—武大本科生课程

第15讲　PyTorch应用案例

(Lecture 15 Application Cases for PyTorch)

机器学习课程组

应用举例

- 应用1：使用LeNet5进行图像分类
- 应用2：使用UNet进行医学图像分割
- 附录：Jupyter Notebook使用教程

应用1：使用LeNet5进行图像分类

一、实验背景及意义

二、图像分类任务介绍

三、实验数据准备

四、图像分类方法概述与模型选择(LeNet5)

五、PyTorch代码实现

六、使用Pytorch环境进行训练

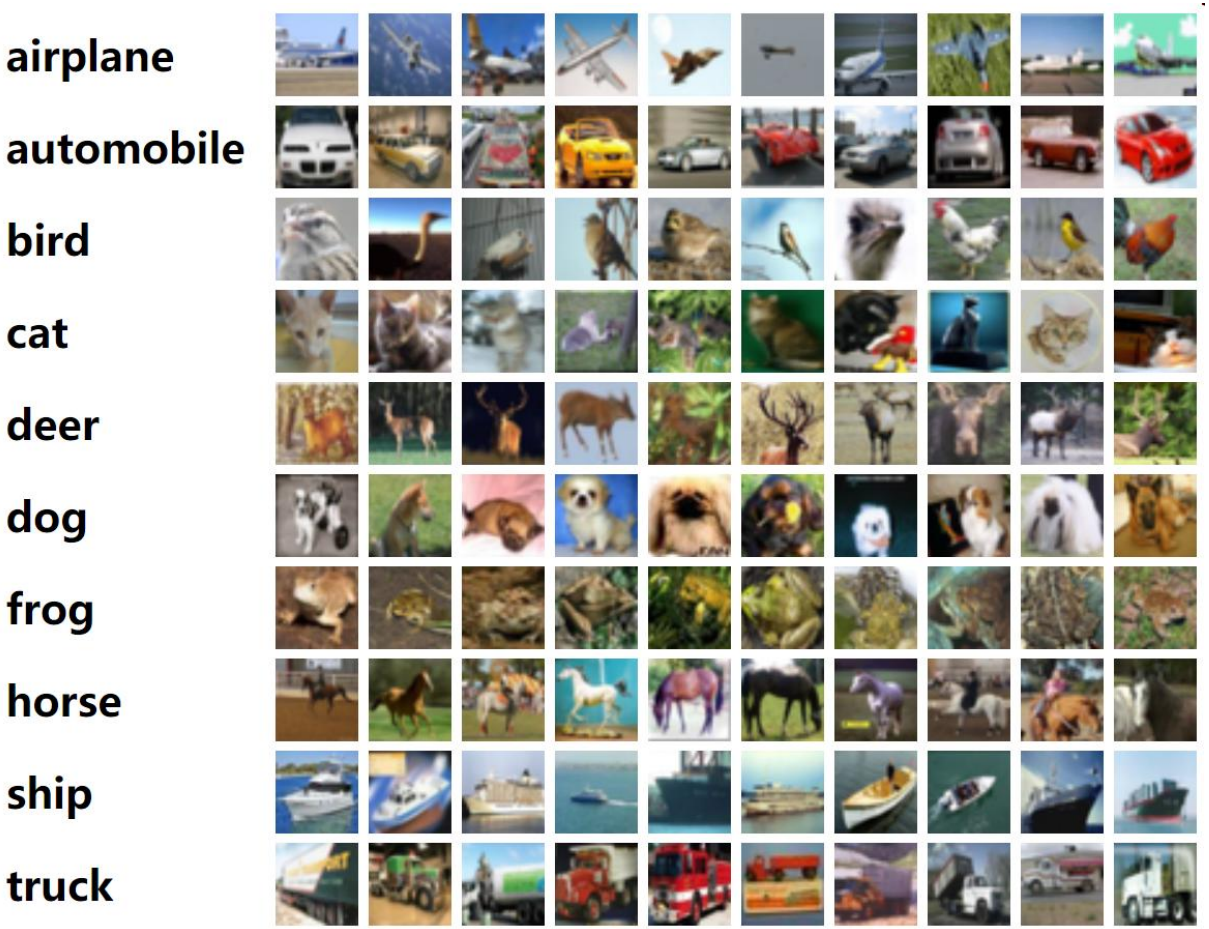
七、案例总结

一、实验背景及意义

- 实验目的：熟悉PyTorch深度学习框架，熟悉开发平台
- 实验背景：图像分类——海鲜市场不同类型的鱼分类
- 模型选择：LeNet5（经典的卷积神经网络模型）

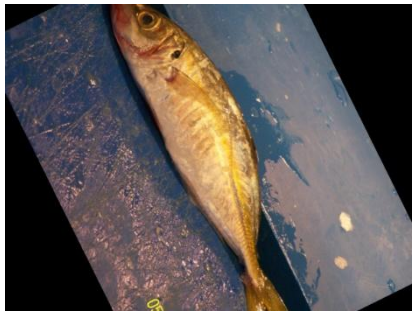
二、图像分类任务介绍

图像分类是一项比较基础的任务，该任务的目标是理解整个图像，并给图像一个类别标签。图像分类是计算机视觉一个重要的研究方向。

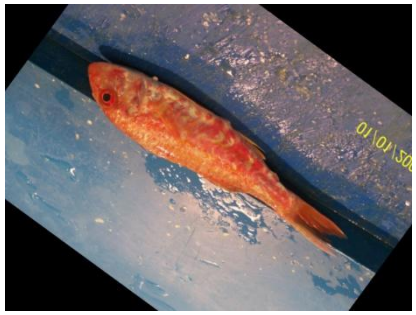


三、实验数据准备

- **数据集名称：**A Large Scale Fish Dataset
- **数据集说明：**该数据集包含从土耳其伊兹密尔的一家超市收集的9种不同的海鲜类型，每种类型包含1000张图片。本实验案例中，选取其中的Hourse Mackerel、Red Mullet、Shrimp、Trout 这4种进行四分类任务（数据大小1.28G）。
- **数据集划分：**90%的图片用于构建训练集，10%的图片用于构建测试集。fish_train_list.txt 文件记录了每个用于训练的图片的文件路径以及其标签；fish_test_list.txt 文件记录了每个用于测试的图片的文件路径以及其标签。



Hourse Mackerel



Red Mullet

Shrimp

Trout

四、图像分类方法概述

- 在传统的模式识别模型中，需要手工设计特征提取器来从输入中收集相关信息，并消除不相关的变量。接着，一个可训练的分类器基于得到的特征进行分类，得到分类结果。在这样的方案中，标准的多层全连接网络可以用作分类器。

- 一个可能更好的方案是尽可能多地依赖于特征提取器本身的学习。虽然这可以通过普通的全连接前馈网络来实现，并且对于诸如字符识别之类的分类任务也取得了一些成功，但是仍然存在一些问题。

