Project Problem 6

小组成员：
王苏然（负责SVM模型部分、可视化）
张子谦（负责CNN模型部分）
杜昊雨（负责VGG模型部分）
陈禹蒙（负责ResNet模型部分，模型之间结果分析）

研究背景

数据集介绍

我们的数据集是包含了经过分割的高通量显微镜图像数据，数据来源于Pärnamaa和Parts（2017）的研究。他们在分割或裁剪之前使用了Chong等人（2015）的数据，该数据存储在CYCLoPs数据库（Koh等人，2015）中。他们从显微镜图像中裁剪出以单个细胞为中心的64x64像素块，因此可能还有其他周围细胞具有相同的荧光模式。细胞的特定荧光模式提供了亚细胞位置的信息。图像中的红色和绿色通道分别标记细胞本体和蛋白质的位置。该数据集包括65,000个训练样本、12,500个验证样本和12,500个测试样本，来自12个亚细胞定位类别（细胞周边、细胞质、内体、内质网、高尔基体、线粒体、核周边、核仁、细胞核、过氧化物酶体、纺锤极和液泡），这些类别的样本数量不平衡。

```
# find ./6-E3_data_processed -type d -exec bash -c 'echo -n "{}: "; find "{}" -type f -iname "*.png" | wc -l' \; | sort -k1

./6-E3_data_processed/test_img/0:     1569
./6-E3_data_processed/test_img/1:     1276
./6-E3_data_processed/test_img/2:      689
./6-E3_data_processed/test_img/3:     1755
./6-E3_data_processed/test_img/4:      382
./6-E3_data_processed/test_img/5:     1243
./6-E3_data_processed/test_img/6:     1164
./6-E3_data_processed/test_img/7:     1263
./6-E3_data_processed/test_img/8:     1627
./6-E3_data_processed/test_img/9:      164
./6-E3_data_processed/test_img/10:      781
./6-E3_data_processed/test_img/11:      587

./6-E3_data_processed/test_img:    12500
./6-E3_data_processed/train_img/0:     6924
./6-E3_data_processed/train_img/10:     4713
./6-E3_data_processed/train_img/11:     6426
./6-E3_data_processed/train_img/1:     6935
./6-E3_data_processed/train_img/2:     2692
./6-E3_data_processed/train_img/3:     6195
./6-E3_data_processed/train_img/4:     2770
./6-E3_data_processed/train_img/5:     6547
./6-E3_data_processed/train_img/6:     6661
./6-E3_data_processed/train_img/7:     7014
./6-E3_data_processed/train_img/8:     6440
./6-E3_data_processed/train_img/9:     1683
./6-E3_data_processed/train_img:    65000

./6-E3_data_processed/val_img/0:      961
./6-E3_data_processed/val_img/1:     1223
./6-E3_data_processed/val_img/2:      697
./6-E3_data_processed/val_img/3:     1393
./6-E3_data_processed/val_img/4:      208
./6-E3_data_processed/val_img/5:     1560
./6-E3_data_processed/val_img/6:     1252
./6-E3_data_processed/val_img/7:     1147
./6-E3_data_processed/val_img/8:     1312
./6-E3_data_processed/val_img/9:      294
```

```
./6-E3_data_processed/val_img/10:     1517
./6-E3_data_processed/val_img/11:      936
./6-E3_data_processed/val_img:    12500

./6-E3_data_processed:    90000
```

三种不同的数据的统计信息如下

类型	0	1	2	3	4	5
train_img	6924	6935	2692	6195	2770	6547
类型	6	7	8	9	10	11
train_img	6661	7014	6440	1683	4713	6426

类型	0	1	2	3	4	5	6	7	8	9	10	11
val_img	961	1223	697	1393	208	1560	1252	1147	1312	294	1517	936

类型	0	1	2	3	4	5	6	7	8	9	10	11
test_img	1569	1276	689	1755	382	1243	1164	1263	1627	164	781	587

研究意义

对荧光蛋白的亚细胞图像进行分类具有以下研究意义：

1. 细胞定位和功能分析：荧光蛋白标记可以帮助研究者确定蛋白质在细胞内的定位和亚细胞结构的功能。通过对荧光蛋白的亚细胞图像进行分类，可以自动识别和定位细胞器、蛋白聚集体和其他亚细胞结构，进而深入理解细胞内的功能和过程。
2. 疾病诊断和药物研发：某些疾病或疾病亚型与细胞内亚细胞结构的异常分布和功能紊乱有关。通过对荧光蛋白的亚细胞图像进行分类，可以帮助诊断疾病并了解细胞内异常的形态学特征。此外，荧光蛋白的亚细胞图像分类也可以用于药物研发，通过识别和评估药物对亚细胞结构和功能的影响，进一步优化药物的疗效和选择性。
3. 细胞生物学研究：对荧光蛋白的亚细胞图像进行分类可以提供关于细胞内动态过程的信息，例如细胞器的形成和运动、细胞周期的变化以及蛋白质相互作用的动态。这有助于揭示细胞内分子和亚细胞结构之间的关联，以及它们在细胞功能和发育中的作用。
4. 计算机视觉和机器学习方法的应用：荧光蛋白的亚细胞图像分类可以为计算机视觉和机器学习方法的应用提供一个重要的基准任务。通过开发和改进图像分类算法，可以提高图像处理和分析的自动化程度，加速大规模图像数据的分析和解释，进而推动生命科学研究的进展。

