机器学习-第三次实验

李昕.202100460065

1 实验目的

任务一：熟悉`Pytorch`的使用，包括：

- `Pytorch`的安装以及环境搭建
- `Pytorch`处理数据
  - `Pytorch`计算梯度以及搭建神经网络
  - `Pytorch`训练模型

任务二：使用`Pytorch`训练CNN模型，实作一个图像分类器，预测 testing set 中三千多笔数据

2 实验环境

训练平台	使用语言	运行内存	显卡配置
Kaggle	python3	30G	P100-16G

3 实验方法-CNN

CNN是卷积神经网络（Convolutional Neural Network）,是一种常用于图像处理和计算机视觉任务的深度学习模型。

CNN的核心思想是利用卷积层和池化层进行特征提取和数据降维，然后通过全连接层进行分类或回归。相比于传统的全连接神经网络，CNN能够更好地处理图像数据的局部关联性和平移不变性，使得模型更加高效和准确。

CNN的主要组成部分包括：

1. 卷积层（Convolutional Layer）：通过应用一系列滤波器（卷积核）对输入图像进行卷积操作，提取图像的局部特征。卷积操作可以捕捉到图像中的边缘、纹理等低级特征。
2. 池化层（Pooling Layer）：通过对卷积层的输出进行下采样，减小特征图的尺寸，提取出更加重要的特征。常用的池化操作有最大池化和平均池化。
3. 全连接层（Fully Connected Layer）：将池化层的输出展平后，连接到一个或多个全连接层，用于分类或回归任务。

4 实验过程

4.1 基础CNN模型

4.1.1 读取数据

在readfile()函数里进行数据读取和大小统一，将每个读入图片的形状修改为为128*128，便于进行处理：

```
#Read image  利用 OpenCV (cv2) 读入照片存放在 numpy array 中
def readfile(path, label):
    # label 是一个 boolean variable
    image_dir = sorted(os.listdir(path))
    # 创建x为输入向量,为输出向量
    x = np.zeros((len(image_dir), 128, 128, 3), dtype=np.uint8)#图片像素大小128*128，3个
channel
    y = np.zeros((len(image_dir)), dtype=np.uint8) #每个图片各有1个label
    for i, file in enumerate(image_dir):  #读取图片数据
        img = cv2.imread(os.path.join(path, file))
        x[i, :, :] = cv2.resize(img,(128, 128))# 改变图像尺寸，统一128*128
        if label:
          y[i] = int(file.split("_")[0])# 判断是否有label，有则存入y
    if label:
      return x, y
    else:
      return x
```

4.1.2 数据预处理（数据增强和归一化）

对训练数据进行如下操作：

1. 将输入的`tensor`转换为`PIL`图像，进行处理。
2. 在训练时进行随机水平翻转，以50%的概率将图像水平翻转。
3. 在训练时进行随机旋转，将图像在-15度到+15度的范围内进行随机旋转。
4. 将处理后的图像转换回`tensor`形式。
5. 对图像进行归一化，使得每个通道的均值为0.5，标准差为0.5。

在训练时，进行数据增强和归一化。在测试时，需要对模型进行评估，而不是训练它，因此不需要增强和归一化操作：

```
 # training 時做 data augmentation
train_transform = transforms.Compose([
    transforms.ToPILImage(),#将tensor转换为PIL图像
    transforms.RandomHorizontalFlip(),#随机水平翻转（RandomHorizontalFlip）通过以50%的概率
将图像水平翻转来增加图像的多样性，提高模型的泛化能力
    transforms.RandomRotation(15), #随机反转（RandomRotation）通过在一定角度范围内对图像进行
随机旋转来增加数据的多样性。转的角度范围是-15到+15度。
    transforms.ToTensor(), #将tensor转换为PIL图像
```

```
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))#对图像进行归一化，使得每个通道的均值为
0.5，标准差为0.5
])
# testing 時不需做 data augmentation
test_transform = transforms.Compose([
    transforms.ToPILImage(),     #将tensor转换为PIL图像
    transforms.ToTensor(),#将tensor转换为PIL图像
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))#对图像进行归一化，使得每个通道的均值为
0.5，标准差为0.5
])
```

4.1.3 设置CNN Model

在torch中，卷积运算函数`torch.nn.Conv2d`进行二维卷积运算，归一化处理函数`torch.nn.BatchNorm2d`是用来在过激活函数之前对特征进行归一化处理的。它的参数是特征图的通道数`channels`。

激活函数`torch.nn.ReLU`是常用的非线性激活函数之一。ReLU（Rectified Linear Unit）函数将所有负值变为零，保留正值不变。

池化函数`torch.nn.MaxPool2d`采用 MaxPooling 策略进行池化操作，可以降低特征图的维度，减少模型的计算量。它的参数包括池化核的大小`kernel_size`、步长`stride`和填充`padding`。