使用全连接网络进行图像分类存在的问题

- 问题一：通常图像的像素很多，通常有几百个变量(像素)。若全连接层第一层的具有一百个单元，则第一层将已经包含几万个权重。如此大量的参数增加了系统的容量，因此需要更大的训练集。这样会增加训练的难度。

- 问题二：全连接网络架构的忽略了输入的拓扑结构，其输入变量可以以任何顺序呈现，而不会影响训练的结果。然而，图像具有很强的二维局部结构：空间上邻近的像素高度相关。全连接网络忽略了这种空间结构。

卷积神经网络介绍

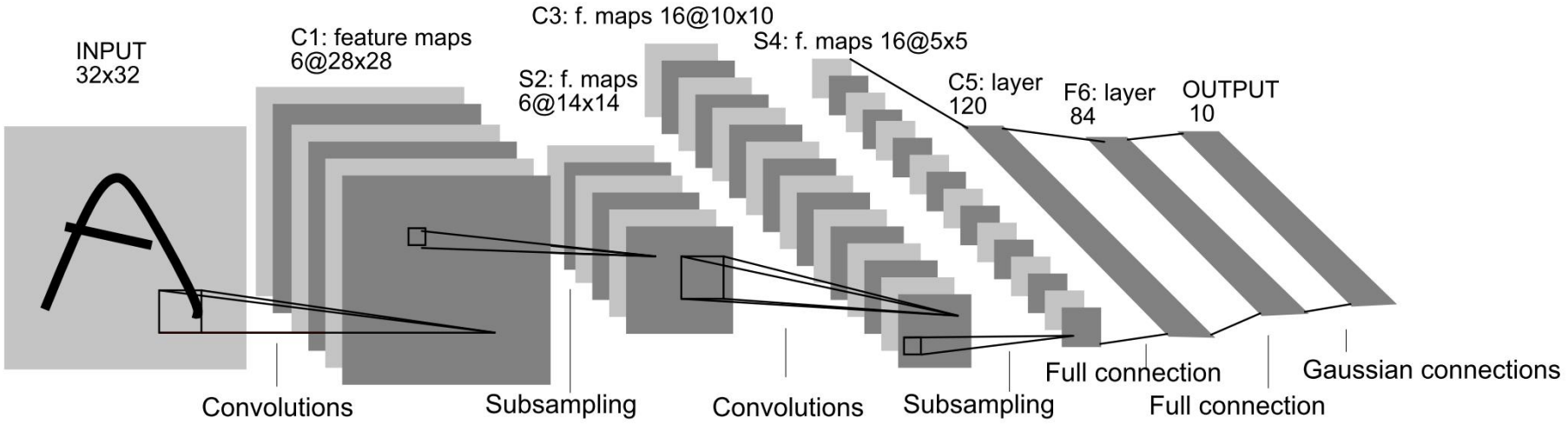
- 卷积神经网络（CNN）结合了三种架构思想：局部感受野、共享权重（或权重复制）、空间或时间子采样，以确保一定程度的移位、缩放和失真不变性。

- CNN的局部感受野和共享权重特点，一方面减少了的权值的数量使得网络易于优化，另一方面降低了过拟合的风险。

- 一个卷积神经网络通常包括输入输出和多个隐藏层，隐藏层通常具有由卷积层、池化层（下采样层）、激活函数等网络层。

模型选择——LeNet5

- 模型结构图：LeNet5



LeNet5介绍：

LeNet5是由Yann LeCun等人1998年在论文Gradient-Based Learning Applied to Document Recognition提出的模型。LeNet5是LeNet多次迭代后的模型，该模型是一个在手写体字符分类任务上非常高效的**卷积神经网络**，该网络在其他图片分类任务上也表现不错。

案例中的LeNet5网络结构详解

本案例中，将图片都缩放为128×128分辨率，再输入到LeNet5网络中。本案例的LeNet5网络的各层结构细节如下：

1. INPUT：分辨率为128×128的RGB图片。
2. C1层：该层为第一个卷积层，卷积核的大小为5×5，该层输出6×124×124的特征图。
3. S2层：该层为池化层（下采样层），过滤器尺寸为2×2，采用最大池化，该层输出6×62×62的特征图。
4. C3层：该层为第二个卷积层，卷积核的大小为5×5，该层输出6×58×58的特征图。
5. S4层：该层为池化层（下采样层），过滤器尺寸为2×2，采用最大池化，该层输出6×29×29的特征图。
6. C5层：该层为第三个卷积层，使用120个尺寸为29×29的过滤器，该层输出1×1×120的特征图。该层也可看作是输入单元数量120×29×29，输出单元数量为120的全连接层。
7. F6层：该层是全连接层，该全连接层输入有120个单元，输出有84个单元。
8. F7层：该层是全连接层，该全连接层输入84个单元，输出有4个单元。这是因为本案例是4分类，所以最后输出是4个单元。

五、PyTorch代码实现

- 代码文件的结构图如下：

```
classification
├── four_fish
|   ├── Hourse Mackerel
|   |   ├── 00001.png
|   |   ├── 00002.png
|   |   ├── ...
|   |   └── 01000.png
|   ├── Red Mullet
|   ├── Shrimp
|   ├── Trout
|   ├── fish_test_list.txt
|   └── fish_train_list.txt
├── data_process.py
├── dataset.py
├── model.py
├── main.py
└── README.md
```

- four_fish文件夹存放数据集图片
- data_process.py 用于处理数据集，生成fish_test_list.txt和fish_trian_list.txt。
- dataset.py是用于创建读取数据的Dataset类。
- model.py用于构建LeNet5网络模型
- main.py是主函数启动文件

基于PyTorch构建LeNet5网络

```python
class LeNet5(nn.Module):
    def __init__(self, num_class=4, num_channel=3):
        super().__init__()
        self.conv1 = nn.Conv2d(num_channel, 6, 5)  # 默认无padding
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16*29*29, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, num_class)
        self.relu = nn.ReLU()
        self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
        self.flatten = nn.Flatten()
        self.drop_out = nn.Dropout()

    def forward(self, x):
        x = self.max_pool2d(self.drop_out(self.relu(self.conv1(x))))
        x = self.max_pool2d(self.drop_out(self.relu(self.conv2(x))))
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.drop_out(x)
        x = self.relu(self.fc2(x))
        x = self.drop_out(x)
        x = self.fc3(x)
        return x
```

定义LeNet5网络需要的操作

构建LeNet5网络

构建数据集生成器

```python
class FishDataset(data.Dataset):
    def __init__(self,dataset_path, txt_path):
        print(f'create_dataset: dataset_path={dataset_path}, txt_path={txt_path}')
        f = open(txt_path)
        lines = f.readlines()
        f.close()
        images = []
        labels = []
        for line in lines:
            col = line.strip().split(',')
            image_path = os.path.join(dataset_path, col[0])
            np_image = np.array(Image.open(image_path)).astype(np.float32)
            images.append(np_image)
            labels.append(int(col[1]))

        self.images = images
        self.labels = labels

    def __getitem__(self, index):
        return self.images[index], self.labels[index]

    def __len__(self):
        return len(self.images)
```

