中山大学计算机学院 人工智能 本科生实验报告

课程名称：Artificial Intelligence

学号	姓名
22336327	庄云皓

一、 实验题目

利用pytorch框架搭建神经网络实现中药图片分类

实验要求:

搭建合适的网络框架，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

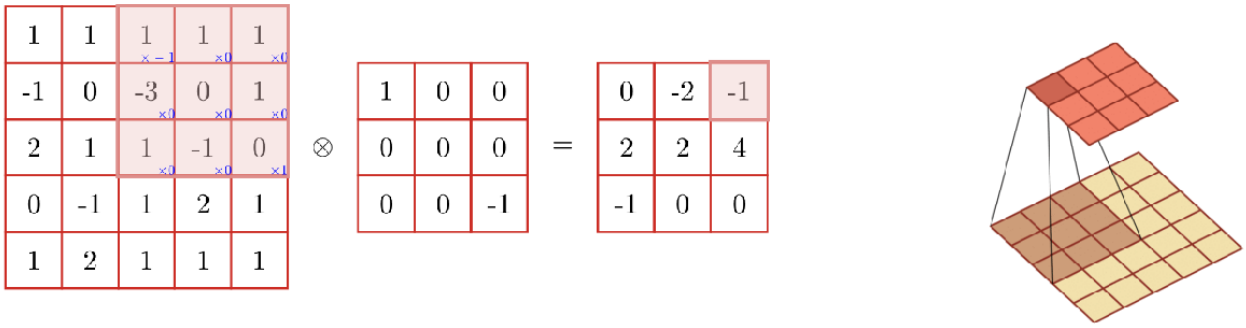
1. **算法原理**

   CNN简介

   一个卷积神经网络主要由以下5层组成：

   - 数据输入层/ Input layer
   - 卷积计算层/ CONV layer
   - ReLU激励层 / ReLU layer
   - 池化层 / Pooling layer
   - 全连接层 / FC layer

   二维卷积操作示意图：



   池化 汇聚函数

- 常用的汇聚函数有两种:
- (1)最大汇聚（Maximum Pooling或Max Pooling）：对于一个区域$A = R^d_{h,w}$选择这个区域内所有神经元的最大活性值作为这个区域的表示，即:

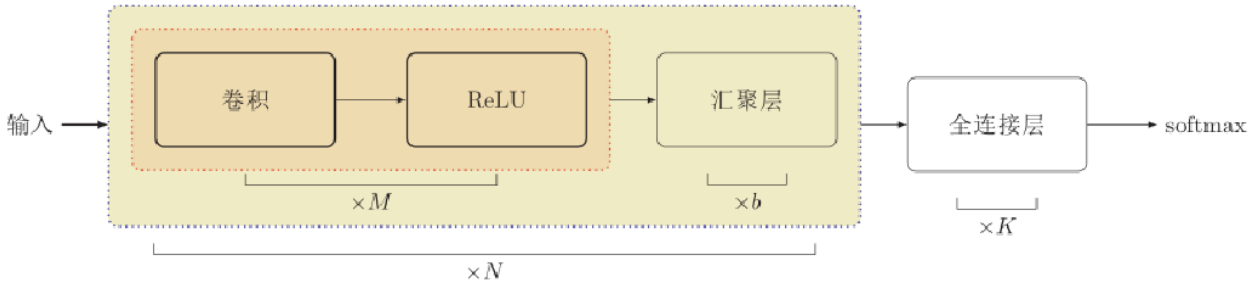
$$y^d_{h,w} = \max_{i \in R^d_{h,w}} x_i$$

其中，x_i为这个区域A内每个神经元的活性值

- (2)平均汇聚（Mean Pooling）：一般是取区域内所有神经元活性值的平均值，即:

$$y^d_{h,w} = \frac{1}{|R^d_{h,w}|} \sum_{i \in R^d_{h,w}} x_i$$

一个卷积块为连续C个卷积层和R个汇聚层（C通常设置为2 ~ 5，R为0或1）。一个卷积网络中可以堆叠B个连续的卷积块，然后接着K 个全连接层（B 的取值区间比较大，比如1 ~ 100或者更大；K一般为0 ~ 2）。



实验过程

1.1数据集介绍

- 训练集:

共902张图片，分为5类：百合，党参，枸杞，怀化，金银花。

每张图片通道数为3

- 测试集:

共10张图片

1.2数据预处理:

下面图像处理的过程将输入的图像先调整大小为(128, 128)，然后将其转换为张量形式，并对每个通道的像素值进行标准化处理，使其符合标准正态分布。这样处理后的图像可以更好地适应神经网络模型的训练和收敛过程。

transform中的Compose类允许我们将多个transform操作串联起来，形成一个完整的预处理流程

```
transform = transforms.Compose([
 transforms.Resize((128, 128)),
 transforms.RandomHorizontalFlip(),#以给定概率水平对称
 transforms.ToTensor(),#array类型为uint8  经过ToTensor() 后数值由 [0,255] 变为[0,1]，通过将每个数据除以255进行归一化。
 transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
 #别人的解答：数据如果分布在(0,1)之间，可能实际的bias，就是神经网络的输入b会比较大，而模型初始化时b=0的，这样会导致神经网络收敛比较慢，经过Normalize后，可以加快模型的收敛速度。
 ])
```

1.3构造自己的Dataset

首先，根据pytorch基础知识，写出Dataset类的基本框架：

```
class My_Dataset(Dataset):
   def __init__(self):
      pass
   def __len__(self):
      pass
   def __getitem__(self,idx):
      pass
```

- TestDataset

构造函数中有两个参数file_list和transform,获取图象的路径列表,和对图象的transform操作。

getitem函数打开图片，transorm,返回图片和标签，测试集不需要标签，这里的标签为0.

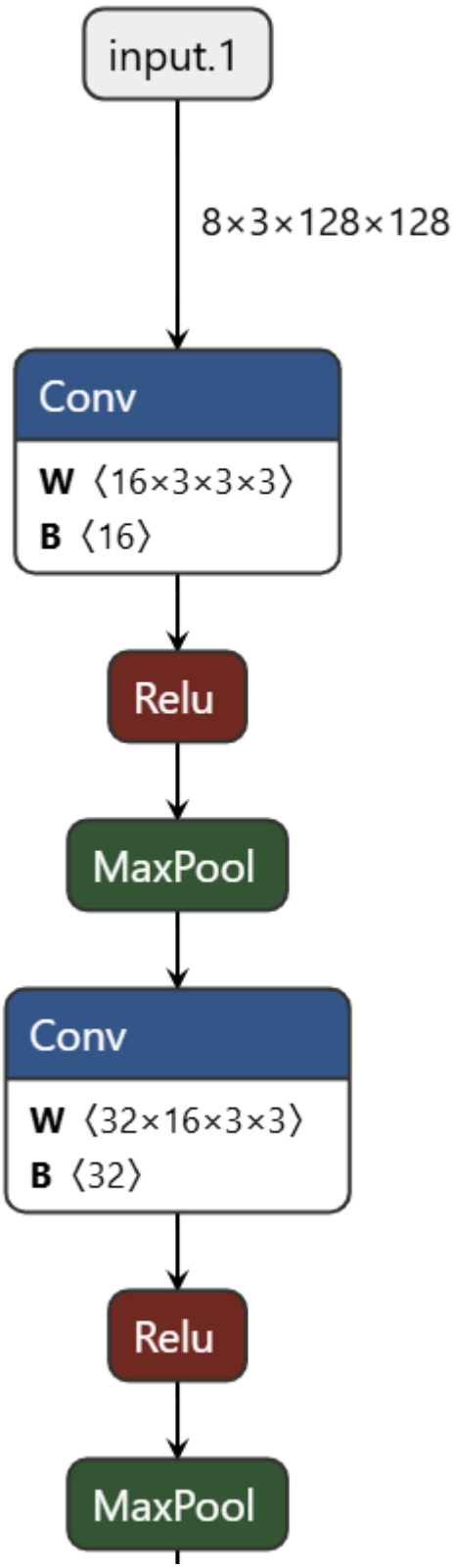
dataloader将自定义的的Dataset根据batch size大小、是否shuffle等封装成一个Batch Size大小的Tensor，用于后面的训练。