设置五层的卷积层和最大池化层如下(具体分层见代码注释)：

```
self.cnn = nn.Sequential(
            # ----------------  第一个卷积层+最大池化层  ------------------#
            nn.Conv2d(3, 64, 3, 1, 1),
    #第一个卷积层，输入通道数为3，输出通道数为64，卷积核大小为3x3，步长为1，填充为1
            nn.BatchNorm2d(64),# 批标准化层，用于对卷积层的输出进行归一化
            nn.ReLU(), # ReLU激活函数
            nn.MaxPool2d(2, 2, 0),# 最大池化层，池化核大小为2x2，步长为2
            # ----------------  第二个卷积层+最大池化层  ------------------#
            nn.Conv2d(64, 128, 3, 1, 1),
    #第二个卷积层，输入通道数为64，输出通道数为128，卷积核大小为3x3，步长为1，填充为1
            nn.BatchNorm2d(128),# 批标准化层，用于对卷积层的输出进行归一化
            nn.ReLU(),# ReLU激活函数
            nn.MaxPool2d(2, 2, 0),# 最大池化层，池化核大小为2x2，步长为2
            # ----------------  第三个卷积层+最大池化层  ------------------#
            nn.Conv2d(128, 256, 3, 1, 1),
    #第三个卷积层，输入通道数为128，输出通道数为256，卷积核大小为3x3，步长为1，填充为1
            nn.BatchNorm2d(256),#同上，以下不再解释
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),
            # ----------------  第四个卷积层+最大池化层  ------------------#
            nn.Conv2d(256, 512, 3, 1, 1),
            nn.BatchNorm2d(512),
```

```
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),
            # ----------------  第五个卷积层+最大池化层  ------------------#
            nn.Conv2d(512, 512, 3, 1, 1),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(2, 2, 0),
        )
```

参数数量 = (输入通道数 × 卷积核大小 × 卷积核大小) × 输出通道数+全连接层输出*(1+全连接层输出)，则总参数5层
=1728+73728+294912+1179648+2359296+8389632+524800+5643=12829387 个。

定义全连接网络（Fully Connected Network），其中包含了三个线性层和两个ReLU激活函数：

```
self.fc = nn.Sequential(  #构建全连接神经网络
            # 输入512*4*4, 输出1024*1
            nn.Linear(512*4*4, 1024),#全连接分类器
            nn.ReLU(),
            # 输入1024*1, 输出512*1
            nn.Linear(1024, 512),#全连接分类器
            nn.ReLU(),#激活函数选择ReLU
            # 输入512*1, 输出11*1, 即11种食物种类
            nn.Linear(512, 11)
        )
```

4.1.4 训练和验证

初始化超参数，使用train_loader迭代训练集，获取每个batch的数据和标签，将将模型分别设为训练模式和评估模式，统计每个batch的训练/验证准确率和总loss：

```
for epoch in range(num_epoch):
    epoch_start_time = time.time()# 记录开始时间
    train_acc = 0.0 # 初始化系列准确率
    train_loss = 0.0
    val_acc = 0.0
    val_loss = 0.0
    model.train() # 確保 model 是在 train model
    for i, data in enumerate(tqdm(train_loader)):#使用tqdm显示进度
        optimizer.zero_grad() # 梯度参数归零，清除之前的梯度信息
        train_pred = model(data[0].cuda()) # 将输入数据传入模型中进行前向计算，得到预测值
        batch_loss = loss(train_pred, data[1].cuda())# 计算这个batch的loss
        batch_loss.backward() # 对损失值进行反向传播，计算每个参数的梯度
        optimizer.step() #根据计算的梯度更新模型的参数，以使模型朝着损失函数的最小化方向优化。
        train_acc += np.sum(np.argmax(train_pred.cpu().data.numpy(), axis=1) ==
data[1].numpy())## 更新准确率
        train_loss += batch_loss.item()# batch loss加入总loss
    model.eval()  #将模型设为评估模式，即不进行梯度计算，仅进行前向传播推断
```

```
    with torch.no_grad(): # 禁用梯度计算
        for i, data in enumerate(val_loader):
            val_pred = model(data[0].cuda())# 将输入数据传递给模型并在CUDA设备上进行计算
            batch_loss = loss(val_pred, data[1].cuda())# 计算损失函数值
            val_acc += np.sum(np.argmax(val_pred.cpu().data.numpy(), axis=1) ==
data[1].numpy())# 统计正确预测的样本数
            val_loss += batch_loss.item() # 累计批次损失值
```

4.1.5 拼接训练集和验证集

通过训练找到了比较好的参数，此时将训练集和验证集合并为新的更大的训练集，在这个新的训练集上再次训练，这样充分利用了实验给出的数据集，代码与注释如下：

```
#得到好的参数后，使用 training set 和 validation set 共同训练
train_val_x = np.concatenate((train_x, val_x), axis=0)
# 将训练集和验证集的特征数据按行连接在一起
train_val_y = np.concatenate((train_y, val_y), axis=0)
# 将训练集和验证集的标签数据按行连接在一起
train_val_set = ImgDataset(train_val_x, train_val_y, train_transform)   # 创建一个包含训
练-验证数据集的自定义数据集对象，用于训练和验证
train_val_loader = DataLoader(train_val_set, batch_size=batch_size, shuffle=True)  # 使用
训练-验证数据集创建一个数据加载器，用于批量加载数据并进行训练
```

