中山大学计算机学院

人工智能

本科生实验报告

（2024学年春季学期）

课程名称：Artificial Intelligence

教学班级	人工智能(22信计+系统结构)	专业（方向）	信息与计算科学
学号	22336044	姓名	陈圳煌

一、 实验题目

深度学习：卷积神经网络（CNN）

二、 实验内容

1.算法原理

CNN即卷积神经网络是一种深度学习模型，可以处理具有网格结构的数据，如图像和视频。包含：

（1）卷积层（Conv2d）：每个卷积层由多个过滤器（也称为卷积核）组成。过滤器是一组可学习的权重，它在输入数据上滑动执行卷积操作。通过将过滤器与输入数据的小窗口相乘并求和，得到卷积结果。这个过程可以看作是特征提取的过程，通过不同的过滤器可以提取出不同的特征。

（2）池化层（Pool2d）：用于降低特征图的空间尺寸，减少参数数量和计算复杂度，同时保留最重要的特征。在本次实验中，使用最大池化（Max Pooling），取池化窗口的最大值作为池化结果。

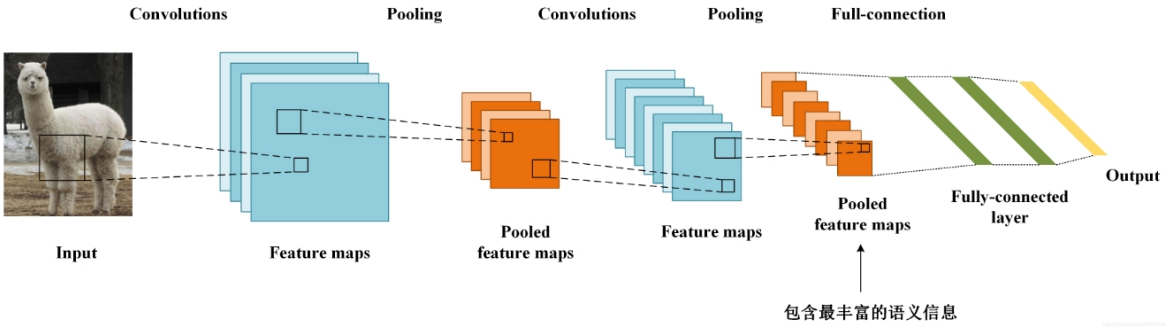
（3）激活函数（ReLU）：在卷积层和池化层之间，通常会添加一个激活函数，如ReLU。激活函数引入非线性，使得神经网络能够学习非线性关系，从而提高模型的表达能力。

（4）全连接层（Linear）：在卷积层和输出层之间，通常会添加一个或多个全连接层，用于将卷积层的输出转换为模型的最终输出。全连接层将卷积层输出的特征图展平成一维向量，然后通过全连接神经网络进行分类或回归等任务。

（5）损失函数（Loss）：CNN的训练过程需要定义一个损失函数，用于衡量模型预测结果与真实标签之间的差异。本次实验中使用“交叉熵函数”。

（6）优化器（Optimizer）：CNN的训练过程通过优化算法来更新模型参数以最小化损失函数。本次实验使用随机梯度下降（SGD）。

模型图：



2.伪代码

```
#预处理
train_set=dataset(训练文件)
test_set=dataset(测试文件)
#数据读入
trainloader=dataloader(train_set)
testloader=dataloader(test_set)
#定义卷积神经网络类
class CNN:
    初始化函数
    def __init__():继承nn.Module
    #将一个卷积层、一个激活函数、一个池化层当成一个小模块
    self.conv1(
        nn.Conv2d(
            输入的通道数in_channels;
            输出的通道数out_channels;
            卷积核的宽度kernel_size;
            步幅stride;
            填充padding;
        )卷积层
        nn.ReLU()激活函数
        nn.MaxPool2d()池化层
    )
    self.fc(输入维度，输出展成的维度)全连接层
    #前向传播函数
    def forward():
        x=conv1(x)经过第一个模块
        x=relu(x)激活函数
        x=fc(x)
train部分
for i in 训练轮数:
    for batch in 批次数:
        optimizer.zero_grad()清楚上一步的梯度
        loss计算损失函数
        loss.backward()反向传播
        更新优化器

输出训练信息

test部分
for data in 训练集:
    pred=取展平向量的最大值（即概率最大）作为预测值
```

