人工智能实验报告 第十三周

姓名:丁晓琪 学号:22336057

人工智能实验报告 第十三周
一.实验题目
二.实验内容
1.算法原理（CNN网络搭建和训练）
a.卷积层
Convolution:
Max Pooling:
b.全连接层
c.总体网络结构
d.网络训练:
2.关键代码展示
a.训练集和测试集的提取与数据化
b.CNN网络搭建
c.训练和测试
三.实验结果及分析
1.实验结果展示
2.评测指标展示及分析
四.参考资料(可选)

一.实验题目

利用pytorch框架搭建神经网络实现中药图片分类，其中中药图片数据分为训练集`train`和测试集`test`，训练集仅用于网络训练阶段，测试集仅用于模型的性能测试阶段。训练集和测试集均包含五种不同类型的中药图片：`baihe`、`dangshen`、`gouqi`、`huaihua`、`jinyinhua`。请合理设计神经网络架构，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的loss曲线、准确率曲线。

二.实验内容

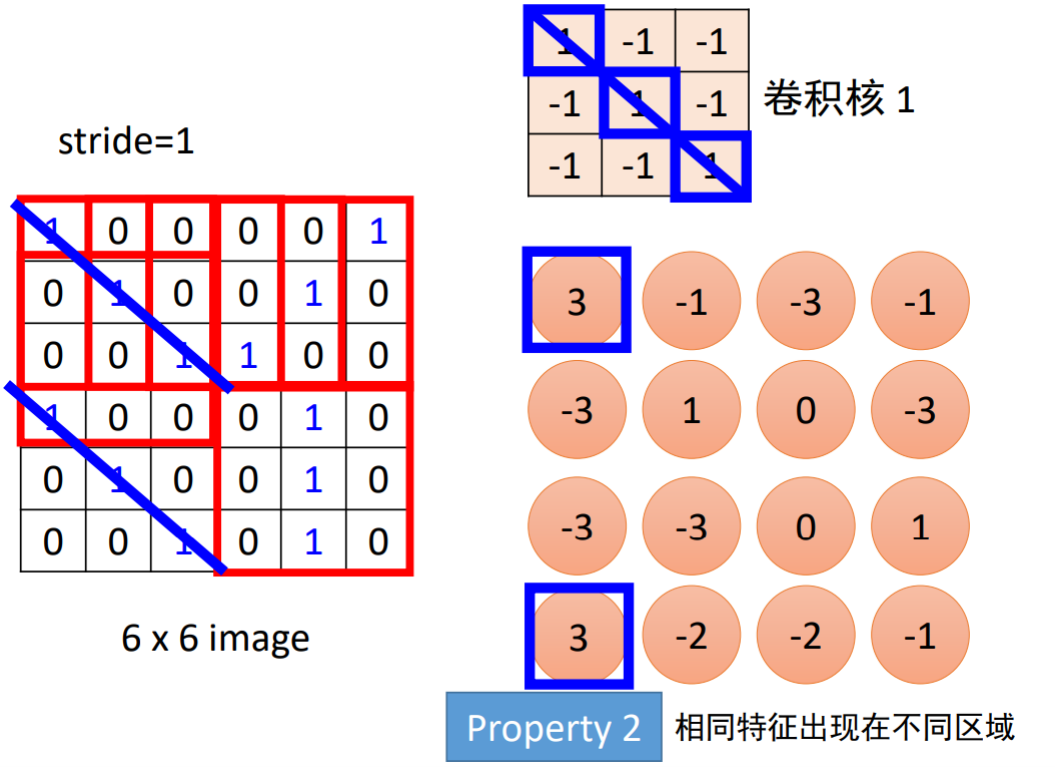
1.算法原理（CNN网络搭建和训练）

a.卷积层

Convolution:

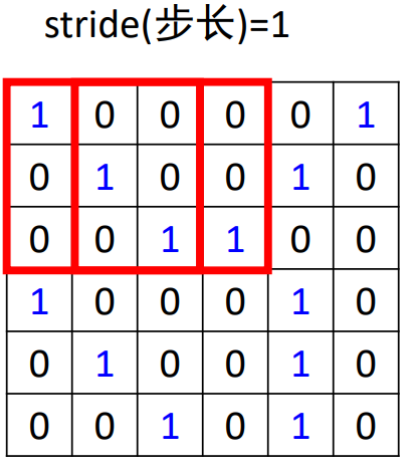
目的：提取图片中各个区域的特征

参数：卷积核(Kernel_size)：基于某些参数的特征可能比整张图片小的原则，所以卷积核矩阵的大小会比图片矩阵小；而且一张图片可能有多个特征，所以需要多个卷积核



实现：基于相同特征可能会出现再不同区域的原则，所以要对图片矩阵的各个部分与卷积核做卷积操作得到新的矩阵

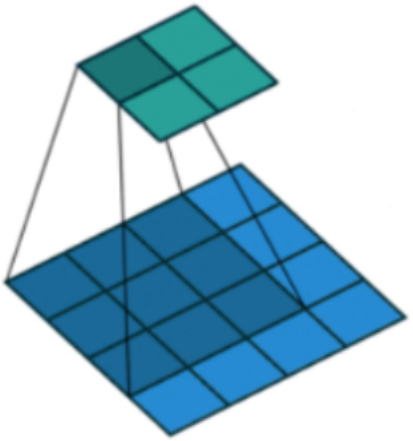
步长(stride)：在卷积操作中滑动卷积核的步幅，当步长较大时，卷积核在输入数据上滑动的速度会更快，这会导致输出特征图的尺寸减小。较大的步长可以减少模型的计算复杂度和内存消耗，但可能会丢失一些细节信息



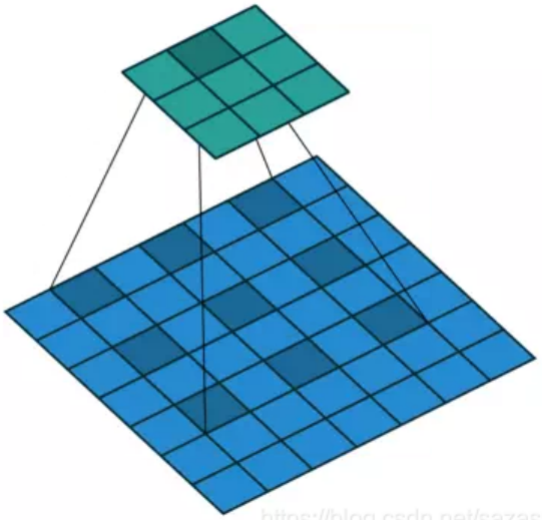
填充(Padding)：在输入图像的周围增加额外的像素值，以扩大输入图像的尺寸。由于在卷积操作中，卷积出来的特征矩阵的尺寸会缩小，为了更好地处理图像边缘信息，控制输出特征图的大小，需要Padding在图像边缘填充0或其他来调整