4.2 CNN深度减半的模型

实验要求实作与第一题接近的参数量，但CNN深度（CNN层数）减半的模型，由于构造的第一个模型是5层模型，故构造一个新的三层模型，模型代码如下：

```
        self.cnn = nn.Sequential(
            #为了达到与模型A相近的参数量，需要适当调整filter的数量
            # ----------------  第一个卷积层+最大池化层  ------------------#
            nn.Conv2d(3, 256, 3, 1, 1), #第一个卷积层，输入通道数为3，输出通道数为256，卷积核
大小为3x3，步长为1，填充为1
            nn.BatchNorm2d(256),# 批标准化层，用于对卷积层的输出进行归一化
            nn.ReLU(), # ReLU激活函数
            nn.MaxPool2d(2, 2, 0),# 最大池化层，池化核大小为2x2，步长为2
            # ----------------  第二个卷积层+最大池化层  ------------------#
            nn.Conv2d(256, 512, 3, 1, 1),#第二个卷积层，输入通道数为256，输出通道数为512，卷
积核大小为3x3，步长为1，填充为1
            nn.BatchNorm2d(512),# 批标准化层，用于对卷积层的输出进行归一化
            nn.ReLU(),# ReLU激活函数
            nn.MaxPool2d(4, 4, 0),# 最大池化层，池化核大小为4x4，步长为4
            # ----------------  第三个卷积层+最大池化层  ------------------#
            nn.Conv2d(512, 512, 3, 1, 1),#第三个卷积层，输入通道数为512，输出通道数为512，卷
积核大小为3x3，步长为1，填充为1
            nn.BatchNorm2d(512),#同上，以下不再解释
            nn.ReLU(),
```

```
            nn.MaxPool2d(4, 4, 0), )
```

模型包含三个卷积层和池化层：

1. 第一个卷积层：输入通道数为3，输出通道数为256，卷积核大小为3x3，经过ReLU激活函数和最大池化操作（池化核大小为2x2，步长为2）；
2. 第二个卷积层：输入通道数为256，输出通道数为512，卷积核大小为3x3，经过ReLU激活函数和最大池化操作（池化核大小为4x4，步长为4）；
3. 第三个卷积层：输入通道数为512，输出通道数为512，卷积核大小为3x3，经过ReLU激活函数和最大池化操作（池化核大小为4x4，步长为4）。

参数数量 = (输入通道数 × 卷积核大小 × 卷积核大小) × 输出通道数+全连接层输出*(1+全连接层输出)，则总参数3层=6912+1179648+2359296+8389632+524800+5643=12,465,904个。保证其与5层的参数数量接近。

4.3 模型加入归一化和数据增强

归一化是指将输入数据进行标准化处理，使其数值范围在一个较小的区间内，以便更好地进行训练。计算公式为：

$$z_i = \frac{x_i - \mu}{\sigma}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

在本实验中，通过`transforms.Normalize`语句对输入进行归一化操作。

数据增强是通过对原始数据进行一系列随机变换来扩充数据集的方法。这些随机变换可以包括平移、旋转、缩放、翻转、剪切等。数据增强有助于增加数据的多样性，使模型能够更好地泛化和抵抗过拟合。通过随机变换数据，可以生成更多的样本，从而提高模型的鲁棒性和泛化能力。

在本实验中，通过`#transforms.RandomHorizontalFlip()`随机水平反转语句和`transforms.RandomRotation(15)`随机旋转语句进行数据增强操作。