计算准确率和损失函数

3.关键代码展示（带注释）

```
#数据路径
train_path="cnn_data/train"
test_path="cnn_data/test"
# 将一系列数据预处理步骤组合在一起
transform=transforms.Compose([
    #短边调整256，长边按比例缩放
    transforms.Resize(256),
    #裁剪成224*224
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    #标准化为均值[0.485, 0.456, 0.406]，标准差[0.229, 0.224, 0.225]
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
# 创建ImageFolder数据集实例
train_set=datasets.ImageFolder(root=train_path,transform=transform)
test_set=datasets.ImageFolder(root=test_path,transform=transform)
# 加载数据集
trainloader=torch.utils.data.DataLoader(train_set,batch_size=32,shuffle=True)#训练集大小为902,trainloader长度为29
testloader=torch.utils.data.DataLoader(test_set,batch_size=32,shuffle=False)#测试集大小为10
#定义卷积神经网络类
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.conv1=nn.Sequential(
            nn.Conv2d(
                #输入为3*224*224（channel x width x height）
                #根据计算公式:
                #width出=(width入-kernel_size+2*padding)/stride+1
                #height出=(height入-kernel_size+2*padding)/stride+1
                in_channels=3,
                #输出为16*226*226
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),#激活函数
            nn.MaxPool2d(kernel_size=2)#2x2的方块进行池化维度变为16*113*113
        )#
        self.conv2=nn.Sequential(
            nn.Conv2d(16,32,3,1,2),#16*113*123->32*115*115
            nn.ReLU(),
            nn.MaxPool2d(2)#32*115*115->32*57*57
        )
        self.fc1=nn.Linear(32*57*57,5)#32*57*57->5

    def forward(self,x):
```

```
        x=torch.relu(self.conv1(x))
        x=torch.relu(self.conv2(x))
        x=x.view(x.size(0),-1)
        x=self.fc1(x)
        return x

model=CNN()
#损失函数
Loss=nn.CrossEntropyLoss()
#优化函数
optimizer=torch.optim.SGD(model.parameters(),lr=0.01,momentum=0.9)

#train
iteration=24
loss_set=[]
for epoch in range(iteration):
    #调用了model.train()表示进入训练模式，这会启用dropout和batch normalization等特殊行为
    model.train()
    time1=time.time()
    for batch_idx, (data,target) in enumerate(trainloader):
        optimizer.zero_grad()#清楚上一步的梯度
        out=model(data)#初始化
        loss=Loss(out,target)#计算损失函数
        loss_set.append(loss.item())
        loss.backward()#反向传播
        optimizer.step()#优化器更新
        if batch_idx==0 and epoch==0:
            print(f'Initial Loss: {loss.item():>8.6f}')
        if batch_idx==len(trainloader)-1:
            time2=time.time()
            print(f'Train Epoch: [{epoch+1:<2d}/24] \tLoss: {loss.item():>8.6f}
\t Used time: {time2-time1:>6.5f}s' )

#test
#在测试时，调用了model.eval()表示进入测试模式，此时不会进行dropout和batch normalization等
特殊行为
model.eval()
test_loss=0
correct=0
#为了防止跟踪历史记录（和使用内存），使用with torch.no_grad()封装
with torch.no_grad():
    for data,target in testloader:
        out=model(data)
        test_loss+=Loss(out,target).item()
        pred=out.data.max(1,keepdim=True)[1]
        correct+=pred.eq(target.data.view_as(pred)).sum()
```