定义读取数据的Dataset类，需要继承torch.utils.data.Dataset类，并实现__init__()、__getitem__()和__len__()方法。

定义训练过程

- main.py：

```python
num_class = 4 # 四分类问题
dataset_path = './four_fish'
train_txt = './four_fish/fish_train_list.txt'
test_txt = './four_fish/fish_test_list.txt'
save_path = './results/' # 模型保存的位置
device = torch.device("cuda" if torch.cuda.is_available() else "cpu") # GPU是否可用

def train():
    net = LeNet5(num_class=num_class).to(device)
    data_train = FishDataset(dataset_path, train_txt)
    data_train_loader = DataLoader(data_train,batch_size=32) # 一次读取32个数据
    criterion = nn.CrossEntropyLoss() # 使用交叉熵损失函数
    optimizer = optim.SGD(net.parameters(), lr = 0.001, momentum=0.9) # 初始学习率为0.001
    net.train() # 设置为训练状态
    for epoch in range(25): # 训练25个epoch
        for ii, (images, labels) in enumerate(data_train_loader):
            images = images.to(device)
            labels = labels.to(device)
            optimizer.zero_grad() # 训练开始之前将梯度清零
            output = net(images)
            loss = criterion(output, labels.long()) # 计算损失
            loss.backward() # 反向传播计算梯度
            optimizer.step() # 根据梯度使用SGD优化器更新网络参数
            if ii % 10 == 0: # 每10个batch打印一次信息
                print('Train - Epoch %d, Batch: %d, Loss: %f' % (epoch, ii, loss.detach().cpu().item()))
        if (epoch + 1) % 5 == 0: # 每5个epoch保存一下模型
            torch.save(net.state_dict(), save_path+'checkpoint_%d.pth' % (epoch))
```

设置训练参数，构建网络、损失函数、优化器等，然后一次读取batch_size的数据对网络进行训练，在固定epoch内保存训练好的模型。

模型验证

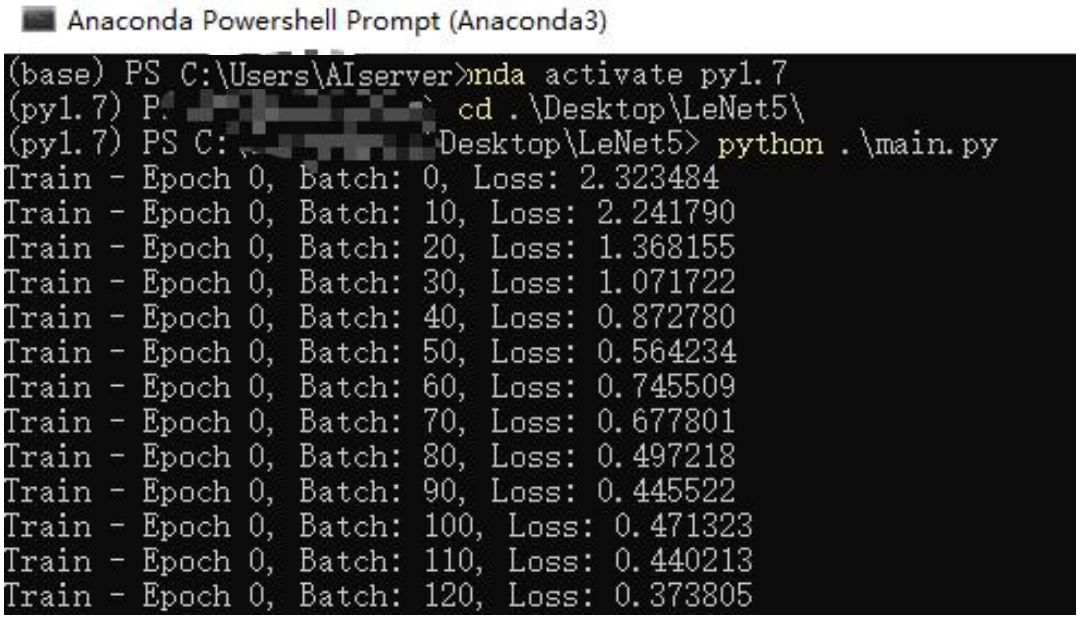
- main.py：

```
def test(epoch):
    net = LeNet5(num_class=num_class).to(device)
    net.load_state_dict(torch.load(save_path+'checkpoint_%d.pth' % (epoch)))
    data_test = FishDataset(dataset_path, test_txt)
    data_test_loader = DataLoader(data_test,batch_size=32)
    criterion = nn.CrossEntropyLoss()
    net.eval() # 设置为测试状态
    total_correct = 0
    avg_loss = 0.0
    with torch.no_grad(): # 测试阶段不需要计算梯度
        for i, (images, labels) in enumerate(data_test_loader):
            images = images.to(device)
            labels = labels.to(device)
            output = net(images)
            loss = criterion(output, labels.long())
            avg_loss += loss.sum()
            pred = output.max(dim=1)[1] # 根据概率获取类别
            correct = pred.eq(labels.view_as(pred)).sum()
            total_correct += correct
            print('Test Loss: %f, Accuracy: %f' % (loss.detach().cpu().item(), float(correct)/pred.view(-1).shape[0]))
    avg_loss /= len(data_test)
    print('Test Avg. Loss: %f, Accuracy: %f' % (avg_loss.cpu().item(), float(total_correct)/len(data_test)))
```

定义测试模型的函数。加载模型参数到网络中，调用test函数进行模型验证，验证指标采用分类的准确率。

六、使用PyTorch环境进行训练

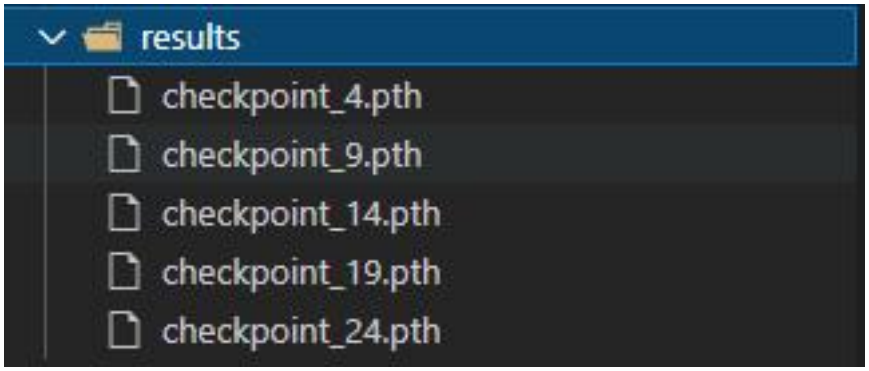
- Step 1：打开Anaconda终端，进入Pytorch环境
  - 执行conda activate pytorch
  - cd 到对应文件目录
  - 运行python main.py开始训练
  - Anaconda安装和Pytorch环境搭建参考附录