5 结果分析与评估

5.1 5层CNN模型的准确率

在五层的CNN模型中，采取了五个卷积层和池化层，每个卷积层都包含了一个卷积操作、批标准化和ReLU激活函数，然后接着进行最大池化操作。最后由三层全连接神经网络将提取到的特征映射到输出类别上，以进行分类，总参数为12829387 ，训练结果如下：

```
Reading data
Size of training data = 9866
Size of validation data = 3430
Size of Testing data = 3347
[001/030] 21.01 sec(s) Train Acc: 0.241537 Loss: 0.017376 | Val Acc: 0.269096 loss: 0.016538
[002/030] 15.34 sec(s) Train Acc: 0.348672 Loss: 0.014565 | Val Acc: 0.342566 loss: 0.015013
[003/030] 15.42 sec(s) Train Acc: 0.404825 Loss: 0.013445 | Val Acc: 0.386880 loss: 0.013752
[004/030] 15.33 sec(s) Train Acc: 0.461991 Loss: 0.012159 | Val Acc: 0.410496 loss: 0.013731
[005/030] 15.36 sec(s) Train Acc: 0.508717 Loss: 0.011236 | Val Acc: 0.355394 loss: 0.015480
[006/030] 15.28 sec(s) Train Acc: 0.547942 Loss: 0.010266 | Val Acc: 0.432362 loss: 0.013133
[007/030] 15.37 sec(s) Train Acc: 0.586357 Loss: 0.009447 | Val Acc: 0.418950 loss: 0.014708
[008/030] 15.27 sec(s) Train Acc: 0.604095 Loss: 0.008924 | Val Acc: 0.543149 loss: 0.010688
[009/030] 15.28 sec(s) Train Acc: 0.637847 Loss: 0.008219 | Val Acc: 0.504082 loss: 0.012490
[010/030] 15.26 sec(s) Train Acc: 0.666532 Loss: 0.007573 | Val Acc: 0.537609 loss: 0.011720
[011/030] 15.35 sec(s) Train Acc: 0.702210 Loss: 0.006902 | Val Acc: 0.445190 loss: 0.016813
[012/030] 15.28 sec(s) Train Acc: 0.703223 Loss: 0.006649 | Val Acc: 0.568805 loss: 0.010515
[013/030] 15.40 sec(s) Train Acc: 0.737989 Loss: 0.005969 | Val Acc: 0.500875 loss: 0.013343
[014/030] 15.62 sec(s) Train Acc: 0.772654 Loss: 0.005208 | Val Acc: 0.581050 loss: 0.010398
[015/030] 15.55 sec(s) Train Acc: 0.790391 Loss: 0.004721 | Val Acc: 0.556560 loss: 0.012761
[016/030] 15.58 sec(s) Train Acc: 0.820495 Loss: 0.004164 | Val Acc: 0.524781 loss: 0.015687
[017/030] 15.57 sec(s) Train Acc: 0.812589 Loss: 0.004293 | Val Acc: 0.619825 loss: 0.010356
[018/030] 15.50 sec(s) Train Acc: 0.854044 Loss: 0.003375 | Val Acc: 0.605248 loss: 0.012395
[019/030] 15.48 sec(s) Train Acc: 0.890432 Loss: 0.002460 | Val Acc: 0.614869 loss: 0.012941
[020/030] 15.60 sec(s) Train Acc: 0.889520 Loss: 0.002482 | Val Acc: 0.545773 loss: 0.017137
[021/030] 15.47 sec(s) Train Acc: 0.934523 Loss: 0.001539 | Val Acc: 0.604082 loss: 0.014926
[022/030] 15.33 sec(s) Train Acc: 0.926515 Loss: 0.001657 | Val Acc: 0.604665 loss: 0.014340
[023/030] 15.38 sec(s) Train Acc: 0.956315 Loss: 0.001053 | Val Acc: 0.614577 loss: 0.015877
[024/030] 15.32 sec(s) Train Acc: 0.934725 Loss: 0.001475 | Val Acc: 0.636152 loss: 0.014805
[025/030] 15.40 sec(s) Train Acc: 0.944253 Loss: 0.001244 | Val Acc: 0.518367 loss: 0.020768
[026/030] 15.33 sec(s) Train Acc: 0.940401 Loss: 0.001313 | Val Acc: 0.606122 loss: 0.017311
[027/030] 15.39 sec(s) Train Acc: 0.976079 Loss: 0.000619 | Val Acc: 0.644898 loss: 0.015969
[028/030] 15.52 sec(s) Train Acc: 0.972228 Loss: 0.000637 | Val Acc: 0.657434 loss: 0.015966
[029/030] 15.36 sec(s) Train Acc: 0.979424 Loss: 0.000462 | Val Acc: 0.620700 loss: 0.017269
[030/030] 15.26 sec(s) Train Acc: 0.982364 Loss: 0.000451 | Val Acc: 0.641983 loss: 0.018636
```