三、 实验结果及分析

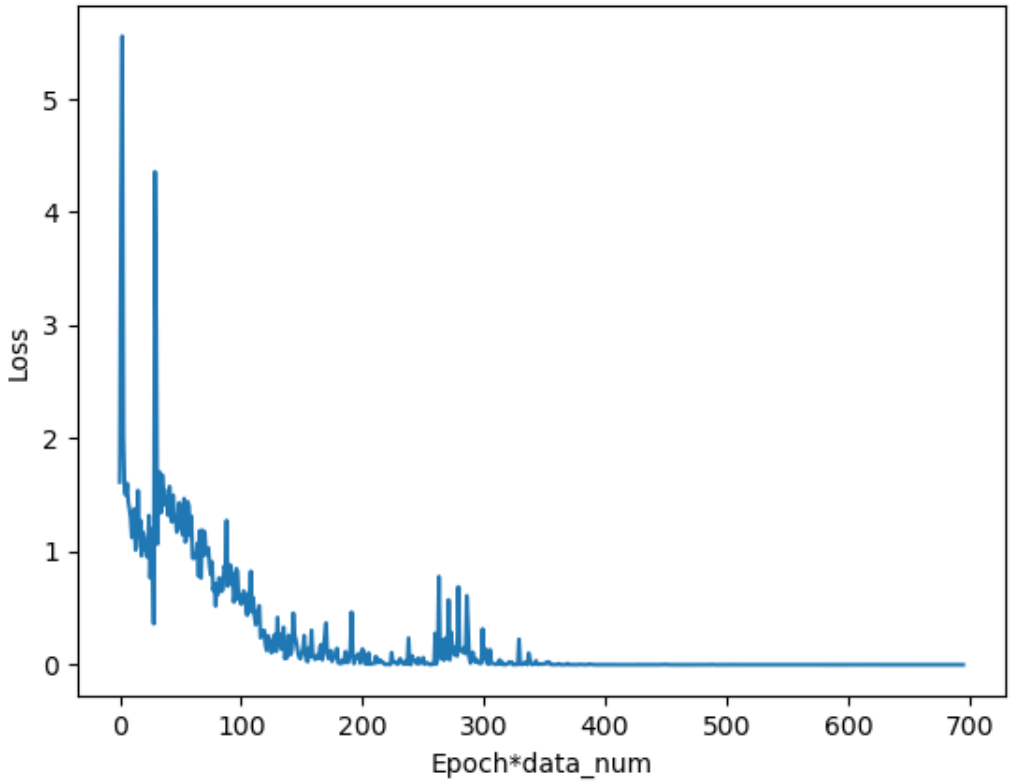
1.结果展示

两层模块（卷积层+激活函数+池化层）叠加：

（结果展示模块采用先训练后进行测试的方式，仅展示损失函数曲线，最终得到的准确率仅为最后一次测试的结果）

第一次测试：

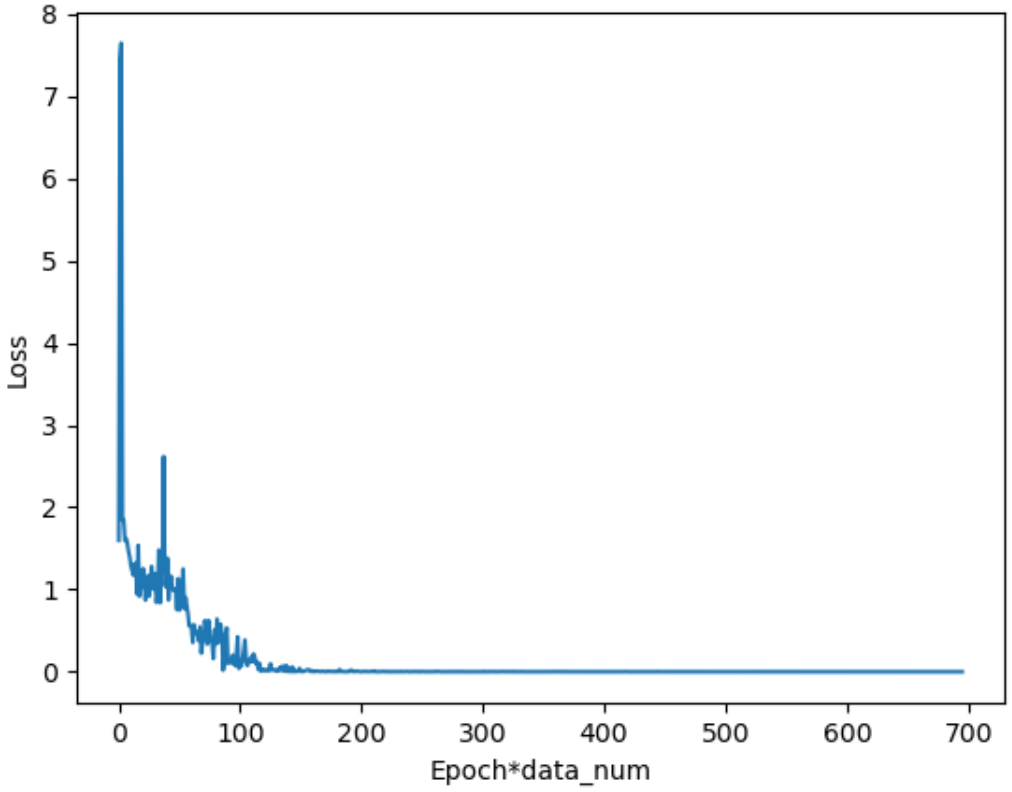
训练次数：24，最终test准确率70%。




```
Test set: Average loss: 0.1003, Accuracy: 7/10
```