在本次图像分类任务中，本项目选取了不同的四种**机器学习**方法和**深度学习**方法进行比较。

深度学习方法中本项目选取了**不同层数、不同结构**的网络进行**递进式**的结构功能比较。从简单到复杂，本项目选取了：

> 机器学习选取了 **SVM算法**

> 深度学习选取了 **MLP(2 layers)、MLP(3 layers)、VGG(11 layers)**作为**不同层数**的深度学习网络进行比较

> 由于网络层数加深，导致了网络退化以及训练时间过长的问题，因此，我们引入了**ResNet(18 layers)**，作为**不同结构**的深度学习网络进行比较分析

四种模型

SVM

模型介绍

- SVM属于一种线性分类器。可以理解为将一系列的数据映射到分类上。以图像分类为例，图像的像素个数理解为维数，那么每个图片在就是在这个高维空间里的一个点。SVM可以通过核方法（kernel method）进行非线性分类，是常见的核学习（kernel learning）方法之一。一些线性不可分的问题可能是非线性可分的，即特征空间存在超曲面将正类和负类分开。使用非线性函数可以将非线性可分问题从原始的特征空间映射至更高维的希尔伯特空间 ，从而转化为线性可分问题。常见的核函数有：多项式核（阶为1时称为线性核）、径向基函数核（又称RBF核、高斯核）、拉普拉斯核、Sigmoid核。
- 在本次的实验中，使用的是最常见的RBF高斯核函数，能把低维特征映射为无穷维的特征。
- 为了更好的理解高斯函数，通过一个类比来解释：比如有m个数据，每个数据是10维，那么根据公式可得到每个数据Xi和任意数据Xk的“距离”Dx，有m个数据，就计算m次，得到m个Xi与Xk的“距离”，然后将Xi里的第k个元素更换为Xi和Xk的“距离”，也就是变成m维了，因此可用于扩充特征维度，让模型对数据点有更好的认识和区分。

参数设置

- `C`：根据官方文档，这是一个软间隔分类器，对于在边界内的点有惩罚系数C，C的取值在0-1。0之间，默认值为1.0。C越大代表这个分类器对在边界内的噪声点的容忍度越小，分类准确率高，但是容易过拟合，泛化能力差。所以一般情况下，应该适当减小C，对在边界范围内的噪声有一定容忍。
- `kernel`:核函数类型，默认为‘rbf’，高斯核函数，exp(-gamma|u-v|^2),其他可选项有
  - 'linear':线性核函数，u'*v*
  - 'poly':多项式核函数，(gamma*u*'v + coef0)^degree
  - 'sigmoid':sigmoid核函数，tanh(gamma*u*'v + coef0)
- `degree`:多项式核的阶数，默认为3,对其他核函数不起作用
- `gamma`:核函数系数，只对'rbf','poly','sigmoid'起作用,浮点数，默认auto
  - 输入auto，自动让gamma = 1/n_features
  - 输入scale，使用1/(n_features*X.std())

结果解析

- 本次测试当中，发现SVM使用训练模型时间随着训练数据规模的增大而显著增大，虽然准确率有所提升，但是训练时间和占用空间较大。

训练集规模	模型建立时间	测试时间	准确率	准确预测数
5000	188.27 s	282.94 s	49.14%	6143
10000	613.59 s	460.29 s	53.70%	6713

训练集规模	模型建立时间	测试时间	准确率	准确预测数
30000	4377.41 s	1516.106	60.14%	7515
65000	568min	67min	82.65%	10375

	准确率
Train	83.43%
Validation	80.72%
Test	82.65%

- 算法缺点分析
    - SVM算法对大规模训练样本难以实施

    SVM的空间消耗主要是存储训练样本和核矩阵，由于SVM是借助二次规划来求解支持向量，而求解二次规划将涉及m阶矩阵的计算（m为样本的个数），当m数目很大时该矩阵的存储和计算将耗费大量的机器内存和运算时间。如果数据量很大，SVM的训练时间就会比较长。

    - 用SVM解决多分类问题存在困难

    经典的支持向量机算法只给出了二类分类的算法，而在数据挖掘的实际应用中，一般要解决多类的分类问题。可以通过多个二类支持向量机的组合来解决。主要有一对多组合模式、一对一组合模式和SVM决策树；再就是通过构造多个分类器的组合来解决。主要原理是克服SVM固有的缺点，结合其他算法的优势，解决多类问题的分类精度。