训练集准确率为0.9823，验证集准确率为0.6419.

5.2 3层和10层CNN模型的准确率

在五层的CNN模型中，采取了五个卷积层和池化层，每个卷积层都包含了一个卷积操作、批标准化和ReLU激活函数，然后接着进行最大池化操作。最后由三层全连接神经网络将提取到的特征映射到输出类别上，以进行分类，总参数为12,465,904，训练结果如下：

```
Reading data
Size of training data = 9866
Size of validation data = 3430
Size of Testing data = 3347
[001/030] 48.32 sec(s) Train Acc: 0.245895 Loss: 0.019146 | Val Acc: 0.307580 loss: 0.015175
[002/030] 42.62 sec(s) Train Acc: 0.354956 Loss: 0.014423 | Val Acc: 0.341108 loss: 0.014714
[003/030] 42.71 sec(s) Train Acc: 0.435739 Loss: 0.012819 | Val Acc: 0.404373 loss: 0.013646
[004/030] 42.86 sec(s) Train Acc: 0.482364 Loss: 0.011607 | Val Acc: 0.443440 loss: 0.012610
[005/030] 42.84 sec(s) Train Acc: 0.533550 Loss: 0.010570 | Val Acc: 0.434111 loss: 0.013151
[006/030] 42.86 sec(s) Train Acc: 0.572066 Loss: 0.009772 | Val Acc: 0.474636 loss: 0.013107
[007/030] 42.81 sec(s) Train Acc: 0.606122 Loss: 0.008947 | Val Acc: 0.470554 loss: 0.013232
[008/030] 42.79 sec(s) Train Acc: 0.624569 Loss: 0.008458 | Val Acc: 0.568222 loss: 0.010130
[009/030] 42.80 sec(s) Train Acc: 0.677884 Loss: 0.007370 | Val Acc: 0.389213 loss: 0.020956
[010/030] 42.82 sec(s) Train Acc: 0.691364 Loss: 0.007187 | Val Acc: 0.477843 loss: 0.014069
[011/030] 42.88 sec(s) Train Acc: 0.667444 Loss: 0.007518 | Val Acc: 0.558017 loss: 0.011130
[012/030] 42.88 sec(s) Train Acc: 0.729374 Loss: 0.006226 | Val Acc: 0.543440 loss: 0.011502
[013/030] 42.93 sec(s) Train Acc: 0.749443 Loss: 0.005703 | Val Acc: 0.562682 loss: 0.011750
[014/030] 42.92 sec(s) Train Acc: 0.758058 Loss: 0.005495 | Val Acc: 0.529155 loss: 0.013364
[015/030] 42.91 sec(s) Train Acc: 0.773870 Loss: 0.005131 | Val Acc: 0.633528 loss: 0.009634
[016/030] 42.94 sec(s) Train Acc: 0.807926 Loss: 0.004330 | Val Acc: 0.650146 loss: 0.009338
[017/030] 42.91 sec(s) Train Acc: 0.826272 Loss: 0.003885 | Val Acc: 0.572012 loss: 0.013432
[018/030] 42.89 sec(s) Train Acc: 0.855159 Loss: 0.003360 | Val Acc: 0.601458 loss: 0.011857
[019/030] 42.92 sec(s) Train Acc: 0.843706 Loss: 0.003436 | Val Acc: 0.649271 loss: 0.010432
[020/030] 42.87 sec(s) Train Acc: 0.854348 Loss: 0.003318 | Val Acc: 0.587464 loss: 0.012731
[021/030] 42.95 sec(s) Train Acc: 0.873708 Loss: 0.002922 | Val Acc: 0.637026 loss: 0.011414
[022/030] 42.82 sec(s) Train Acc: 0.909386 Loss: 0.002156 | Val Acc: 0.531195 loss: 0.021739
[023/030] 42.91 sec(s) Train Acc: 0.868944 Loss: 0.003147 | Val Acc: 0.606706 loss: 0.014291
[024/030] 42.81 sec(s) Train Acc: 0.874113 Loss: 0.002908 | Val Acc: 0.556560 loss: 0.016286
[025/030] 42.82 sec(s) Train Acc: 0.911210 Loss: 0.002023 | Val Acc: 0.629446 loss: 0.012977
[026/030] 42.82 sec(s) Train Acc: 0.941415 Loss: 0.001368 | Val Acc: 0.541691 loss: 0.021412
[027/030] 42.89 sec(s) Train Acc: 0.922157 Loss: 0.001805 | Val Acc: 0.645773 loss: 0.013192
[028/030] 42.92 sec(s) Train Acc: 0.946382 Loss: 0.001283 | Val Acc: 0.643149 loss: 0.013126
[029/030] 42.92 sec(s) Train Acc: 0.956923 Loss: 0.001047 | Val Acc: 0.660058 loss: 0.014044
[030/030] 42.99 sec(s) Train Acc: 0.965234 Loss: 0.000802 | Val Acc: 0.654519 loss: 0.013969
```