使用PyTorch环境进行训练

- Step 2：等待训练完成，训练过程中可以查看训练日志输出
  - 训练好的模型将保存在results目录下

```
Train - Epoch 0, Batch: 0, Loss: 2.343844
Train - Epoch 0, Batch: 10, Loss: 2.327393
Train - Epoch 0, Batch: 20, Loss: 2.280467
Train - Epoch 0, Batch: 30, Loss: 2.309814
Train - Epoch 0, Batch: 40, Loss: 2.309140
Train - Epoch 0, Batch: 50, Loss: 2.296454
Train - Epoch 0, Batch: 60, Loss: 2.286792
Train - Epoch 0, Batch: 70, Loss: 2.296350
Train - Epoch 0, Batch: 80, Loss: 2.269650
Train - Epoch 0, Batch: 90, Loss: 2.242959
Train - Epoch 0, Batch: 100, Loss: 2.294012
Train - Epoch 0, Batch: 110, Loss: 2.264864
Train - Epoch 0, Batch: 120, Loss: 2.269121
Train - Epoch 0, Batch: 130, Loss: 2.260136
Train - Epoch 0, Batch: 140, Loss: 2.225929
Train - Epoch 0, Batch: 150, Loss: 2.279640
Train - Epoch 0, Batch: 160, Loss: 2.281759
Train - Epoch 0, Batch: 170, Loss: 2.262054
Train - Epoch 0, Batch: 180, Loss: 2.292073
Train - Epoch 0, Batch: 190, Loss: 2.242172
Train - Epoch 0, Batch: 200, Loss: 2.233873
```



使用PyTorch环境进行训练

- Step 3：修改main.py文件，调用test函数对模型进行测试

```
if __name__ == '__main__':
    # train()
    test(24) # test函数需要传入加载的模型训练的epoch
```

```
y
Test Loss: 0.031277, Accuracy: 1.000000
Test Loss: 0.033472, Accuracy: 1.000000
Test Loss: 0.049245, Accuracy: 1.000000
Test Loss: 0.040098, Accuracy: 1.000000
Test Loss: 0.029589, Accuracy: 1.000000
Test Loss: 0.056787, Accuracy: 1.000000
Test Loss: 0.045214, Accuracy: 1.000000
Test Loss: 0.033197, Accuracy: 1.000000
Test Loss: 0.067578, Accuracy: 0.968750
Test Loss: 0.033510, Accuracy: 1.000000
Test Loss: 0.129931, Accuracy: 0.937500
Test Loss: 0.029680, Accuracy: 1.000000
Test Loss: 0.045398, Accuracy: 1.000000
Test Loss: 0.095517, Accuracy: 0.968750
Test Loss: 0.068027, Accuracy: 0.968750
Test Loss: 0.057429, Accuracy: 1.000000
Test Loss: 0.033103, Accuracy: 1.000000
```

七、案例总结

```
Test Loss: 0.010285, Accuracy: 1.000000
Test Loss: 0.013265, Accuracy: 1.000000
Test Loss: 0.057992, Accuracy: 0.968750
Test Loss: 0.027841, Accuracy: 1.000000
Test Loss: 0.014091, Accuracy: 1.000000
Test Loss: 0.036418, Accuracy: 1.000000
Test Loss: 0.101254, Accuracy: 0.968750
Test Loss: 0.124046, Accuracy: 1.000000
Test Loss: 0.043950, Accuracy: 1.000000
Test Loss: 0.092504, Accuracy: 0.968750
Test Loss: 0.093877, Accuracy: 0.968750
Test Loss: 0.026068, Accuracy: 1.000000
Test Loss: 0.037357, Accuracy: 1.000000
Test Loss: 0.035356, Accuracy: 1.000000
Test Loss: 0.057884, Accuracy: 1.000000
Test Loss: 0.024208, Accuracy: 1.000000
Test Loss: 0.036718, Accuracy: 1.000000
Test Loss: 0.007455, Accuracy: 1.000000
Test Avg. Loss: 0.001569, Accuracy: 0.991400
```

本实验展示了如何使用Pytorch深度学习框架进行模型搭建、训练、验证等操作。整个实验围绕搭建LeNet5模型，并实现图片四分类任务而进行。通过对LeNet5模型做迭代训练，接着使用训练后的LeNet5模型对数据集进行评估验证，准确率达到99.14%，即说明LeNet5模型很好地学习到了如何进行对图片进行分类。

应用2：使用UNet进行医学图像分割

一、实验背景及意义

二、图像分割简介

三、实验数据准备

四、模型选择：UNet

五、PyTorch代码实现

六、使用Pytorch进行训练

七、案例总结

一、实验背景及意义

- 实验目的：熟悉基于深度学习的医学图像分割
- 实验背景：医学图像分割——电子显微镜下的细胞分割（果蝇龄期幼虫腹侧神经索细胞）
- 模型选择：U-net（生物医学图像分割领域的经典模型）

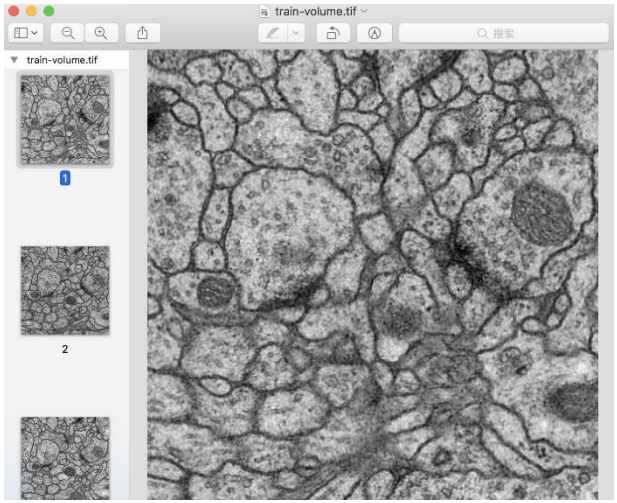
二、图像分割简介

在图像分割中，机器必须将图像分割成不同的segments，每个segment代表不同的实体，如左图所示，各个实体被分割出来。图像分割在从自动驾驶汽车到卫星的许多领域都很有用，也许其中最重要的是医学影像。医学图像的微妙之处是相当复杂的。一台能够理解这些细微差别并识别出必要区域的机器，可以对医疗保健产生深远的影响。