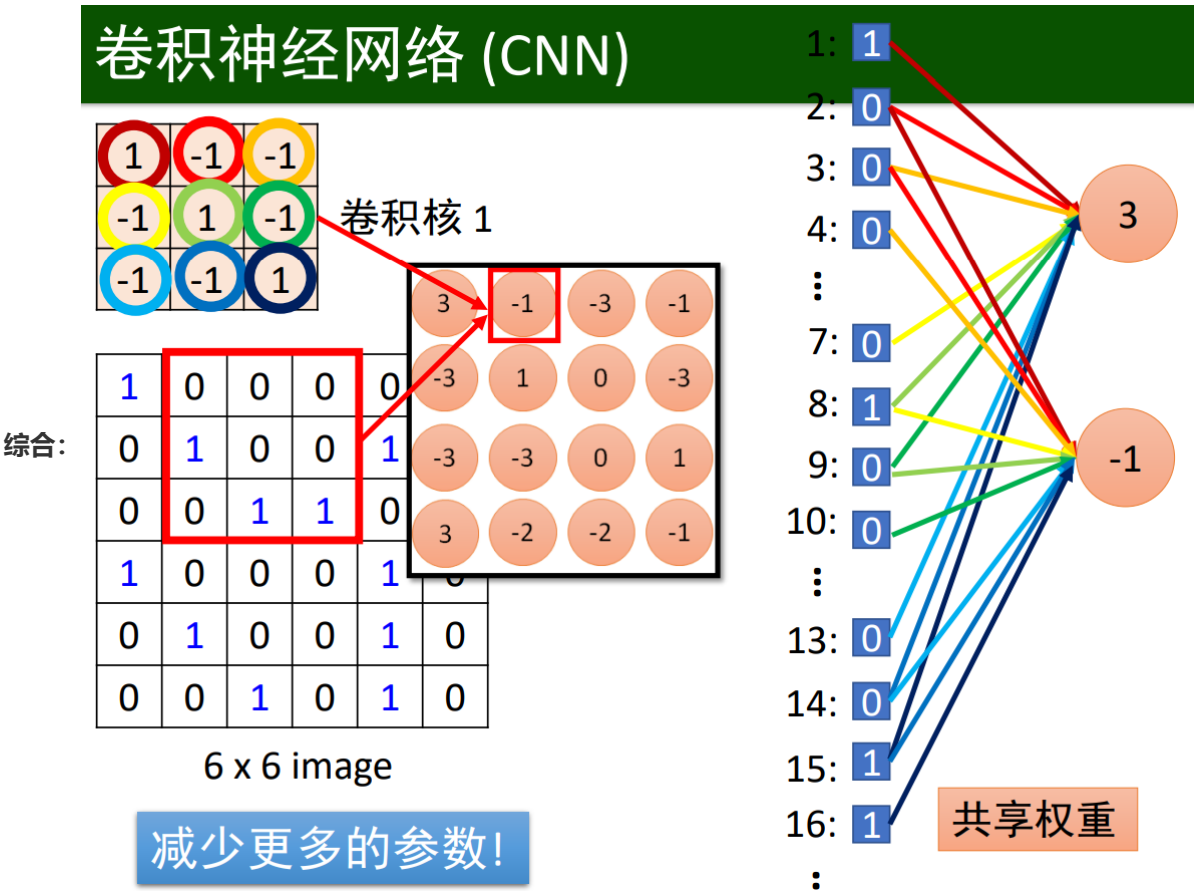
dilation：控制卷积操作中卷积核点的间距。单次计算时覆盖的面积（即感受域）由dilation=0时的`3*3=9`变为了dilation=1时的`5*5=25`。
在增加了感受域的同时却没有增加计算量，保留了更多的细节信息，对图像还原的精度有明显的提升。

（1）dilation=0的话，效果如图：



（2）dilation=1，那么效果如图：
称为扩张卷积（也叫空洞卷积）



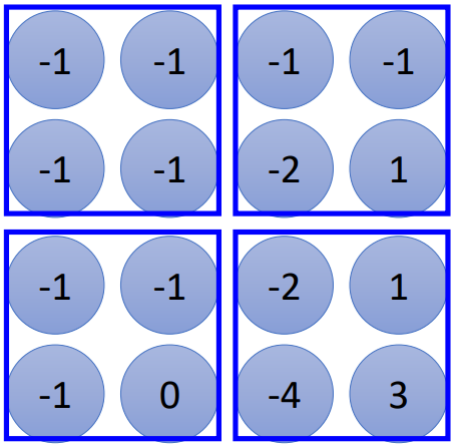


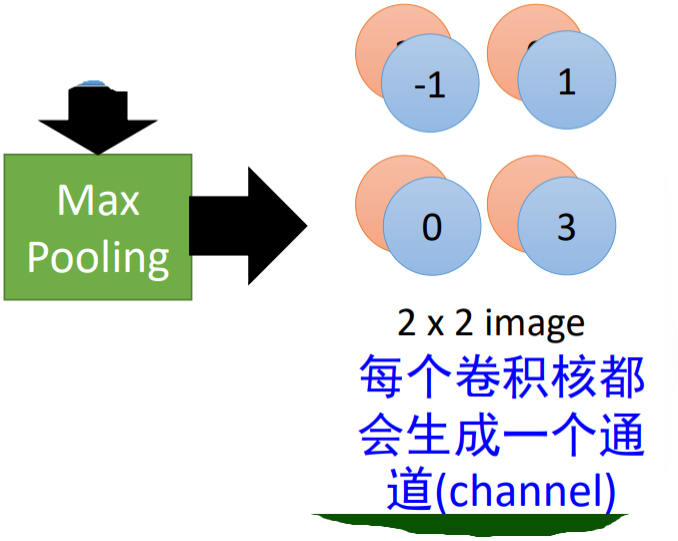
Max Pooling:

目的：二次采样，使用更少的参数来处理图片信息，同时保留图像特征

参数：`pool_size`:池化窗口大小，`strides`:池化操作的移动步长

实现：每个池化窗口的最大值会被提取出来，形成新的特征图





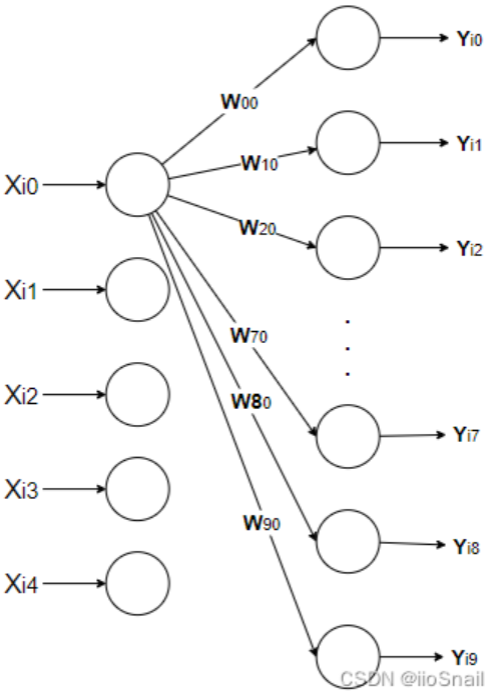
b.全连接层

输入：经过卷积层提取特征和减少参数后的输入

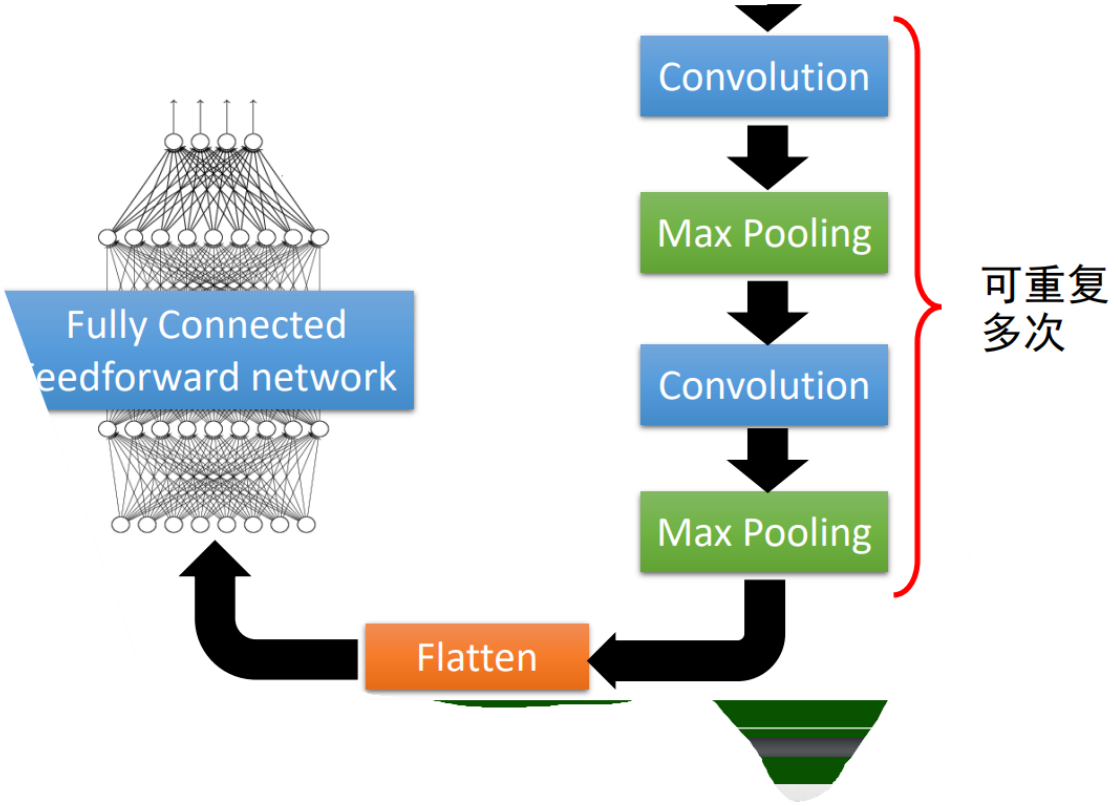
输出：对输入的线性变化，在实验中是要对重要图片分类，所以这里的输出是有5个元素的向量，表征各种种类的概率（未归一化）。输出的数值越大，表示属于对应种类的概率越大

