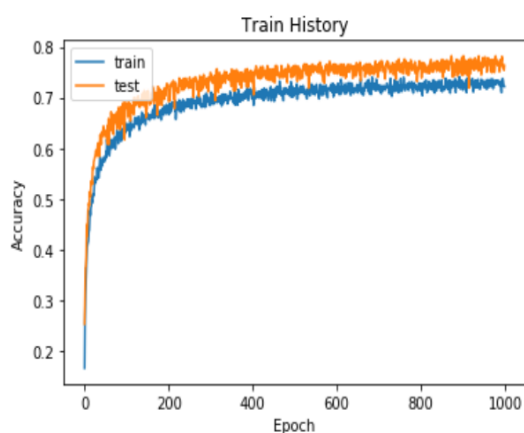


图 14 FCN 精确度

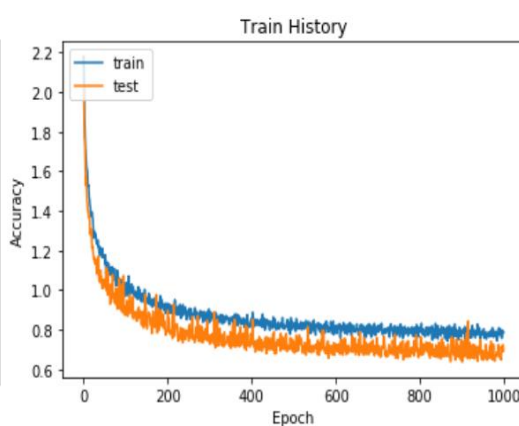


图 15 FCN 损失函数

## 2. FCN 和 CNN 应用场景比较和探讨

在实践中，在使用卷积层替代了池化层和全连接层之后。在较小的数据集上：两者的差异并不大，FCN 的精确度比 CNN 略高一些，时间效率上越低一点。但是在较大数据集上，卷积过程中参数共享这个优势体现了出来。FCN 不仅在训练时间比 CNN 少了许多。

## 4 总结

相比于 CNN，FCN 在以下三方面更棒：

(1) 输入图片尺寸：如果网络中没有全连接层，则可以接受几乎任何尺寸的图像。因为只有全连接层需要特定大小的输入。

(2) 像素空间信息：全连接层通常会造大量的空间信息丢失。因为每一个神经元都与上一层相连，这种架构不能用于分割。

(3) 计算成本：由于卷积层要训练的参数更少。所以计算成本更低。

(4) 精确度：相比于 CNN，FCN 在大数据集上的精确度略高一些。

关于 CNN 和 FCN 图像分类的应用场景总结：FCN 在大数据集上，相比于 CNN 在时间和精确度上都更有优势。在精确度上略高于 CNN。所以，得出以下结论：

(1) 如果数据集比较小：CNN 和 FCN 都能胜任

(2) 若数据集较大，选择 FCN 能让你的模型在精确度是时间效率上都更有优势。

## 参考文献:

- [1] Lawrence S, Giles C L, Tsoi A C, et al. Face recognition: A convolutional neural-network approach[J]. IEEE transactions on neural networks, 1997, 8(1): 98-113.
- [2] 阳哲. 卷积神经网络在印章编号识别中的应用[J]. 现代计算机: 上下旬, 2016 (3): 47-50.
- [3] Specht D F. Probabilistic neural networks[J]. Neural networks, 1990, 3(1): 109-118.
- [4] Cybenko G. Approximation by superpositions of a sigmoidal function[J]. Mathematics of control, signals and systems, 1989, 2(4): 303-314.
- [5] Agostinelli F, Hoffman M, Sadowski P, et al. Learning activation functions to improve deep neural networks[J]. arXiv preprint arXiv:1412.6830, 2014.
- [6] Dai J, Li Y, He K, et al. R-fcn: Object detection via region-based fully convolutional networks[C]//Advances in neural information processing systems. 2016: 379-387.
- [7] Huang G, Liu Z, Weinberger K Q, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, 1(2): 3.
- [8] Abdel-Hamid O, Mohamed A, Jiang H, et al. Applying convolutional neural networks concepts to hybrid NN-HMM model for speech recognition[C]//Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on. IEEE, 2012: 4277-428.