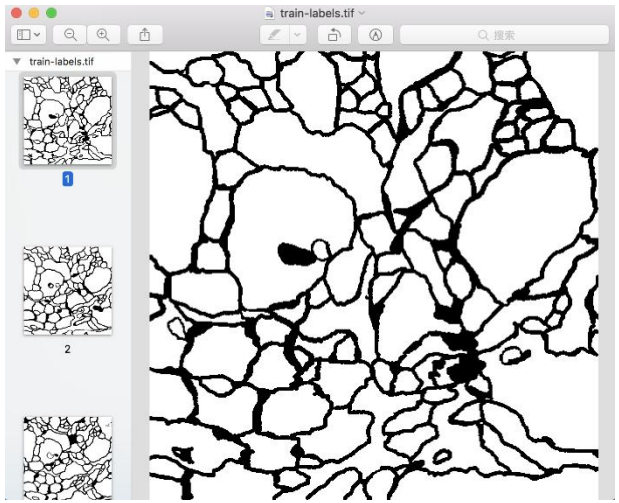
卷积神经网络在简单的图像分割问题上取得了不错的效果，但在复杂的图像分割问题上却几乎没有进展。这就是UNet的作用。UNet最初是专门为医学图像分割而设计的，该方法取得了良好的效果，并在以后的许多领域得到了应用。

三、实验数据准备

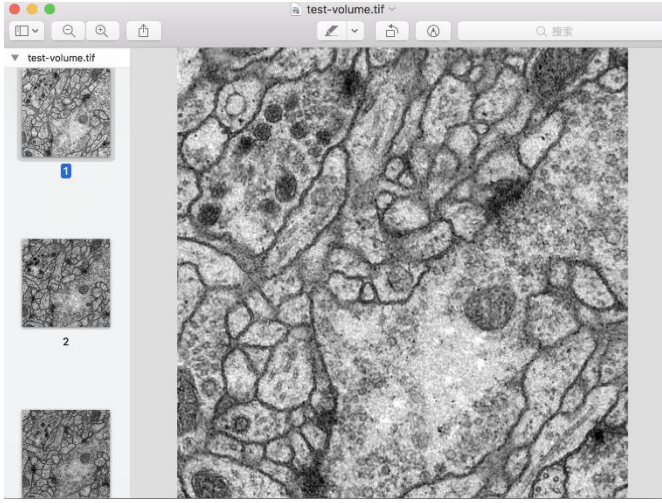
- 数据集大小：22.5M
- 训练集：15M，训练数据包含2个多页TIF文件，每个文件包含30张2D图像。train-volume.tif和train-labels.tif分别包含训练数据和对应的标签。
- 测试集：7.5M，测试数据包含1个多页TIF文件，该文件包含30张2D图像。test-volume.tif包含测试数据。



train-volume.tif



train-labels.tif



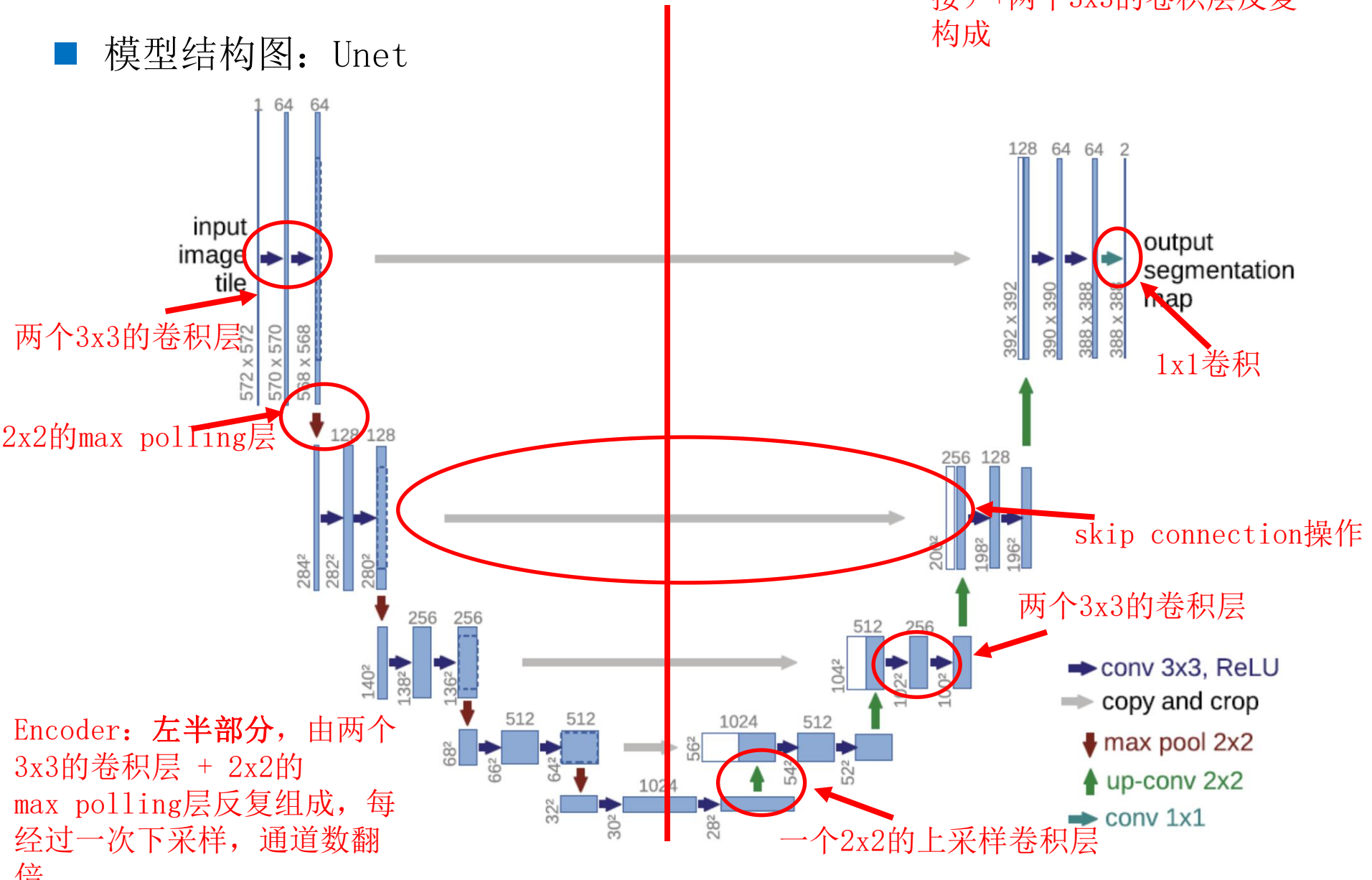
test-volume.tif

train-labels说明：分割要达到的目的是用白色标注细胞，用黑色标注细胞膜

四、模型选择——UNet

Decoder：**右半部分**，由一个2x2的上采样卷积层 + skip connection操作（特征图拼接）+两个3x3的卷积层反复构成

- 模型结构图：Unet



模型结构图说明

- Encoder：左半部分，由两个3x3的卷积层（ReLU）+2x2的max polling层（stride=2）反复组成，每经过一次下采样，通道数翻倍；
- Decoder：右半部分，由一个2x2的上采样卷积层（ReLU）+ skip connection（特征图feature map拼接操作）+2个3x3的卷积层（ReLU）反复构成；
- 最后一层通过一个1x1卷积将通道数变成期望的类别数，本案例中是两种类别，分别为细胞和细胞膜。