训练集准确率为0.9652，验证集准确率为0.6548.可以看到相同的训练次数下，3层CNN神经网络相对于5层收敛的更快一些,但训练结果较差。

为了更好的比较准确率和收敛速度，重新改写一个10层的CNN，尽量保证其参数与5层和3层cnn接近，迭代60次得到以下结果：

```
[006/060] 11.62 sec(s) Train Acc: 0.393067 Loss: 0.013484 | Val Acc: 0.366764 loss: 0.014025
[007/060] 11.70 sec(s) Train Acc: 0.410703 Loss: 0.013022 | Val Acc: 0.328863 loss: 0.016641
[008/060] 11.65 sec(s) Train Acc: 0.436246 Loss: 0.012481 | Val Acc: 0.425948 loss: 0.013095
[009/060] 11.67 sec(s) Train Acc: 0.480843 Loss: 0.011650 | Val Acc: 0.309913 loss: 0.020608
[010/060] 11.75 sec(s) Train Acc: 0.485911 Loss: 0.011662 | Val Acc: 0.350729 loss: 0.015877
[011/060] 11.62 sec(s) Train Acc: 0.510643 Loss: 0.011057 | Val Acc: 0.360933 loss: 0.014979
[012/060] 11.65 sec(s) Train Acc: 0.533144 Loss: 0.010466 | Val Acc: 0.360058 loss: 0.018998
[013/060] 11.80 sec(s) Train Acc: 0.556761 Loss: 0.009994 | Val Acc: 0.453353 loss: 0.013210
[014/060] 11.69 sec(s) Train Acc: 0.578046 Loss: 0.009620 | Val Acc: 0.488047 loss: 0.011864
[015/060] 11.80 sec(s) Train Acc: 0.596594 Loss: 0.009178 | Val Acc: 0.481924 loss: 0.013207
[016/060] 11.65 sec(s) Train Acc: 0.607541 Loss: 0.008838 | Val Acc: 0.510496 loss: 0.012318
[017/060] 11.62 sec(s) Train Acc: 0.632272 Loss: 0.008390 | Val Acc: 0.434402 loss: 0.018039
[018/060] 11.77 sec(s) Train Acc: 0.634908 Loss: 0.008033 | Val Acc: 0.506414 loss: 0.012186
[019/060] 11.66 sec(s) Train Acc: 0.671093 Loss: 0.007529 | Val Acc: 0.453936 loss: 0.015281
[020/060] 11.66 sec(s) Train Acc: 0.667342 Loss: 0.007610 | Val Acc: 0.513411 loss: 0.012819
[021/060] 11.66 sec(s) Train Acc: 0.706771 Loss: 0.006700 | Val Acc: 0.507580 loss: 0.015283
[022/060] 11.62 sec(s) Train Acc: 0.699270 Loss: 0.006870 | Val Acc: 0.520700 loss: 0.012586
[023/060] 11.66 sec(s) Train Acc: 0.726738 Loss: 0.006180 | Val Acc: 0.541399 loss: 0.012428
[024/060] 11.71 sec(s) Train Acc: 0.744679 Loss: 0.005704 | Val Acc: 0.567638 loss: 0.011735
[025/060] 11.65 sec(s) Train Acc: 0.756639 Loss: 0.005401 | Val Acc: 0.585131 loss: 0.010668
[026/060] 11.74 sec(s) Train Acc: 0.787148 Loss: 0.004798 | Val Acc: 0.582216 loss: 0.012648
[027/060] 11.68 sec(s) Train Acc: 0.786844 Loss: 0.004949 | Val Acc: 0.541108 loss: 0.013431
[028/060] 11.67 sec(s) Train Acc: 0.781776 Loss: 0.004904 | Val Acc: 0.596210 loss: 0.010532
[029/060] 11.71 sec(s) Train Acc: 0.797385 Loss: 0.004649 | Val Acc: 0.569096 loss: 0.013599
[030/060] 11.69 sec(s) Train Acc: 0.824549 Loss: 0.003932 | Val Acc: 0.545773 loss: 0.013230
[031/060] 11.64 sec(s) Train Acc: 0.839854 Loss: 0.003670 | Val Acc: 0.473469 loss: 0.020716
[032/060] 11.67 sec(s) Train Acc: 0.850699 Loss: 0.003393 | Val Acc: 0.518367 loss: 0.018487
[033/060] 11.65 sec(s) Train Acc: 0.866917 Loss: 0.003034 | Val Acc: 0.608455 loss: 0.012936
[034/060] 11.70 sec(s) Train Acc: 0.869147 Loss: 0.002871 | Val Acc: 0.505831 loss: 0.017033
[035/060] 11.65 sec(s) Train Acc: 0.875938 Loss: 0.002826 | Val Acc: 0.609038 loss: 0.014388
[036/060] 11.68 sec(s) Train Acc: 0.856375 Loss: 0.003400 | Val Acc: 0.618076 loss: 0.012205
[037/060] 11.68 sec(s) Train Acc: 0.850699 Loss: 0.003419 | Val Acc: 0.600000 loss: 0.014064
[038/060] 11.64 sec(s) Train Acc: 0.885668 Loss: 0.002529 | Val Acc: 0.623032 loss: 0.013160
[039/060] 11.68 sec(s) Train Acc: 0.938577 Loss: 0.001442 | Val Acc: 0.547230 loss: 0.021865
[040/060] 11.72 sec(s) Train Acc: 0.866410 Loss: 0.003145 | Val Acc: 0.620117 loss: 0.014878
[041/060] 11.66 sec(s) Train Acc: 0.928948 Loss: 0.001772 | Val Acc: 0.640525 loss: 0.013710
[042/060] 11.74 sec(s) Train Acc: 0.922968 Loss: 0.001780 | Val Acc: 0.612828 loss: 0.014885
[043/060] 11.64 sec(s) Train Acc: 0.963714 Loss: 0.000850 | Val Acc: 0.618076 loss: 0.017954
[044/060] 11.67 sec(s) Train Acc: 0.945571 Loss: 0.001320 | Val Acc: 0.622449 loss: 0.017271
[045/060] 11.76 sec(s) Train Acc: 0.918711 Loss: 0.001961 | Val Acc: 0.574052 loss: 0.020503
[046/060] 11.65 sec(s) Train Acc: 0.949828 Loss: 0.001158 | Val Acc: 0.610787 loss: 0.018296
[047/060] 11.65 sec(s) Train Acc: 0.969795 Loss: 0.000773 | Val Acc: 0.630029 loss: 0.018886
[048/060] 11.67 sec(s) Train Acc: 0.932800 Loss: 0.001532 | Val Acc: 0.621866 loss: 0.015519
[049/060] 11.63 sec(s) Train Acc: 0.973748 Loss: 0.000821 | Val Acc: 0.576968 loss: 0.021646
[050/060] 11.73 sec(s) Train Acc: 0.832049 Loss: 0.004141 | Val Acc: 0.558892 loss: 0.017548
[051/060] 11.67 sec(s) Train Acc: 0.926009 Loss: 0.001735 | Val Acc: 0.569096 loss: 0.019082
[052/060] 11.65 sec(s) Train Acc: 0.941618 Loss: 0.001392 | Val Acc: 0.648980 loss: 0.014805
[053/060] 11.76 sec(s) Train Acc: 0.974458 Loss: 0.000668 | Val Acc: 0.661224 loss: 0.016611
[054/060] 11.66 sec(s) Train Acc: 0.985404 Loss: 0.000428 | Val Acc: 0.632070 loss: 0.017325
[055/060] 11.65 sec(s) Train Acc: 0.949726 Loss: 0.001330 | Val Acc: 0.604665 loss: 0.019505
[056/060] 11.76 sec(s) Train Acc: 0.905737 Loss: 0.002383 | Val Acc: 0.599417 loss: 0.017989
[057/060] 11.59 sec(s) Train Acc: 0.969491 Loss: 0.000779 | Val Acc: 0.634985 loss: 0.017555
[058/060] 11.67 sec(s) Train Acc: 0.989864 Loss: 0.000278 | Val Acc: 0.653353 loss: 0.016463
[059/060] 11.67 sec(s) Train Acc: 0.994020 Loss: 0.000178 | Val Acc: 0.653936 loss: 0.017875
[060/060] 11.66 sec(s) Train Acc: 0.960977 Loss: 0.000954 | Val Acc: 0.589504 loss: 0.023095
```