第二次测试：

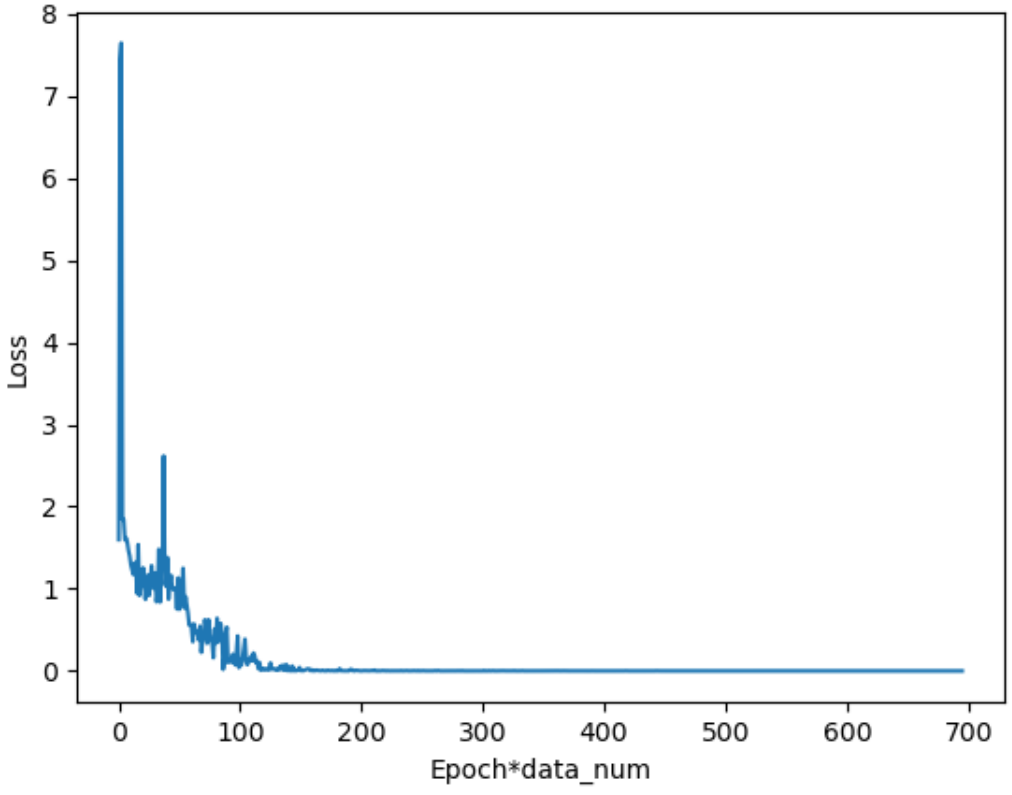
训练次数24，最终test准确率80%。




```
Test set: Average loss: 0.1557, Accuracy: 80.0%
```