    - 对缺失数据敏感，对参数和核函数的选择敏感

    支持向量机性能的优劣主要取决于核函数的选取,所以对于一个实际问题而言,如何根据实际的数据模型选择合适的核函数从而构造SVM算法。目前比较成熟的核函数及其参数的选择都是人为的,根据经验来选取的,带有一定的随意性.在不同的问题领域,核函数应当具有不同的形式和参数,所以在选取时候应该将领域知识引入进来,但是目前还没有好的方法来解决核函数的选取问题。
    部分参考：https://blog.csdn.net/transformed/article/details/90437821

CNN

数据的预处理

> 如果要运行程序，需要安装go-1.20，然后执行`go run ./main.go`，需要提前把数据文件放在当前可执行文件夹下面。
>
> - 如果需要编译为可执行文件，请执行`go build -o main ./main.go`。

为了讲数据图片方便python的程序处理，用go程序编写了一个读取数据的标签并分类的程序。程序的功能主要是读取下面的txt文件，然后讲图片移动到对应的编号的文件夹。

```
Test_1.png  6
Test_2.png  7
Test_3.png  1
Test_4.png  2
```

数据一共有三组，分别是训练集合(train)、测试集合(test)、还有验证集(val)。分别拷贝到对应的文件夹。处理之后的。

```
--
|-train
        |--1
        |--2 ...
|-test
|-val
```

CNN卷积神经网络模型

作为基础模型，我们简易的搭建了一个神经网络模型（分别是两层的和三层的卷积CNN神经网络模型，对应的代码的里面的文件`common.py`和`complex.py`）。处理的数据图片是彩色的，所以是`3*64*64`规模的输入。模型使用了卷积层和全连接层的组合，通过多次卷积、激活函数和池化操作来提取图像特征，并在全连接层中进行分类预测。

- `self.conv1`是一个卷积层，输入通道数为3（RGB图像），输出通道数为5，卷积核大小为5x5。它将输入的64x64图像转换为5通道的60x60图像。
- `self.conv2`是另一个卷积层，输入通道数为5，输出通道数为10，卷积核大小为11x11。它将输入的5通道60x60图像转换为10通道的20x20图像。
- `self.conv2_drop`是一个二维Dropout层，用于随机失活部分神经元，防止过拟合。
- `self.fc1`是一个全连接层，输入大小为250（展平后的10通道5x5图像），输出大小为50。
- `self.fc2`是另一个全连接层，输入大小为50，输出大小为12（预测的类别数）。

前向传播函数（`forward`）：

- 数据通过卷积层1（`self.conv1`），接着通过ReLU激活函数和最大池化层进行处理，从而将输入图像的尺寸从64x64减小为30x30。
- 经过卷积层2（`self.conv2`），再次经过ReLU激活函数和最大池化层，将图像尺寸从30x30减小为5x5，同时应用了Dropout进行正则化。
- 通过调整张量形状（`x.view(-1, 250)`）将其展平为一维向量，方便输入全连接层。
- 数据通过第一个全连接层（`self.fc1`），再经过ReLU激活函数处理。
- Dropout被应用于第一个全连接层的输出（`F.dropout(x, training=self.training)`），在训练过程中进行正则化。
- 最后通过第二个全连接层（`self.fc2`）输出预测结果。
- 最后，通过log_softmax函数（`F.log_softmax(x)`）对输出进行log softmax转换，得到分类的概率分布。

```
class Net(nn.Module):
```

```
    def __init__(self):
        super(Net, self).__init__()

        # 单通道的64x64变成5通道的60x60
        self.conv1 = nn.Conv2d(3, 5, kernel_size=5)
        # 然后变成 5 通道的 30x30
        # 然后变成 10 通道的 20x20
        self.conv2 = nn.Conv2d(5, 10, kernel_size=11)
        self.conv2_drop = nn.Dropout2d()
        # 然后变成 10 通道的 5x5
        self.fc1 = nn.Linear(250, 50)
        self.fc2 = nn.Linear(50, 12)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 4))
        # x = F.relu(F.max_pool2d(self.conv3_drop(self.conv3(x)), 5))
        x = x.view(-1, 250)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x)
```

考虑到两层的神经网络预测的结果并不是很理想，于是乎又尝试了三层的神经网络：

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        # 单通道的64x64变成5通道的60x60
        self.conv1 = nn.Conv2d(3, 5, kernel_size=5)
        # 然后变成 5 通道的 30x30
        # 然后变成 10 通道的 24x24
        self.conv2 = nn.Conv2d(5, 10, kernel_size=7)
        self.conv2_drop = nn.Dropout2d()
        # 然后变成 10 通道的 12x12
        self.conv3 = nn.Conv2d(10, 20, kernel_size=3)
        # 现在变成了 20通道的 10x10
        self.conv3_drop = nn.Dropout2d()
        # 现在编程了 20通道的 5x5
        self.fc1 = nn.Linear(500, 250)
        self.fc2 = nn.Linear(250, 12)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = F.relu(F.max_pool2d(self.conv3_drop(self.conv3(x)), 2))
        x = x.view(-1, 500)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
```