可以看到其收敛速度更慢，在第50-60次迭代时准确率波动，可能是过拟合导致的，但训练效相较于5层和3层有了提升。

5.3 归一化对模型准确率的影响

在五层模型的基础上加入归一化，得到以下结果：

```
Reading data
Size of training data = 9866
Size of validation data = 3430
Size of Testing data = 3347
[001/030] 17.22 sec(s) Train Acc: 0.203020 Loss: 0.018353 | Val Acc: 0.170845 loss: 0.019159
[002/030] 17.17 sec(s) Train Acc: 0.309548 Loss: 0.015441 | Val Acc: 0.311662 loss: 0.015077
[003/030] 17.15 sec(s) Train Acc: 0.372390 Loss: 0.014026 | Val Acc: 0.363265 loss: 0.014698
[004/030] 17.01 sec(s) Train Acc: 0.432293 Loss: 0.012743 | Val Acc: 0.374927 loss: 0.014437
[005/030] 17.06 sec(s) Train Acc: 0.465741 Loss: 0.012008 | Val Acc: 0.457434 loss: 0.012062
[006/030] 17.02 sec(s) Train Acc: 0.516319 Loss: 0.011030 | Val Acc: 0.519242 loss: 0.011067
[007/030] 17.00 sec(s) Train Acc: 0.544294 Loss: 0.010292 | Val Acc: 0.499125 loss: 0.011476
[008/030] 17.05 sec(s) Train Acc: 0.577235 Loss: 0.009545 | Val Acc: 0.372595 loss: 0.015897
[009/030] 16.98 sec(s) Train Acc: 0.612812 Loss: 0.008713 | Val Acc: 0.539067 loss: 0.010882
[010/030] 17.05 sec(s) Train Acc: 0.648490 Loss: 0.008002 | Val Acc: 0.537901 loss: 0.011227
[011/030] 16.99 sec(s) Train Acc: 0.666329 Loss: 0.007418 | Val Acc: 0.491254 loss: 0.013257
[012/030] 17.11 sec(s) Train Acc: 0.708088 Loss: 0.006707 | Val Acc: 0.493586 loss: 0.013883
[013/030] 17.06 sec(s) Train Acc: 0.730387 Loss: 0.005986 | Val Acc: 0.544606 loss: 0.012072
[014/030] 16.99 sec(s) Train Acc: 0.755017 Loss: 0.005576 | Val Acc: 0.565015 loss: 0.011307
[015/030] 17.05 sec(s) Train Acc: 0.757551 Loss: 0.005528 | Val Acc: 0.528571 loss: 0.013186
[016/030] 17.14 sec(s) Train Acc: 0.796270 Loss: 0.004611 | Val Acc: 0.595044 loss: 0.011577
[017/030] 17.34 sec(s) Train Acc: 0.831239 Loss: 0.003794 | Val Acc: 0.650437 loss: 0.009667
[018/030] 17.38 sec(s) Train Acc: 0.865700 Loss: 0.002956 | Val Acc: 0.595335 loss: 0.013243
[019/030] 17.30 sec(s) Train Acc: 0.878370 Loss: 0.002750 | Val Acc: 0.513994 loss: 0.017763
[020/030] 17.26 sec(s) Train Acc: 0.897324 Loss: 0.002399 | Val Acc: 0.581633 loss: 0.014239
[021/030] 17.25 sec(s) Train Acc: 0.915670 Loss: 0.001974 | Val Acc: 0.608455 loss: 0.012611
[022/030] 17.20 sec(s) Train Acc: 0.911008 Loss: 0.002068 | Val Acc: 0.631778 loss: 0.013352
[023/030] 17.12 sec(s) Train Acc: 0.933306 Loss: 0.001541 | Val Acc: 0.632362 loss: 0.014186
[024/030] 16.98 sec(s) Train Acc: 0.933914 Loss: 0.001538 | Val Acc: 0.618659 loss: 0.015557
[025/030] 17.02 sec(s) Train Acc: 0.946483 Loss: 0.001180 | Val Acc: 0.634402 loss: 0.014864
[026/030] 17.01 sec(s) Train Acc: 0.961687 Loss: 0.000878 | Val Acc: 0.648397 loss: 0.015005
[027/030] 17.01 sec(s) Train Acc: 0.974559 Loss: 0.000631 | Val Acc: 0.548980 loss: 0.023453
[028/030] 16.94 sec(s) Train Acc: 0.952666 Loss: 0.001124 | Val Acc: 0.630321 loss: 0.016798
[029/030] 17.06 sec(s) Train Acc: 0.983985 Loss: 0.000399 | Val Acc: 0.627114 loss: 0.018494
[030/030] 17.15 sec(s) Train Acc: 0.993006 Loss: 0.000190 | Val Acc: 0.657434 loss: 0.017045
```