模型结构图说明

- UNet的encoder下采样4次，一共下采样16倍。对称地，其decoder也相应上采样4次，目的是将encoder得到的高层次特征图恢复到原图片的分辨率。相比于FCN，UNet共进行了4次上采样，并在“U型” 的同一水平位置上使用了skip connection，而不是直接在高层次特征上进行监督和loss反传，这样就保证了最后恢复出来的特征图融合了更多的低层次特征。

UNet简介

- UNet网络出自论文《U-Net: Convolutional Networks for Biomedical Image Segmentation》
- UNet最早发表在2015的MICCAI上，目前的引用量已达上万次，足以见得其影响力。而后成为大多做医疗影像语义分割任务的基准模型，也启发了大量研究者去思考U型语义分割网络。而如今在自然影像理解方面，也有越来越多的语义分割和目标检测SOTA模型开始关注和使用U型结构，比如语义分割Discriminative Feature Network(DFN)(CVPR2018)，目标检测Feature Pyramid Networks for Object Detection(FPN)(CVPR 2017)等。

UNet的优点

- UNet网络的每个卷积层得到的特征图都会拼接到对应的上采样层，从而实现对每层特征图都有效使用到后续计算中。这样，同其他的一些网络结构比如FCN（全连接神经网络）比较，UNet避免了直接在高层次feature map中进行监督和loss计算，而是结合了低层次feature map中的特征，从而可以使得最终所得到的feature map中既包含了高层次的feature，也包含很多的低层次的feature，实现了不同层次下feature的融合，提高模型的结果精确度。

UNet的优点

- 医疗图像的数量相对于自然图像要少，获得有临床意义的标签的难度也更大，所以用于医疗图像的网络模型机构不宜太复杂/参数不宜过多，这样容易造成过拟合，影响模型的适用性。UNet的模型参数可以通过调整模型层数、每层的通道数来缩小，与其他模型比做到轻量级。

UNet的优点

- 由于医疗图像在临床使用中，既需要整幅图的信息帮助医生判断诸如位置、病变和正常位置的对比之类的全局信息，也需要某些特点区域或者位置的局部信息，所以对于医学图像来说，每个层次的特征图信息都很重要，都对于诊断结果是有帮助的，所以应当尽量考虑到多尺度下的有效信息，Unet就正好可以结合高层次和低层次的特征，增大了信息量。

五、PyTorch代码实现

- 代码文件的结构图如下：

```
unet
├── data
|   ├── test-volume.tif
|   ├── train-volume.tif
|   └── train-labels.tif
├── src
|   ├── config.py
|   ├── dataset.py
|   ├── loss.py
|   ├── utils.py
|   └── unet
|   |   ├── __init__.py
|   |   ├── unet_model.py
|   |   └── unet_parts.py
├── main.py
└── README.md
```

data文件夹下存放数据集

src文件夹下存放配置文件、依赖文件、损失函数定义文件、模型架构文件等

main.py是主函数启动文件

模型架构文件

- unet_model.py:

```
class UNet(nn.Module):
    def __init__(self, n_channels, n_classes):
        super(UNet, self).__init__()
        self.n_classes = n_classes
        self.inc = double_conv(ch_in=n_channels, ch_out=64)
        self.down1 = down(ch_in=64, ch_out=128)
        self.down2 = down(ch_in=128, ch_out=256)
        self.down3 = down(ch_in=256, ch_out=512)
        self.down4 = down(ch_in=512, ch_out=1024)

        self.up1 = up(ch_in=1024, ch_out=512)
        self.up2 = up(ch_in=512, ch_out=256)
        self.up3 = up(ch_in=256, ch_out=128)
        self.up4 = up(ch_in=128, ch_out=64)

        self.Conv_1x1 = nn.Conv2d(64, n_classes, kernel_size=1, stride=1, padding=0)
```

可以对照前面的UNet模型结构图，down1~down4是左侧encoder部分，up1~up4是右侧decoder部分

定义训练过程

- main.py：

设置训练参数，构建网络、损失函数、优化器等，然后一次读取batch_size的数据对网络进行训练，在固定epoch内保存训练好的模型。

```
lr = 0.001 # 初始学习率
batch_size = 32
train_epoch = 30
num_classes = 4 # 分割的类别数
num_channels = 3 # 输入图像的通道数

image_train = './data/train-volume.tif'
label_train = './data/train-labels.tif'
image_test = './data/test-volume.tif'
save_path = './results/'
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

定义训练过程

- main.py：

```python
def train():
    net = UNet(n_channels=num_channels, n_classes=num_classes).to(device)
    # 创建dataset，前百分之八十作为训练数据
    data_train = ImageDataset(image_train, label_train, is_Train = True, train_rate = 0.8)
    # 根据dataset创建dataloader，可设置多线程读取
    data_train_loader = DataLoader(data_train,batch_size=batch_size)
    criterion = nn.CrossEntropyLoss() # 定义损失函数
    # adam优化器具有两个参数beta_1 和beta_2
    optimizer = optim.Adam(net.parameters(), lr = lr, betas=(0.9,0.99))
    # 指数学习率衰减，lr*gamma**epoch
    scheduler = lr_scheduler.ExponentialLR(optimizer, gamma=0.99)
    net.train() # 设置为训练状态
    for epoch in range(train_epoch):
        for ii, (images, labels) in enumerate(data_train_loader):
            images = images.to(device)
            labels = labels.to(device)
            optimizer.zero_grad() # 每次计算梯度前，都需要将梯度清零
            output = net(images)
            loss = criterion(output, labels.long()) # 计算损失函数
            loss.backward() # 反向传播计算梯度
            optimizer.step() # 更新网络参数
            scheduler.step() # 更新学习率

            if ii % 5 == 0: # 每5个batch打印一次信息
                print('Train - Epoch %d, Batch: %d, Loss: %f, lr: %f'
                      % (epoch, ii, loss.detach().cpu().item(), scheduler.get_last_lr()[0]))
        if (epoch + 1) % 5 == 0: # 每5个epoch保存一次模型参数
            torch.save(net.state_dict(), save_path+'checkpoint_%d.pth' % (epoch))
```

定义模型验证

- main.py：