```
27          return F.log_softmax(x)
```

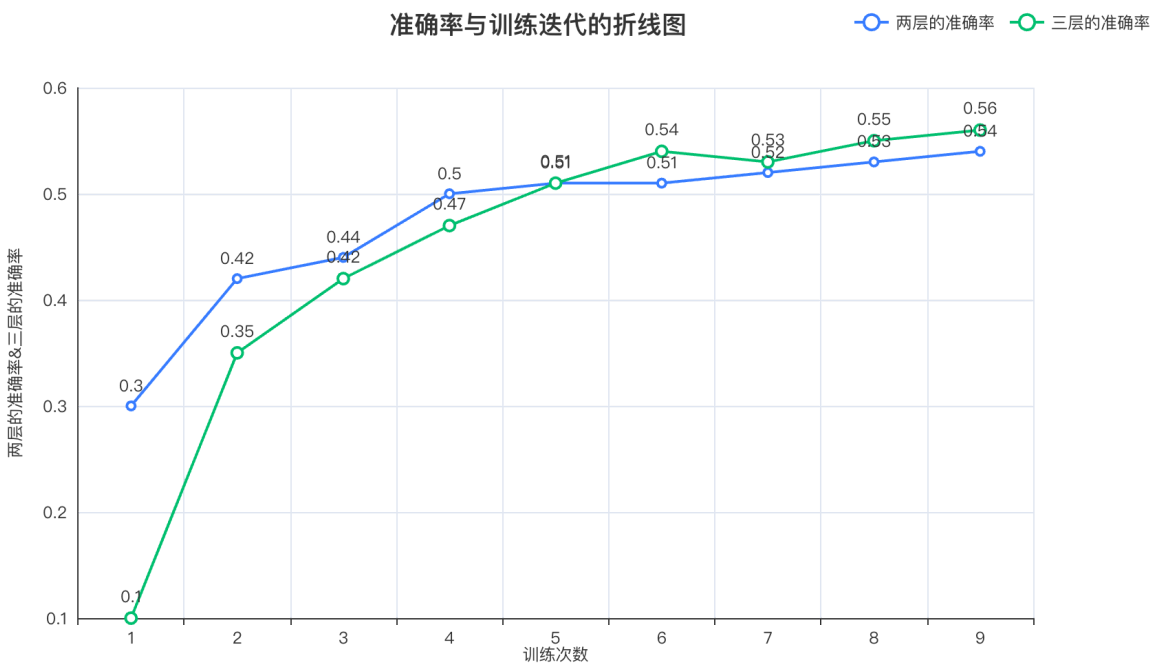
通过三层的神经网络预测的，但是结果并没有很大的优势和提升。

训练结果

- 下面的表格是训练的结果，数据代表的是通过训练的模型预测的准确度

迭代次数	两层的准确率	三层的准确率
1	0.3	0.1
2	0.42	0.35
3	0.44	0.42
4	0.5	0.47
5	0.51	0.51
6	0.51	0.54
7	0.52	0.53
8	0.53	0.55
9	0.54	0.56

基于数据绘图，可以看到



简易分析

- 三层的神经网络预测的结果的准确度要更高，但是相比较于两层的神经网络并不是非常的明显
- 最终三层CNN神经网络预测的准确率是56%，而两层的为54%
- 两层的训练过程中，起步的起点较高，三层的训练过程中，起点虽然较低，但是增长较快。
- 最终两层和三层的预测的准确度差别不大，推测本质原因还是模型过于简单，导致预测精度无法提高，因此本模型更适合作为一个基础，和后面的相关的模型比较。
- 更多模型训练的详细细节输出可以参考 result 的的文本文件。

VGG-like 实现

模型背景介绍

VGG-16架构是一个卷积神经网络（CNN）模型，由牛津大学的视觉几何小组（VGG）提出。它在2014年的ImageNet大规模视觉识别挑战赛（ILSVRC）中取得了出色的表现。VGG-16中的 "16 "指的是网络中的总层数，包括卷积层、池化层和全连接层。

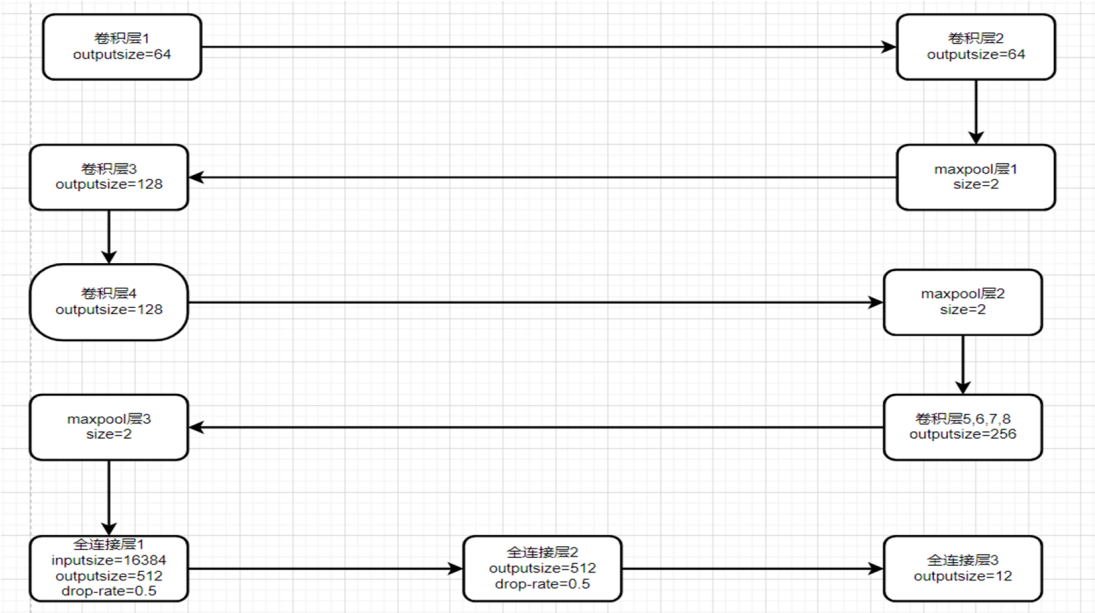