第三次测试：

训练次数：24，最终test准确率：80%。

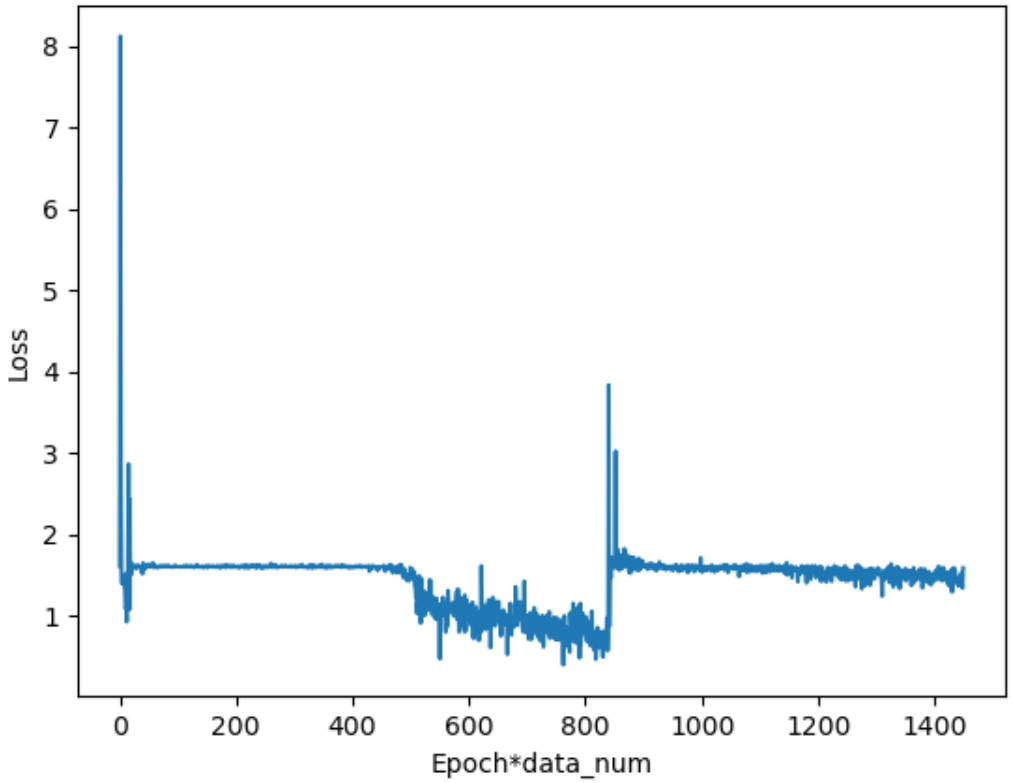


```
Test set: Average loss: 0.0887, Accuracy: 80.0%
```

在两层CNN卷积-池化层下，每一轮的训练时间大约在16s,最终的test效果大部分能够维持在80%，但是在测试中出现过一次特例。

第四次测试：

训练次数：50，最终test准确率20%（效果不好）。




```
Test set: Average loss: 0.1612, Accuracy: 20.0%
```

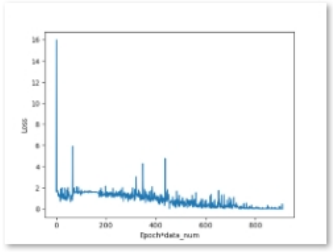
推测是损失函数陷入局部"平坦"从而梯度下降效果不佳或者训练次数过多导致了过拟合。

2.评测指标展示及分析

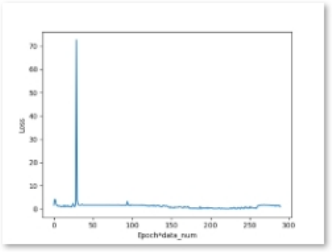
在测试结果中发现在当训练次数过多反而导致了准确率不佳的现象，接下来对比不同的训练轮数和每个批次的数据量对训练结果的影响，使用准确率以及每一轮平均时间作为指标（两层卷积-池化层）。

训练轮数（iteration）\每个批次数据量（batch_size）	**10**	**32**	**50**
10	40%（14s）	20%（14s）	60%（14s）
24	90%（15s）	80%（16s）	50%（14s）
50	50%（14.5s）	70%（14s）	70%（14s）

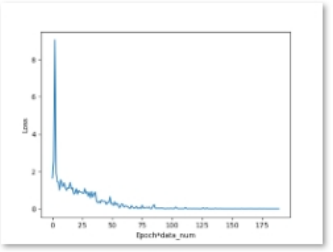
（在result文件中，以iterationxbatch_size命名得到的损失函数图）