$$Y_{no} = X_{ni}W_{io} + b$$

结构：



c.总体网络结构



输入：由于是彩色图片，所以会有RGB三个通道，高度宽度都设置为200，输入为$3 \times 200 \times 200$

conv1(卷积层1)：

设置16个卷积核，输出时将为16个特征图即为16通道；

卷积核大小为5×5，并且通过设置`Padding`来保证经过卷积操作后的宽度和高度不变；

激活函数：ReLU

$$f(x) = max(0, x)$$

优点：缓解梯度消失问题，一定程度上避免过拟合

池化操作：把池化窗口的大小设置为2，则将卷积出来的特征图的宽和高压缩一半

输出：$16 \times 100 \times 100$

conv2(卷积层2)：

设置32个卷积核，输出时将为32个特征图即为32通道；

卷积核大小为5×5，并且通过设置`Padding`来保证经过卷积操作后的宽度和高度不变；

激活函数：ReLU

池化操作：把池化窗口的大小设置为2，则将卷积出来的特征图的宽和高压缩一半

输出：$32 \times 50 \times 50$

conv3(卷积层3)：

设置32个卷积核，输出时将为32个特征图即为32通道；

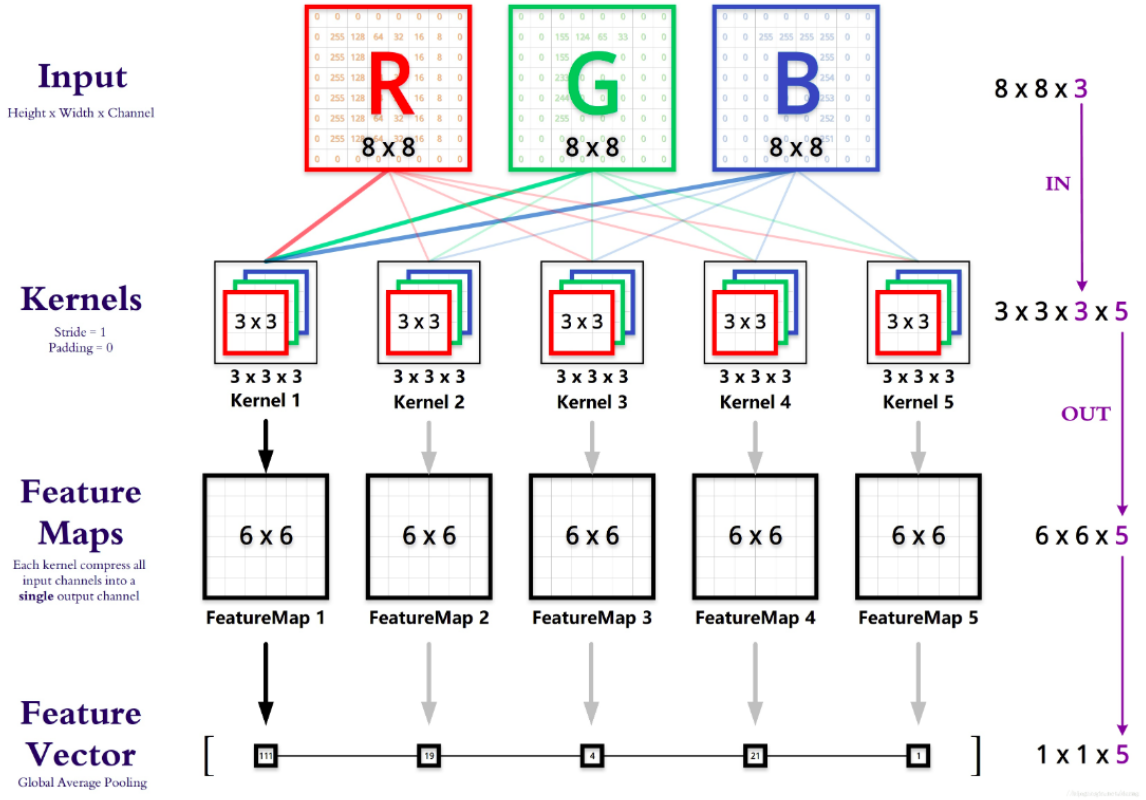
卷积核大小为5×5，并且通过设置`Padding`来保证经过卷积操作后的宽度和高度不变；