VGG-16网络将固定尺寸的RGB图像作为输入，通常尺寸为224x224像素；其首先包括了13个卷积层，使用跨度为1的小型3x3滤波器，并使用零填充来保持特征图的空间尺寸。其激活函数使用RELU函数，进而引入非线性同时避免梯度消失或爆炸。在每两个卷积层之后，VGG-16有一个maxpool层，窗口大小为2x2。在卷积层和池化层之后，VGG-16有三个全连接层，Dropout被用来规范网络并防止过度拟合。最后一个全连接层有1000个神经元，对应于ImageNet数据集中的1000个类别。应用Softmax激活来产生一个关于类的概率分布。

下图是其基本架构：

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

模型实现

由于VGG-16网络结构较为复杂，导致网络参数多，训练时间较长，我们不太可能实现。因此，我们决定对其进行某些改动，进而实现自己的VGG16 like 网络结构。我们的网络结构如下：



我们的模型基本沿袭了VGG-16的各个特点，包括了：

- 卷积层size=3;padding=1;进而保持特征图的空间尺寸
- 数个卷积层之后会使用pool层，其为size=2的maxpool
- 每一层之后使用BatchNorm保持数据正态性
- 激活函数使用RELU函数
- 最终使用三个全连接层，同时最后一个全连接层有12个神经元，对应于数据集中的12个类别。应用Softmax激活来产生一个关于类的概率分布。
- 使用了带动量的SGD优化器，momentum=0.9;learning-rate=0.005;同时使用了L2正则化，weight_decay=0.0005
- 使用了交叉熵损失函数

训练中损失函数值变化

下图是训练中损失函数值变化图，由于存在多次迭代，每一次迭代交换时，就会导致loss_function值突变。但明显我们能够发现，loss函数值随悬链进行逐渐减少，符合训练尝试与经验。

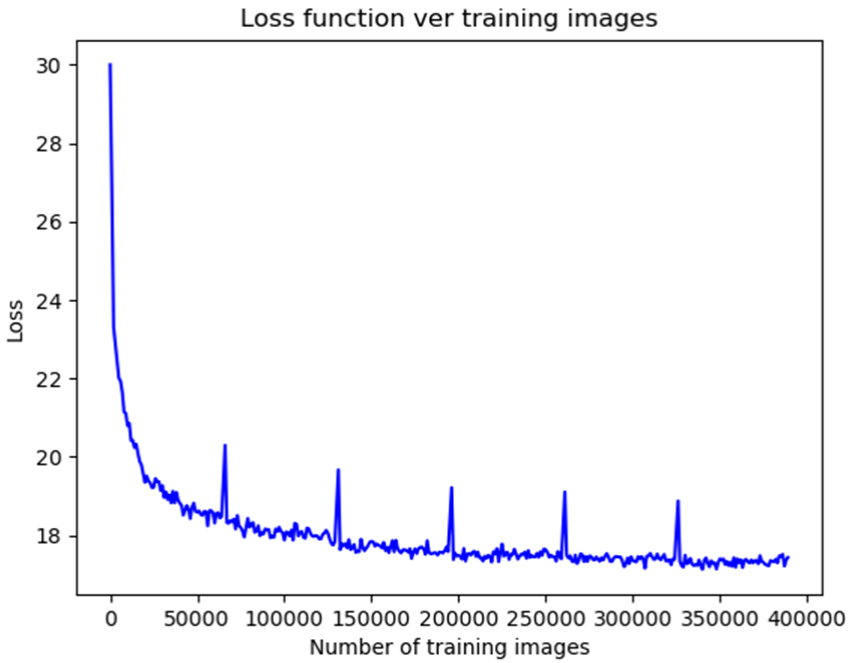
训练结果

在protein.out文件中，包含了程序的输出(训练进度、损失函数值、验证集精度)：