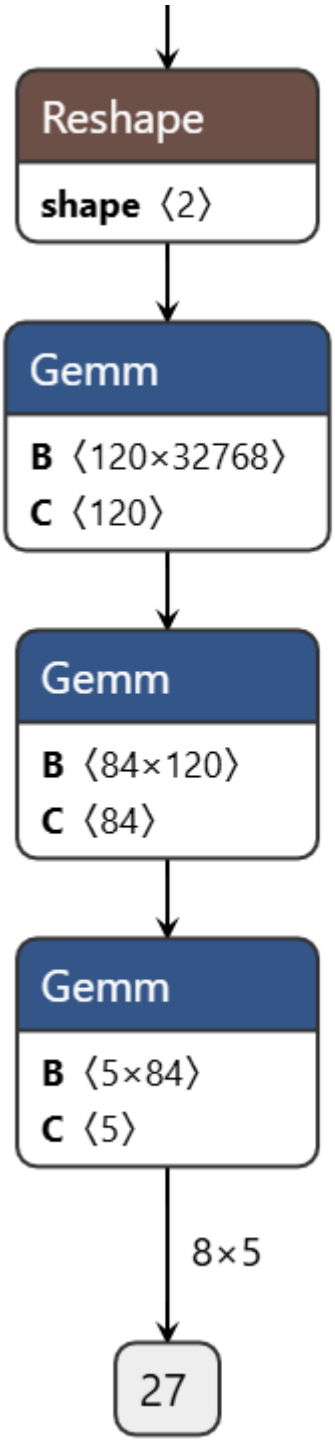
```
'''test_dataset'''
class TestDataset(Dataset):
   def __init__(self,file_list,transform=None):
      self.file_list = file_list
      self.transform = transform
   def __len__(self):
      return len(self.file_list)   # 返回列表元素的数目
   def __getitem__(self, idx):
      img_name = self.file_list[idx]
      #打开图片
      img = Image.open(img_name).convert("RGB")
      if self.transform:#如果没有transform的话就不变换
            img = self.transform(img) #trasform的参数是图象类型
      label = 0#测试集不需要label
      return img,label
test_data = TestDataset(test_list,transform=transform)
test_loader = DataLoader(test_data,batch_size=BATCH_SIZE,shuffle=False)
```

- Train_Dataset

```
'''train_datset'''
train_data = datasets.ImageFolder(train_dir,transform = transform)
train_loader = DataLoader(train_data, batch_size=BATCH_SIZE, shuffle=True)
```

1.4网络结构




Reshape
shape ⟨2⟩
Gemm
B ⟨120×32768⟩
C ⟨120⟩
Gemm
B ⟨84×120⟩
C ⟨84⟩
Gemm
B ⟨5×84⟩
C ⟨5⟩
8×5
27

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc): Sequential(
    (0): Linear(in_features=32768, out_features=120, bias=True)
    (1): Linear(in_features=120, out_features=84, bias=True)
    (2): Linear(in_features=84, out_features=5, bias=True)
  )
)
```

网络结构如上所示，采用了两个卷积层和一个全连接层，kernel大小均为3*3，采用ReLU为激活函数，输入维度为（b,3,128,128), 最后输出维度为（b,5)

```
class CNN(nn.Module):
 def __init__(self,num_classes=5):
      super(CNN, self).__init__()
      self.conv1 = nn.Sequential(         # input shape (1, 28, 28)
          nn.Conv2d(
              in_channels=3,              # input height
              out_channels=16,            # n_filters
              kernel_size=3,              # filter size
              stride=1,                   # filter movement/step
              padding=1,                  # if want same width and length of
this image after Conv2d, padding=(kernel_size-1)/2 if stride=1
          ),                              # output shape (16, 28, 28)
          nn.ReLU(),                      # activation
          nn.Dropout(0.5),
          nn.MaxPool2d(kernel_size=2),    # choose max value in 2x2 area,
output shape (16, 14, 14)
      )
      self.conv2 = nn.Sequential(         # input shape (16, 14, 14)
          nn.Conv2d(16, 32, 3, 1, 1),     # output shape (32, 14, 14)
          nn.ReLU(),                      # activation
          nn.Dropout(0.5),
          nn.MaxPool2d(2),                # output shape (32, 7, 7)
      )

      self.out = nn.Linear(32 * 32 *32, num_classes)   # fully connected
layer, output 10 classes

 def forward(self, x):
      x = self.conv1(x)
```

```
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to
(batch_size, 32 * 7 * 7)
        output = self.out(x)
        return output    # return x for visualization
```

1.5 模型训练与测试

```
def train():
 torch.cuda.empty_cache()
 torch.cuda.set_device(3)
 model = CNN().to(device)
 loss_fun = nn.CrossEntropyLoss()
 optimizer = optim.Adam(params=model.parameters(),lr=LR)#传入模型参数和学习率
 loss_list = []
 for i in range(EPOCH):
     for j,(batch_data,batch_label) in enumerate(train_loader):
         batch_data, batch_label = batch_data.cuda(),batch_label.cuda()#数据
进行.cuda()处理。就可以将内存中的数据复制到GPU的显存中去
         #梯度清零
         optimizer.zero_grad()
         #前向传播
         output = model(batch_data)
         #计算损失
         loss = loss_fun(output,batch_label)
         loss_list.append(loss.item())
         #反向传播
         loss.backward()
         #更新权重
         optimizer.step()
 plt.plot(loss_list)
```