可以看到经过归一化后训练集准确率为0.9931，验证集准确率为0.6574。准确率相对于不加归一化有了一定的提升。

5.4 数据增强对模型准确率的影响

在五层模型中加入数据增强，得到以下结果：

```
Reading data
Size of training data = 9866
Size of validation data = 3430
Size of Testing data = 3347
[001/030] 17.37 sec(s) Train Acc: 0.260491 Loss: 0.016983 | Val Acc: 0.305248 loss: 0.015229
[002/030] 17.35 sec(s) Train Acc: 0.367930 Loss: 0.014275 | Val Acc: 0.252770 loss: 0.025566
[003/030] 17.44 sec(s) Train Acc: 0.419826 Loss: 0.013185 | Val Acc: 0.270845 loss: 0.018122
[004/030] 17.40 sec(s) Train Acc: 0.465640 Loss: 0.012091 | Val Acc: 0.432653 loss: 0.013724
[005/030] 17.40 sec(s) Train Acc: 0.503142 Loss: 0.011339 | Val Acc: 0.390087 loss: 0.014791
[006/030] 17.40 sec(s) Train Acc: 0.539124 Loss: 0.010534 | Val Acc: 0.438192 loss: 0.013058
[007/030] 17.39 sec(s) Train Acc: 0.553923 Loss: 0.010101 | Val Acc: 0.447230 loss: 0.013315
[008/030] 17.44 sec(s) Train Acc: 0.584938 Loss: 0.009595 | Val Acc: 0.383673 loss: 0.015509
[009/030] 17.36 sec(s) Train Acc: 0.601865 Loss: 0.009044 | Val Acc: 0.477259 loss: 0.013666
[010/030] 17.41 sec(s) Train Acc: 0.617880 Loss: 0.008672 | Val Acc: 0.551020 loss: 0.010551
[011/030] 17.43 sec(s) Train Acc: 0.632881 Loss: 0.008356 | Val Acc: 0.548105 loss: 0.011509
[012/030] 17.37 sec(s) Train Acc: 0.657409 Loss: 0.007891 | Val Acc: 0.492128 loss: 0.013193
[013/030] 17.39 sec(s) Train Acc: 0.673627 Loss: 0.007464 | Val Acc: 0.576968 loss: 0.010235
[014/030] 17.45 sec(s) Train Acc: 0.675451 Loss: 0.007372 | Val Acc: 0.581924 loss: 0.009942
[015/030] 17.46 sec(s) Train Acc: 0.700588 Loss: 0.006760 | Val Acc: 0.559767 loss: 0.011221
[016/030] 17.48 sec(s) Train Acc: 0.718528 Loss: 0.006496 | Val Acc: 0.483382 loss: 0.015987
[017/030] 17.46 sec(s) Train Acc: 0.723393 Loss: 0.006380 | Val Acc: 0.636735 loss: 0.008945
[018/030] 17.40 sec(s) Train Acc: 0.755423 Loss: 0.005589 | Val Acc: 0.590671 loss: 0.010532
[019/030] 17.34 sec(s) Train Acc: 0.762214 Loss: 0.005447 | Val Acc: 0.635277 loss: 0.009652
[020/030] 17.33 sec(s) Train Acc: 0.770626 Loss: 0.005229 | Val Acc: 0.630904 loss: 0.010035
[021/030] 17.35 sec(s) Train Acc: 0.775694 Loss: 0.004940 | Val Acc: 0.651020 loss: 0.009235
[022/030] 17.42 sec(s) Train Acc: 0.790189 Loss: 0.004716 | Val Acc: 0.601749 loss: 0.010487
[023/030] 17.36 sec(s) Train Acc: 0.802453 Loss: 0.004408 | Val Acc: 0.575219 loss: 0.012571
[024/030] 17.36 sec(s) Train Acc: 0.798500 Loss: 0.004531 | Val Acc: 0.573469 loss: 0.013730
[025/030] 17.41 sec(s) Train Acc: 0.830732 Loss: 0.003766 | Val Acc: 0.617493 loss: 0.010994
[026/030] 17.32 sec(s) Train Acc: 0.831137 Loss: 0.003854 | Val Acc: 0.592420 loss: 0.013269
[027/030] 17.44 sec(s) Train Acc: 0.842287 Loss: 0.003545 | Val Acc: 0.637901 loss: 0.010183
[028/030] 17.39 sec(s) Train Acc: 0.866714 Loss: 0.002983 | Val Acc: 0.656560 loss: 0.010781
[029/030] 17.41 sec(s) Train Acc: 0.871174 Loss: 0.002824 | Val Acc: 0.572303 loss: 0.014399
[030/030] 17.36 sec(s) Train Acc: 0.879789 Loss: 0.002640 | Val Acc: 0.644315 loss: 0.011562
```

可以看到经过数据增强后训练集准确率为0.8797，验证集准确率为0.644315。虽然在训练集上的准确率相比数据增强前有所降低，但是在验证集上的准确率有了提升，可以认为数据增强增强了模型的泛化能力。

6 结论

根据上述结果分析，对比3，5，10三个模型，随着层数增加，模型在训练集上训练的慢一些（收敛速度慢），表现变好（训练准确率略高），差别不大的原因可能为层数相差不大，当层数继续加深，训练收敛速度会更慢，训练效果也会提升。

归一化对于模型的准确率有一定的提升，这可能是因为归一化可以使得输入数据更加稳定，有助于模型的训练和收敛。

数据增强可以提高模型的泛化能力，有助于减少过拟合。虽然在训练集上的准确率可能会有所降低，但是在验证集上的准确率有所提升，这表明模型的泛化能力得到了提升。

综合来看，CNN训练中应该在模型训练时采取适当的措施，如归一化和数据增强，以提高模型的准确率和泛化能力。