```
for epoch 1 1000/65000 loss is 26.85
for epoch 1 2000/65000 loss is 23.31
for epoch 1 3000/65000 loss is 22.87
for epoch 1 4000/65000 loss is 22.40
for epoch 1 5000/65000 loss is 22.01
for epoch 1 6000/65000 loss is 21.93
for epoch 1 7000/65000 loss is 21.65
for epoch 1 8000/65000 loss is 21.14
for epoch 1 9000/65000 loss is 21.11
……
for this epoch1, the accuracy of validation  set is 0.71

```

我们最终得到如下的训练结果：



迭代次数	预测精度
1	0.71
2	0.75
3	0.81
4	0.82
5	0.82
6	0.86
测试集	0.85

总结

总的说来，VGG-16 like模型在荧光蛋白分类任务中取得了很好的效果，这一方面是已有成熟的研究结果，一方面是CNN模型在图像处理领域的天然优势导致的。我们认为该模型较为成功地完成了目标任务。

ResNet18

模型引入

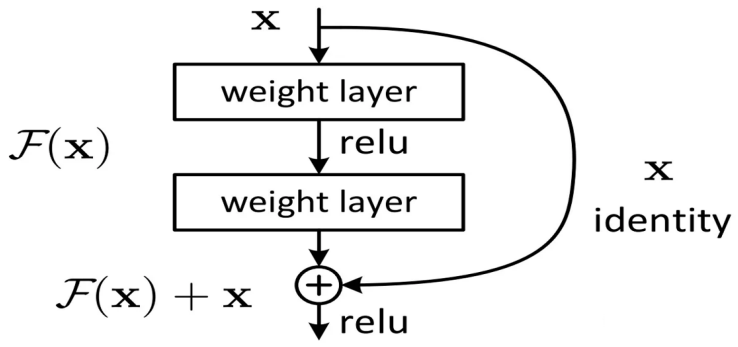
Q：为什么会出现Resnet18的结构？串联结构的VGG不行吗？

A:

- 随着网络越来越深，训练变得原来越难，网络的优化变得越来越难。理论上，越深的网络，效果应该更好；但实际上，由于训练难度，过深的网络会产生退化问题，效果反而不如相对较浅的网络(随着层数的增多，训练集上的效果变差,这被称为退化问题)
- 随着网络越来越深, 堆叠到一定网络深度时，就会出现梯度消失或梯度爆炸问题

不论是多少层的ResNet网络，它们都有以下共同点：

- 网络一共包含5个卷积组，每个卷积组中包含1个或多个基本的卷积计算过程（Conv-> BN->ReLU）
- 每个卷积组中包含1次下采样操作，使特征图大小减半，下采样通过以下两种方式实现：
- - 最大池化，步长取2，只用于第2个卷积组（Conv2_x）
  - 卷积，步长取2，用于除第2个卷积组之外的4个卷积组
- 第1个卷积组只包含1次卷积计算操作，5种典型ResNet结构的第1个卷积组完全相同，卷积核均为7x7， 步长为均2
- 第2-5个卷积组都包含多个相同的残差单元，在很多代码实现上，通常把第2-5个卷积组分别叫做Stage1、Stage2、Stage3、Stage4
- 首先是第一层卷积使用kernel 7∗7，步长为2，padding为3。之后进行BN，ReLU和maxpool。这些构成了第一部分卷积模块conv1。
- 然后是四个stage，有些代码中用make_layer()来生成stage，每个stage中有多个模块，每个模块叫做building block，resnet18= [2,2,2,2]，就有8个building block。注意到他有两种模块**BasicBlock**和**Bottleneck**。resnet18和resnet34用的是**BasicBlock**，resnet50及以上用的是**Bottleneck**。无论**BasicBlock**还是**Bottleneck**模块，都用到了**残差连接(**shortcut connection)方式：



ResNet18结构介绍

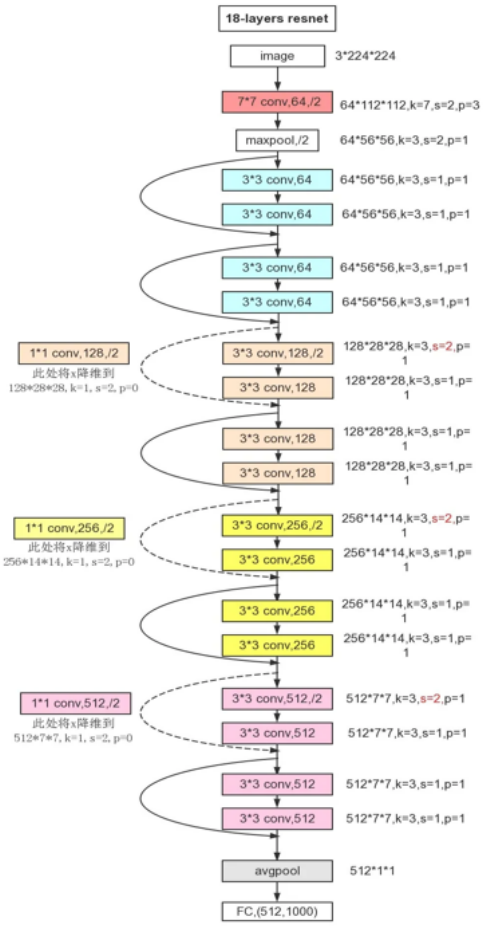
ResNet18的18层代表的是带有权重的18层，包括卷积层和全连接层，不包括池化层和BN层。

layer1

ResNet18，使用的是`BasicBlock`。`layer1`，特点是没有进行降采样，卷积层的`stride = 1`，不会降采样。在进行`shortcut`连接时，也没有经过`downsample`层。

layer2，layer3，layer4

而`layer2`，`layer3`，`layer4`的结构图如下，每个`layer`包含 2 个`BasicBlock`，但是第 1 个`BasicBlock`的第 1 个卷积层的`stride = 2`，会进行降采样。在进行`shortcut`连接时，会经过`downsample`层，进行降采样和降维。



数据加载

由于给出的label.txt和img在不同的文件中，希望使用`datasets.ImageFolder()`函数加载，所以对于文件格式进行处理，根据不同的label将图片分在不同的文件夹中。