```
def test(model):
model.eval()
correct = 0
total = 0
with torch.no_grad():#反向传播时就不会自动求导了，因此大大节约了显存
    for j,(batch_data,batch_label) in enumerate(test_loader):
        print(len(batch_data))
        batch_data, batch_label =
batch_data.to(device),batch_label.to(device)
        outputs = model(batch_data)
        labels = [train_data.class_to_idx[os.path.basename(f).rsplit('0', 1)
[0]] for f in test_list]
        labels = torch.tensor(labels).to(device)
        length = len(labels)
        #labels = [train_data.class_to_idx[os.path.basename(f).rsplit('0',
1)[0]] for f in test_list]
```

```
            temp_labels = labels[j*BATCH_SIZE:min((j+1)*BATCH_SIZE,length)]
            temp_len=min((j+1)*BATCH_SIZE,length)-j*BATCH_SIZE
            total += temp_len
            print("outputs",outputs)
            predict = torch.Tensor.argmax(outputs.data,axis=1)
            #print(np.argmax(a, axis=0))  #竖着比较，返回行号
            #print(np.argmax(a, axis=1))  #横着比较，返回列号
            print("tmp_labels",temp_labels)
            correct += (predict==temp_labels).sum()

    print("Test Accuracy %.2f%"%(correct/total))
```

2. **关键代码展示**

   见上一部分

3. **创新点&优化**

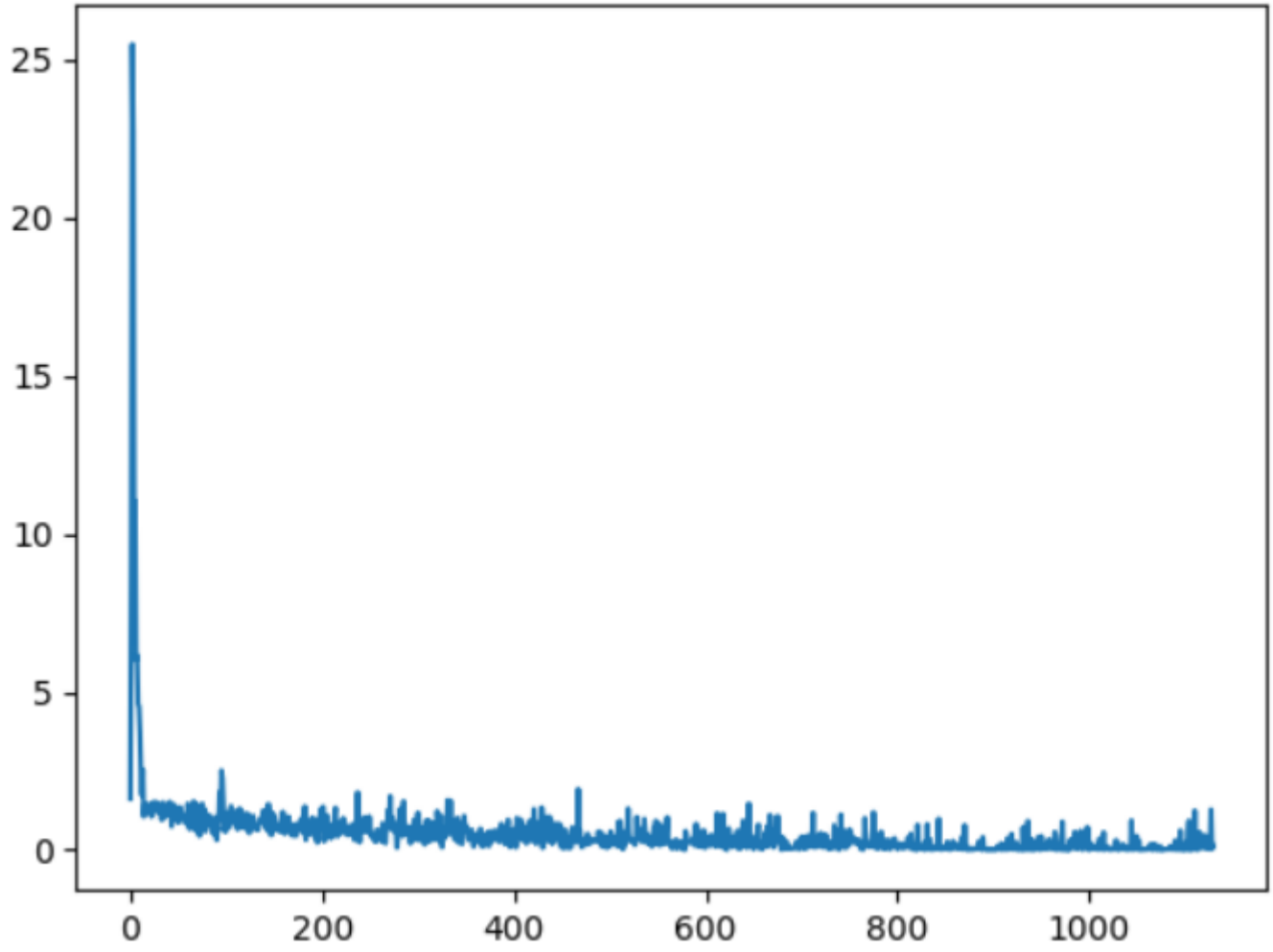
   自动调整学习率:Adam优化器会自动调整学习率，因此我们不需要手动进行调整。

三、 实验结果及分析

1. 实验结果展示示例

可直接复制图片粘贴在此处.test每个batch的size为8,共十张图片。

训练集的loss(采用交叉熵损失函数) 横坐标为batch数量，纵坐标为loss



2. 评测指标展示及分析

在测试集上的表现，测试集仍然按照batch_size=8进行，准确率达到了100%

output tenser每行最大值的下标就是类别，与tmp_labels比较可得准确率