```python
def test(epoch):
    net = UNet(n_channels=num_channels, n_classes=num_classes).to(device)
    net.load_state_dict(torch.load(save_path+'checkpoint_%d.pth' % (epoch))) # 加载网络模型
    # 创建dataset，后百分之八十作为测试数据
    data_test = ImageDataset(image_train, label_train, is_Train = False, train_rate = 0.8)
    data_test_loader = DataLoader(data_test,batch_size=batch_size)
    criterion = nn.CrossEntropyLoss()
    net.eval() # 设置为测试模式
    avg_loss = 0.0
    avg_dice = 0.0
    with torch.no_grad():
        for i, (images, labels) in enumerate(data_test_loader):
            images = images.to(device)
            labels = labels.to(device)
            output = net(images)
            loss = criterion(output, labels.long())
            avg_loss += loss.sum()
            # 将概率图转为分割类别，值为几就表示属于第几类
            out = torch.argmax(torch.softmax(output, dim=1), dim=1)
            dice = dice_coeff(out, labels, num_classes)
            avg_dice += dice
            print('Test Loss: %f, Dice Score: %f' % (loss.cpu().item(), dice.cpu().item()/batch_size))
    avg_loss /= len(data_test)
    avg_dice /= len(data_test)
    print('Test Avg. Loss: %f, Dice Score: %f' % (avg_loss.cpu().item(), avg_dice.cpu().item()))
```

加载模型参数到网络中，使用test函数进行模型验证。验证指标使用Dice coefficient，是一种集合相似度度量函数，通常用于计算两个样本的相似度，广泛应用于医学图像分割领域以度量分割准确率。

Dice coefficient的计算

- main.py:

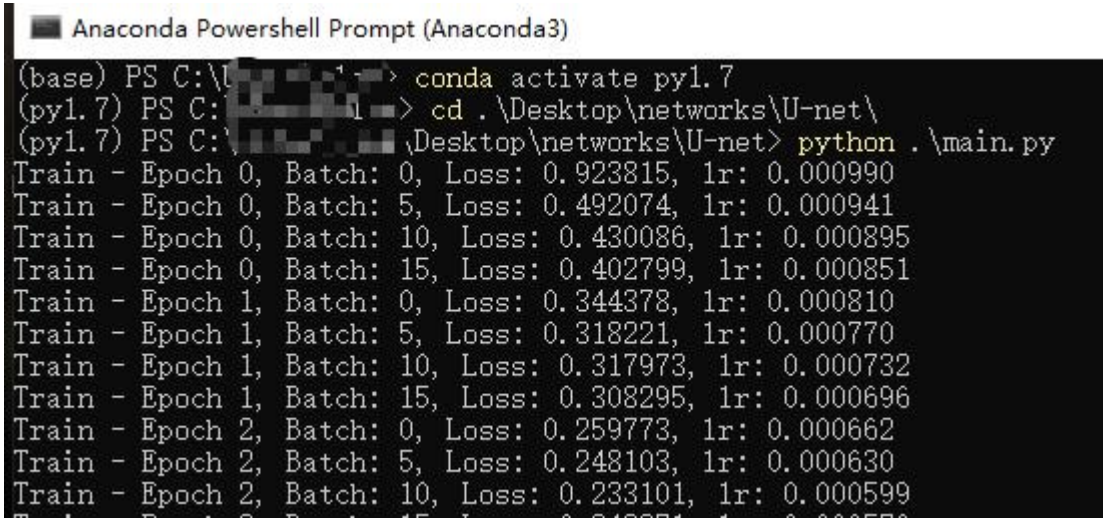
```
def dice_coeff(pred_masks, labels, num_class):
    """ computational formula:
        dice = (2 * (pred ∩ gt)) / (pred ∪ gt)
    """
    dice_score = 0.0
    for i in range(num_classes):
        pred = (pred_masks == i)
        pred = pred.view(pred_masks.shape[0],-1)
        label = (labels == i)
        label = label.view(pred_masks.shape[0],-1)
        intersection = (pred * label).sum(1)
        unionset = pred.sum(1) + label.sum(1)
        dice_score += 2 * intersection / unionset
    dice_score /= num_classes
    return dice_score.sum()
```

有关Dice coefficient 的计算函数。

Dice系数: $\frac{2(A \cap B)}{A + B}$

六、使用PyTorch环境进行训练

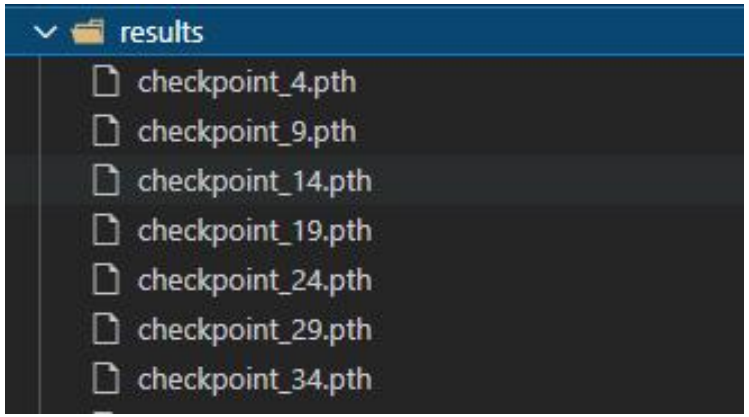
- Step 1：打开Anaconda终端，进入Pytorch环境
  - 执行conda activate pytorch
  - cd 到对应文件目录
  - 运行python main.py开始训练
  - Anaconda安装和Pytorch环境搭建参考附录



使用PyTorch环境进行训练

- Step 2：等待训练完成，训练过程中可以查看训练日志输出
  - 训练好的模型将保存在results目录下

```
Train - Epoch 0, Batch: 0, Loss: 0.640121, lr: 0.000990
Train - Epoch 0, Batch: 1, Loss: 0.578598, lr: 0.000980
Train - Epoch 0, Batch: 2, Loss: 0.524517, lr: 0.000970
Train - Epoch 0, Batch: 3, Loss: 0.460553, lr: 0.000961
Train - Epoch 0, Batch: 4, Loss: 0.412805, lr: 0.000951
Train - Epoch 0, Batch: 5, Loss: 0.380913, lr: 0.000941
Train - Epoch 0, Batch: 6, Loss: 0.385873, lr: 0.000932
Train - Epoch 0, Batch: 7, Loss: 0.376006, lr: 0.000923
Train - Epoch 0, Batch: 8, Loss: 0.348089, lr: 0.000914
Train - Epoch 0, Batch: 9, Loss: 0.331315, lr: 0.000904
Train - Epoch 1, Batch: 0, Loss: 0.330081, lr: 0.000895
Train - Epoch 1, Batch: 1, Loss: 0.368050, lr: 0.000886
Train - Epoch 1, Batch: 2, Loss: 0.323452, lr: 0.000878
Train - Epoch 1, Batch: 3, Loss: 0.328108, lr: 0.000869
Train - Epoch 1, Batch: 4, Loss: 0.322338, lr: 0.000860
Train - Epoch 1, Batch: 5, Loss: 0.293669, lr: 0.000851
```



使用PyTorch环境进行训练

- Step 3：修改main.py文件，调用test函数对模型进行测试

```
if __name__ == '__main__':
    # train()
    test(29) # test函数需要传入加载的模型训练的epoch
```