```
def classify_data(txt_path,labels):
    fh = open(txt_path, 'r', encoding='utf-8')
    lines = fh.readlines()
    cnt=0
    for line in lines:
        line = line.strip('\n')  # 移除字符串首尾的换行符
        line = line.rstrip()  # 删除末尾空
```

```
        words = line.split()  # 以空格为分隔符 将字符串分成两部分
        imgs_name = words[0]  # imgs中包含有图像路径和标签
        src_path="E:/大三下/生物大数据分析/project/6-E3_data/6-E3_data/train_img"
        srcfile = src_path+'/'+imgs_name
        imgs_label = int(words[1])
        target_pth="E:/大三下/生物大数据分析/project/data/train/"
        # shutil.copy(srcfile, target_pth+labels[imgs_label])
        # cnt+=1
        for root, dirs, files in os.walk(src_path):
            for filename in files:
                if imgs_name == filename:
                  shutil.copy(srcfile, target_pth+labels[imgs_label])
                  cnt+=1
                  break
                # else:
                #   print(imgs_name+' not found!')

    print("Copy files Successfully!")
    print(cnt)
```

数据预处理

input size

由于原始ResNet训练图像是224*224，所以有两种方案：

> 方案一

将图片resize成224*224

> 方案二

直接输入图片，大小为64*64

经过预实验最终决定将图片resize

归一化、正则化

正则化的目标是降低模型的复杂数度，减少过拟合现象，提升模型的泛化能力。使得损失函数跟容易找到最值，使模型更加简单、稳定。对于三通道255像素图像，我们:

$$pixel = pixel/255$$

$$pixel = \frac{pixel-mean}{sd}$$

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

模型搭建

选择交叉熵损失函数

```
criterion = nn.CrossEntropyLoss()
```

选择简单的Adam优化器，使用`adjust_learning_rate`函数将学习率调低，随着训练的epoch逐渐减小，从而更好的逼近最优值：

```
optimizer = optim.Adam(model.parameters(), lr=modellr)

def adjust_learning_rate(optimizer, epoch):
    """Sets the learning rate to the initial LR decayed by 10 every 30 epochs"""
    modellrnew = modellr * (0.1 ** (epoch // 50))
    print("lr:", modellrnew)
    for param_group in optimizer.param_groups:
        param_group['lr'] = modellrnew
```

从`pytorch`库中实例化`resnet18`,并且加载`imagenet`的预训练参数。

```
model = torchvision.models.resnet18(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs, 12)
```

模型训练

每1600样本输出loss

```
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    sum_loss = 0
    total_num = len(train_loader.dataset)
    print(total_num, len(train_loader))
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data).to(device), Variable(target).to(device)
        output = model(data)
        loss = criterion(output, target)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        print_loss = loss.data.item()
        sum_loss += print_loss
        if (batch_idx + 1) % 50 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, (batch_idx + 1) * len(data), len(train_loader.dataset),
                       100. * (batch_idx + 1) / len(train_loader),
loss.item()))
            writer.add_scalar('Train_loss', loss.item(), epoch)
```