```
8
outputs tensor([[ 1.1709,  1.6508, -0.1051, -0.0567, -2.3610],
        [ 1.8947,  0.1113, -0.6055, -0.6799, -0.2202],
        [-1.0645, -0.2436, -0.9937,  1.6749, -0.0372],
        [-2.6794, -1.2531,  9.8453, -1.1870, -7.3367],
        [-0.7099, -1.0279, -0.6006,  0.8405,  1.1300],
        [-0.4967, -1.1624, -1.7586,  1.9546,  0.9601],
        [-3.0231, -2.6076, -2.4438,  2.6807,  4.8665],
        [ 0.5757,  2.7887, -1.4024,  1.2335, -2.8800]], device='cuda:3')
tmp_labels tensor([1, 0, 3, 2, 4, 3, 4, 1], device='cuda:3')
2
outputs tensor([[ 8.7234e-01,  1.4273e-01, -9.8747e-01, -4.4075e-02,  3.6059e-01],
        [-3.5363e+00, -5.9320e-03,  1.0812e+01, -1.4192e+00, -8.3417e+00]],
       device='cuda:3')
tmp_labels tensor([0, 2], device='cuda:3')
Test Accuracy 100.00%
```

参数量：

The total number of parameters: 3947957 The parameters of Model CNN: 3.947957M

优化了网络结构，没有保存优化前的结果所以没有办法展示

四、 参考资料

【PyTorch】PyTorch搭建神经网络处理图片分类的完整代码-CSDN博客

在 jupyter notebook 中如何指定使用指定 GPU 进行训练？_jupyter notebook中怎么指定用gpu训练-CSDN博客

SYSU_AI_lab/lab11_pytorch中药分类/[CS]_20337025_cuicanming/code/CNN_CM.py at main · 91Mrcui/SYSU_AI_lab (github.com)

Neural-Networks/DL.py at main · Adam-226/Neural-Networks (github.com)

PyTorch-Tutorial/tutorial-contents/401_CNN.py at master · MorvanZhou/PyTorch-Tutorial (github.com)

CNN中卷积核个数大小及层数的确定_cnn卷积层数确定-CSDN博客

梯度值与参数更新optimizer.zero_grad(),loss.backward、和optimizer.step()、lr_scheduler.step原理解析_模型共享权重参数更新-CSDN博客

https://openaccess.thecvf.com/content_CVPR_2019/papers/Li_Partial_Order_Pruning_For_Best_SpeedAccuracy_Trade-Off_in_Neural_Architecture_CVPR_2019_paper.pdf

https://stackoverflow.com/questions/59129812/how-to-avoid-cuda-out-of-memory-in-pytorch