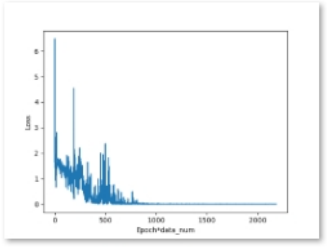
10x10.png



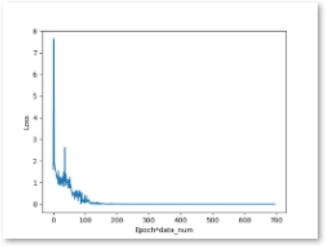
10x32.png



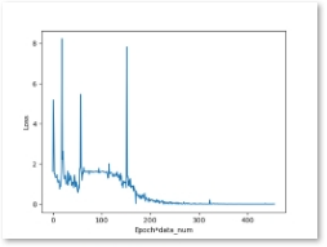
10x50.png



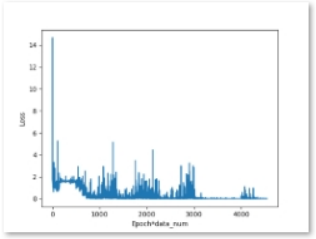
24x10.png



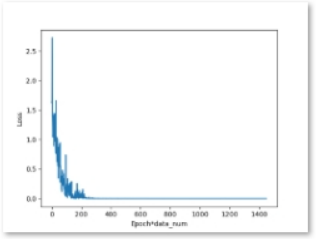
24x32.png



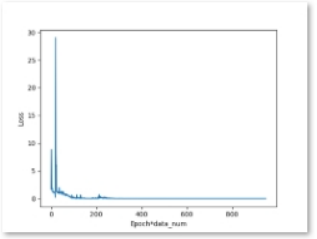
24x50.png



50x10.png



50x32.png

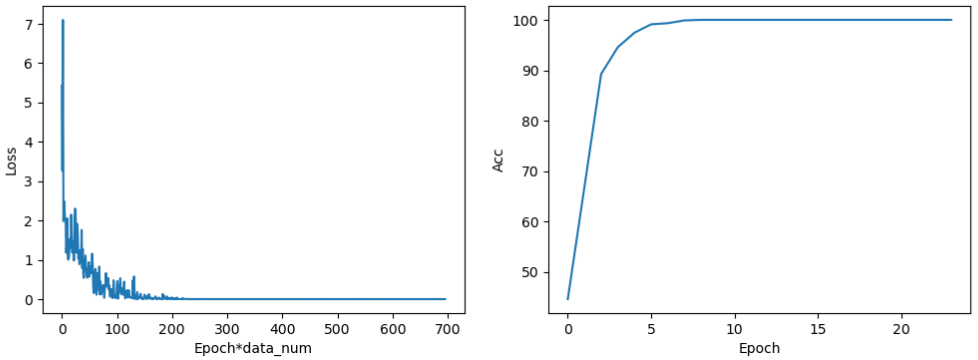


50x50.png

由测试集的准确率和损失函数的下降图像可以看出，当分批次时每个批次取过多或者过少都可能导致损失函数的下降不稳定。当训练轮数较少时，有时候会出现测试准确率很低的情况；当训练次数较多时，可能会导致过拟合，虽然损失函数到后面已经接近0，但是准确率反而降低了。由此可得当训练轮数为24，分批次时每个批次数据量为32时，准确率较优。但是由于测试集数据仅有10例，因此如果最后未趋于稳定，则可能得到的准确率是带有比较大的随机性的。