```
    ave_loss = sum_loss / len(train_loader)
    print('epoch:{},loss:{}'.format(epoch, ave_loss))
    writer.close()
```

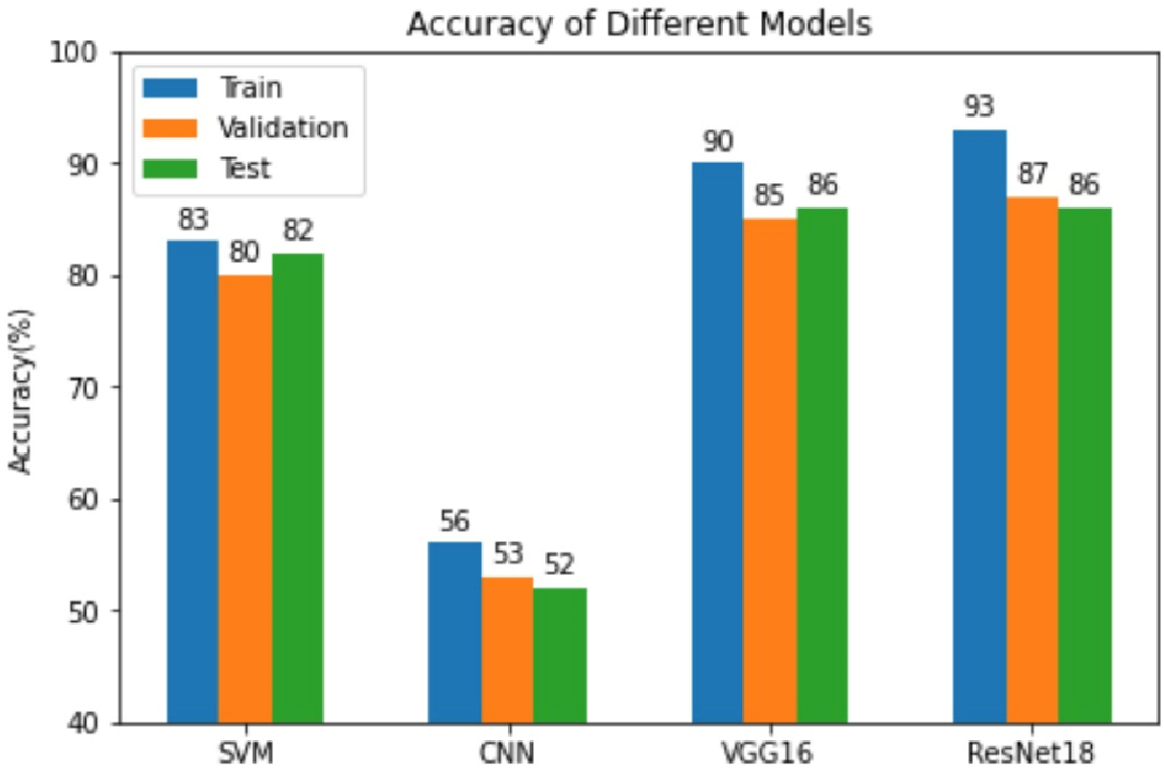
模型验证

```
def val(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    total_num = len(test_loader.dataset)
    print(total_num, len(test_loader))
    with torch.no_grad():
        for data, target in test_loader:
            data, target = Variable(data).to(device),
Variable(target).to(device)
            output = model(data)
            loss = criterion(output, target)
            _, pred = torch.max(output.data, 1)
            correct += torch.sum(pred == target)
            print_loss = loss.data.item()
            test_loss += print_loss
        correct = correct.data.item()
        acc = correct / total_num
        avgloss = test_loss / len(test_loader)
        print('\nVal set: Average loss: {:.4f}, Accuracy: {}/{}
({:.0f}%)\n'.format(
            avgloss, correct, len(test_loader.dataset), 100 * acc))
```

训练结果

dataset	Accuracy
Train	93%
Test	86%
Val	87%

四种模型结果和性能指标可视化



	Time(min)
SVM	~568 (CPU)
CNN	~15 (GPU)
VGG	~120(CPU)
ResNet	~50 (GPU)

机器学习模型与深度学习模型之间的比较

根据结果，我们可以看出，SVM的分类能力可以超过简单的卷积神经网络CNN，这是由于：

深度学习是数据驱动的，因此在数据量小的情况下，深度学习模型很容易过拟合。在我们的实验中，train数据集的准确率明显更高，SVM无需大量调参，选择合适的kernel函数后，具有很好的非线性分类能力，在数据量不大的情况也展现出很好的分类能力。

但是SVM的训练时间明显长于深度学习网络，这是由于在深度学习中我们使用GPU加速进行矩阵运算，深度学习算法在很大程度上依赖于高端机器，这与传统的机器学习算法相反，后者可以在低端机器上运行。深度学习算法本质上是做大量的矩阵乘法运算，而使用GPU可以有效的优化这些操作。

深度学习模型之间的比较

深度学习相比于机器学习需要大量的超参数设置，从图中可以看到，随着网络层数的加深，VGG与CNN相比，准确率提高，但是于此同时，网络层越深，训练时间越长，而且越容易遇到网络退化问题，因此我们引入ResNet，通过**残差连接(shortcut connection)**解决退化问题，可以看到18 layer的ResNet能够达到本数据集的最好的分类效果。这说明ResNet在解决退化问题上是十分有效的。

总结与启示

机器学习和深度学习没有一定的优劣之分，其中，选取适当的kernel函数，机器学习会在较小量数据集发挥更好的非线性分类效果，但是需要较长的训练时间。

对于深度学习训练

- 覆盖特征全面的数据集
- 合适的超参数集（如网络结构、
- 合适的网络优化结构（如basicblock）

更够达到更好的训练结果。