```
Test Loss: 0.119871, Dice Score: 0.894314
Test Loss: 0.146983, Dice Score: 0.869568
Test Loss: 0.162600, Dice Score: 0.848153
Test Loss: 0.134744, Dice Score: 0.890178
Test Loss: 0.156636, Dice Score: 0.856700
Test Loss: 0.117833, Dice Score: 0.884439
Test Loss: 0.181808, Dice Score: 0.814109
Test Loss: 0.127640, Dice Score: 0.898950
Test Loss: 0.145557, Dice Score: 0.879263
Test Loss: 0.118392, Dice Score: 0.896236
```

七、案例总结

```
Test Loss: 0.128263, Dice Score: 0.881150
Test Loss: 0.155010, Dice Score: 0.864061
Test Loss: 0.180119, Dice Score: 0.826713
Test Loss: 0.138773, Dice Score: 0.886308
Test Loss: 0.144473, Dice Score: 0.862572
Test Loss: 0.152875, Dice Score: 0.851341
Test Loss: 0.201741, Dice Score: 0.837899
Test Loss: 0.112302, Dice Score: 0.905401
Test Loss: 0.112326, Dice Score: 0.902401
Test Loss: 0.114496, Dice Score: 0.890210
Test Avg. Loss: 0.072019, Dice Score: 0.870806
```

本实验展示了如何使用Pytorch进行图片分割任务的训练和验证，以及开发和训练UNet模型。通过对UNet模型做几代的训练，然后使用训练后的UNet模型对数据集进行评估验证，dice系数大于0.87。即UNet模型学习到了如何进行图片分割。

搭建神经网络进行深度学习流程总结

1. 准备数据集
   - •包括数据预处理，以及构建读取数据的Dataset类
2. 搭建网络模型
   - •包括定义网络结构和前向传播过程
3. 搭建网络模型
   - •根据训练任务，选择合适的损失函数、优化器和学习率等
4. 读取训练数据，根据损失函数计算梯度并对网络参数进行优化
   - •训练开始之前，需要将网络设置为训练状态（net.train()），并将梯度清零（optimizer.zero_grad()）。一次读取batch_size的数据，前向传播得到输出，根据输出结果和损失函数计算损失，反向传播计算梯度（loss.backward()），然后根据设置好的优化策略来优化网络参数（opitimizer.step()）

搭建神经网络进行深度学习流程总结

5. 打印日志信息，保存模型

   •在控制台或以可视化的方式显示网络训练过程中的损失和其他评价指标，每训练一段时间，就保存一下网络的模型参数

   •当网络训练到合适阶段时，即可停止训练，避免过拟合

6. 读取模型参数和测试数据，对模型进行评估

   •测试过程，首先将网络设置为评估状态（net.eval()），由于不需要对网络参数进行优化，所以不需要计算梯度信息（with torch.no_grad(): ）

附录：Jupyter Notebook使用教程

- Jupyter Notebook是一个交互式笔记本，支持运行40多种编程语言。在开始使用notebook之前，需要先安装该库：
  - 在命令行中执行pip install jupyter来安装；
  - 安装后在anaconda的菜单里打开jupyter notebook，或通过命令行输入jupyter notebook或jupyter-notebook运行。
  - jupyter notebook会在浏览器中打开，在浏览器的地址栏会显示：http://localhost:8888/tree
  - 一次打开多个jupyter notebook，端口号会依次递增8889，8890依次递增。

附录：Jupyter Notebook使用教程

- 设置密码登入jupyter
  - password ：给某一个打开的jupyter notebook 服务设置密码，后面直接输入所要添加的密码即可。

```
(base) C:\Users\AIserver>jupyter notebook password
Enter password:
Verify password:
[NotebookPasswordApp] Wrote hashed password to C:\Users\AIserver\.jupyter\jupyter_notebook_config.json
```

附录：Jupyter Notebook使用教程

- 常见jupyter notebook命令
  - 查看jupyter notebook相关帮助

    jupyter-notebook --help

    jupyter-notebook --help-all （会显示更详细的信息）

    jupyter-notebook -h

  - jupyter notebook常用的子命令——subcommand

    list ：列出当前的所打开的jupyter notebook的一些信息

    stop：关闭所给定的端口号的那一个jupyter

```
(base) C:\Users\AIserver>jupyter notebook list
Currently running servers:
http://localhost:8888/ :: C:\Users\AIserver

(base) C:\Users\AIserver>jupyter notebook stop 8888
Shutting down server on port 8888 ...
```

附录：Jupyter Notebook使用教程

- 查看并修改配置文件
  - 在cmd中使用如下命令：jupyter-notebook --generate-config

```
(base) C:\Users\AIserver>jupyter notebook --generate-config
Writing default config to: C:\Users\AIserver\.jupyter\jupyter_notebook_config.py
```

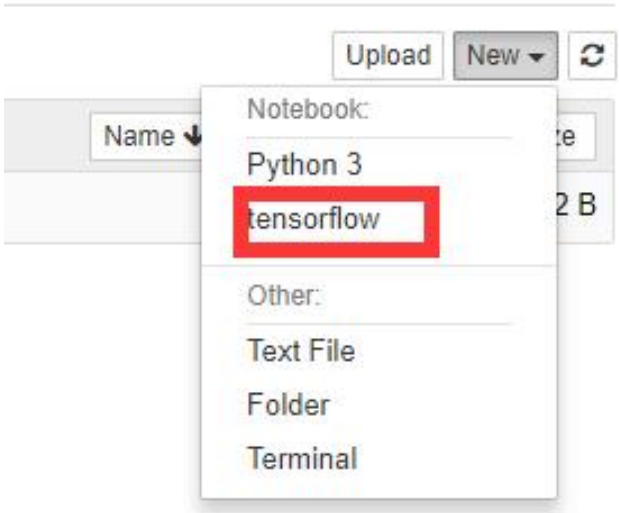
  - 打开配置文件，设置jupyter目录后保存并关闭。（文件夹需要提前创建好）

```
## The directory to use for notebooks and kernels.
#c.NotebookApp.notebook_dir = 'D:\myjupyter'
```

```
(base) C:\Users\AIserver>jupyter notebook
[I 17:54:17.256 NotebookApp] JupyterLab extension loaded from D:\anaconda3\lib\site-packages\jupyterlab
[I 17:54:17.256 NotebookApp] JupyterLab application directory is D:\anaconda3\share\jupyter\lab
[I 17:54:17.344 NotebookApp] Serving notebooks from local directory: D:\myjupyter
[I 17:54:17.344 NotebookApp] The Jupyter Notebook is running at:
[I 17:54:17.345 NotebookApp] http://localhost:8888/
[I 17:54:17.345 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

附录：Jupyter Notebook使用教程

- jupyter运行环境配置--python运行环境为例
  - 创建环境conda create -n tensorflow(环境名称)
  - 安装ipykernel conda install ipykernel
  - 激活相应环境 activate tensorflow
  - python -m ipykernel install --user --name tensorflow --display-name 内核名称



End of this lecture.

Thanks !