激活函数：ReLU

池化操作：把池化窗口的大小设置为2，则将卷积出来的特征图的宽和高压缩一半

输出：$32 \times 25 \times 25$

以下图片为经过卷积的示意图（数据不同，但是结构相似，并且最后的池化操作本次实验用的是最大池化而不是平均池化）



ouput（全连接层）：

输入：$32 \times 25 \times 25$

输出：5 （与图片属于5个种类的概率成正比）

d.网络训练:

- 前向计算求得Loss
- 将Loss反向传播到各级网络上
- 对各级网络参数基于Loss的梯度进行梯度更新

2.关键代码展示

a.训练集和测试集的提取与数据化

（1）定义转换，预处理图像

```
transform = transforms.Compose([
    transforms.Resize((200, 200)),  # 调整图像大小为200x200像素
    transforms.ToTensor(),  # 将图像转换为torch.Tensor
])
```

(2) `datasets.ImageFolder(root= ,transform=)`:

从文件夹中加载图像数据，文件夹的结构要为每个标签一个子文件夹，样本按标签存放在对应文件夹下

属性:

- `classes`：一个列表，包含数据集中所有类别的名称（即文件夹的名称）。
- `class_to_idx`：一个字典，将类别名称映射到整数索引。这个字典在内部用于将标签从字符串转换为整数。
- `imgs`：一个列表，包含数据集中所有图像的元组（路径，类别）。这主要用于调试和内部使用。

(3) `DataLoader(train_dataset, batch_size=, shuffle=)`:

参数:

- `batch_size`： 表现把数据集分成多少个批次。由于训练的数据一般很多，为了达到更好的训练效果，会将训练数据分为多个批次，逐次输入训练
- `shuffle`：是否将样本数据打乱

属性:

`for step, (b_x, b_y) in enumerate(train_loader)`：迭代每个批次，获得当前批次的索引`step`。并且从每个批次中解包出包含一批图像数据的张量`b_x`和包含对应标签的张量`b_y`

(4) 对测试图像集的提取

由于测试图像集并没有按标签存放在子文件夹下，所以先把测试图像集转换成按标签存放在子文件的形式。然后把它加载到迭代数据加载器中，批次大小为整个测试图像集的大小，这样能方便对测试图像集的测试

```
source_dir="./test"
destination_dir="./test_plus"
os.makedirs("./test_plus",False)
os.makedirs("./test_plus/baihe")
os.makedirs("./test_plus/dangshen")
os.makedirs("./test_plus/gouqi")
os.makedirs("./test_plus/huaihua")
os.makedirs("./test_plus/jinyinhua")
##读取test里面的文件并且转移到test_plus中按标签存放到子文件夹中
for filename in os.listdir(source_dir):
    #构建完整的source路径
    file_path=os.path.join(source_dir,filename)
    if "baihe" in filename:
        destination_path="./test_plus/baihe"
    elif "dangshen" in filename:
         destination_path="./test_plus/dangshen"
    elif "gouqi" in filename:
         destination_path="./test_plus/gouqi"
    elif "huaihua" in filename:
         destination_path="./test_plus/huaihua"
    elif "jinyinhua" in filename:
         destination_path="./test_plus/jinyinhua"
    destination_path=os.path.join(destination_path,filename)
```

```
    shutil.move(file_path, destination_path)

##转换图片为数据##dataloader自动化处理样本
transform=transforms.Compose([
    transforms.Resize((200,200)), #图片大小调整为200*200
    transforms.ToTensor()]  #转化为张量
)  #图片预处理操作步骤集合
train_dataset=datasets.ImageFolder(root="./train",transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_dataset = datasets.ImageFolder(root="./test_plus", transform=transform)

# 定义数据加载器（这里不使用 batch_size，因为希望一次处理整个测试集
test_loader = DataLoader(dataset=test_dataset, batch_size=10, shuffle=False)
```

b.CNN网络搭建

```
##CNN结构
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()#类继承的初始化
        self.conv1=nn.Sequential( #输入 3*200*200
            nn.Conv2d(  #卷积操作在2维上操作，宽和高，每个特征方程已经结合了三个通道了
                in_channels=3 , #彩色通道RGB
                out_channels=16, #16个滤波器，卷积核
                kernel_size=5, #卷积核大小为5*5
                stride=1 ,#卷积窗口移动步长为1
                padding=2 #填充0保证行列数不变
            ),#输出 16*200*200
            nn.ReLU(),#激活函数 RReLU
            nn.MaxPool2d(2),#最大池化层 2*2里面挑一个最大的 输出16*100*100
        )
        self.conv2=nn.Sequential(
            nn.Conv2d(16,32,5,1,2),
            nn.ReLU(),
            nn.MaxPool2d(2),#输出 32*50*50
        )
        self.conv3=nn.Sequential(
            nn.Conv2d(32,32,5,1,2),
            nn.ReLU(),
            nn.MaxPool2d(2),#输出 32*25*25
        )
        self.output=nn.Linear(32*25*25,5)

    def forward(self,x): #每次输入会自动调用
        x=self.conv1(x)
        x=self.conv2(x)
        x=self.conv3(x)
        x=x.view(x.size(0),-1)  #保留第一维度，剩下的都压成一维
        output=self.output(x)
        return output,x
```

c.训练和测试

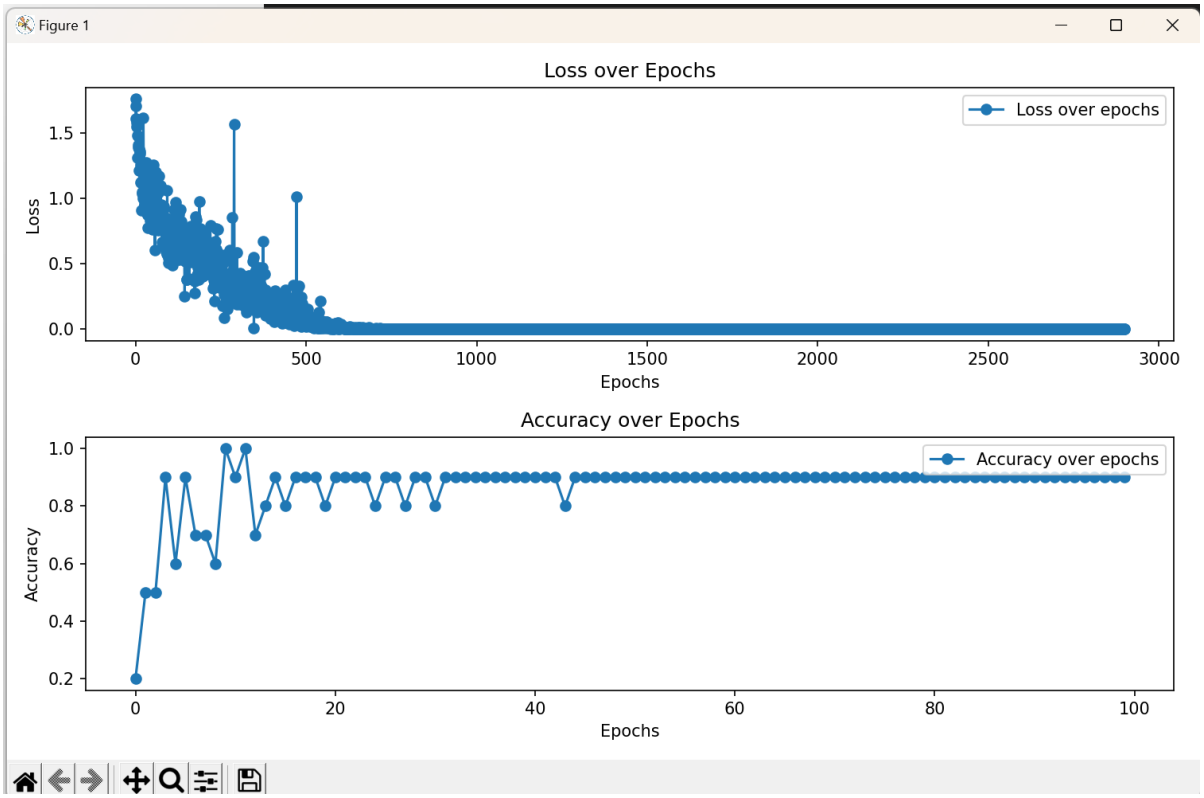
迭代训练`EPOCH`次，每次迭代中对所有批次进行训练，并且在测试集中测试模型的正确率

```
##实现网络优化，调参
device = torch.device("cuda")
cnn=CNN()
cnn.to(device)
optimizer=torch.optim.Adam(cnn.parameters(),lr=LR) #设置更新网络参数的优化器，lr为学习率
loss_func=nn.CrossEntropyLoss() #交叉熵,定义损失函数

for epoch in range(EPOCH): #每次迭代，EPOCH迭代次数
    for step,(b_x,b_y) in enumerate(train_loader):#step为批次索引，b_x为每一批次的输入，b_y为每一批次的标签
        # 将数据和标签移动到GPU
        b_x = b_x.to(device)
        b_y = b_y.to(device)
        output=cnn(b_x)[0]  #输入每一批次到cnn，自动调用forward
        loss=loss_func(output,b_y) #计算损失函数
        loss_rate.append(loss.item())
        optimizer.zero_grad() #清除梯度
        loss.backward() #损失函数的反向传播
        optimizer.step() #参数优化

        if step % 32 == 0: #一次迭代中所有批次训练完，测试测试集，打印准确率
            for step2,(test_x,test_y) in enumerate(test_loader):
                test_x = test_x.to(device)
                test_y = test_y.to(device)
                test_output, last_layer = cnn(test_x)
                pred_y = torch.max(test_output, 1)[1].data.detach().cpu().numpy()  #GPU数据移到cpu再移到numpy  #返回概率最大的索引
                accuracy = float((pred_y == test_y.data.detach().cpu().numpy()).astype(int).sum()) / float(test_y.size(0))
                accuracy_rate.append(accuracy)
                print('Epoch: ', epoch, '| train loss: %.4f' % loss.data.detach().cpu().numpy(), '| test accuracy: %.2f' % accuracy)
```

三.实验结果及分析

1.实验结果展示

参数：学习步长为0.001，迭代次数为100次，训练数据分为32个批次




```
Epoch:  86 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  87 | train loss: 0.0001 | test accuracy: 0.90
Epoch:  88 | train loss: 0.0001 | test accuracy: 0.90
Epoch:  89 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  90 | train loss: 0.0001 | test accuracy: 0.90
Epoch:  91 | train loss: 0.0001 | test accuracy: 0.90
Epoch:  92 | train loss: 0.0001 | test accuracy: 0.90
Epoch:  93 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  94 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  95 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  96 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  97 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  98 | train loss: 0.0000 | test accuracy: 0.90
Epoch:  99 | train loss: 0.0000 | test accuracy: 0.90
final_train_accuracy: 1.0 (correct= 902 )
```

2.评测指标展示及分析

损失率：在迭代过程中呈现下降趋势

在测试集中正确率：逐步上升，并且趋于稳定的0.9

在训练集中正确率：最后训练完后为1.0，存在过拟合的可能性

四.参考资料(可选)

https://github.com/MorvanZhou/PyTorch-Tutorial/blob/master/tutorial-contents/401_CNN.py