首先先验证训练层数为24层时对训练集的损失函数和准确率能够达到拟合：

```
Initial Loss: 5.431210
Train Epoch: [1 /24]    Loss:  1.900318        Acc: 44.57%     Used time: 17.95539s
Train Epoch: [2 /24]    Loss:  0.312150        Acc: 66.85%     Used time: 18.08474s
Train Epoch: [3 /24]    Loss:  0.360806        Acc: 89.25%     Used time: 18.02799s
Train Epoch: [4 /24]    Loss:  0.018869        Acc: 94.57%     Used time: 18.63819s
Train Epoch: [5 /24]    Loss:  0.029784        Acc: 97.45%     Used time: 18.31936s
Train Epoch: [6 /24]    Loss:  0.000340        Acc: 99.11%     Used time: 18.54602s
Train Epoch: [7 /24]    Loss:  0.000201        Acc: 99.33%     Used time: 18.41860s
Train Epoch: [8 /24]    Loss:  0.000013        Acc: 99.89%     Used time: 18.44500s
Train Epoch: [9 /24]    Loss:  0.000374        Acc: 100.00%    Used time: 18.50257s
Train Epoch: [10/24]    Loss:  0.000464        Acc: 100.00%    Used time: 18.36819s
Train Epoch: [11/24]    Loss:  0.000052        Acc: 100.00%    Used time: 18.34564s
Train Epoch: [12/24]    Loss:  0.003684        Acc: 100.00%    Used time: 18.36702s
Train Epoch: [13/24]    Loss:  0.000051        Acc: 100.00%    Used time: 18.58212s
Train Epoch: [14/24]    Loss:  0.000650        Acc: 100.00%    Used time: 18.98961s
Train Epoch: [15/24]    Loss:  0.000357        Acc: 100.00%    Used time: 18.52834s
Train Epoch: [16/24]    Loss:  0.000058        Acc: 100.00%    Used time: 18.38217s
Train Epoch: [17/24]    Loss:  0.000060        Acc: 100.00%    Used time: 18.32131s
Train Epoch: [18/24]    Loss:  0.000193        Acc: 100.00%    Used time: 18.24259s
Train Epoch: [19/24]    Loss:  0.000034        Acc: 100.00%    Used time: 18.26554s
Train Epoch: [20/24]    Loss:  0.000133        Acc: 100.00%    Used time: 18.36593s
Train Epoch: [21/24]    Loss:  0.000900        Acc: 100.00%    Used time: 18.17084s
Train Epoch: [22/24]    Loss:  0.000035        Acc: 100.00%    Used time: 18.41482s
Train Epoch: [23/24]    Loss:  0.000206        Acc: 100.00%    Used time: 18.20566s
Train Epoch: [24/24]    Loss:  0.000089        Acc: 100.00%    Used time: 18.22978s
Test set: Average loss: 0.2295, Accuracy: 60.0%
```



可以看到先训练层数为24层的时候，损失函数和准确率都能够达到稳定，表明了24层时已经能够达到训练要求。

【注：次数的Acc是训练集的准确率】

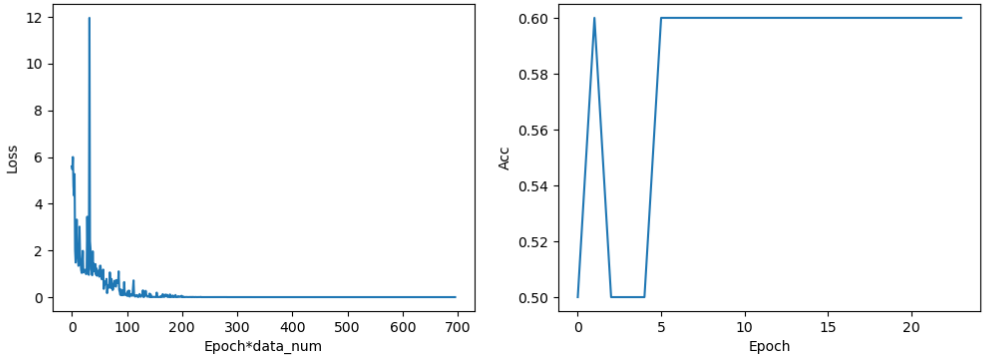
接下来每训练一轮就进行一次预测，观察每一轮的准确率：

【后续的测试以上述24x32的参数设计进行测试，以下的输出中Accuracy均表示每一轮测试后测试集的准确率】

第五次测试：

训练次数：24

```
Initial Loss: 5.600892
Train Epoch: [1 /24]    Loss:   3.443604        Accuaracy: 50.00%       Used time: 19.75912s
Train Epoch: [2 /24]    Loss:   1.180011        Accuaracy: 60.00%       Used time: 19.78126s
Train Epoch: [3 /24]    Loss:   0.306937        Accuaracy: 50.00%       Used time: 19.13249s
Train Epoch: [4 /24]    Loss:   0.022747        Accuaracy: 50.00%       Used time: 19.42876s
Train Epoch: [5 /24]    Loss:   0.000521        Accuaracy: 50.00%       Used time: 19.08903s
Train Epoch: [6 /24]    Loss:   0.102752        Accuaracy: 60.00%       Used time: 19.03504s
Train Epoch: [7 /24]    Loss:   0.000021        Accuaracy: 60.00%       Used time: 19.20930s
Train Epoch: [8 /24]    Loss:   0.004577        Accuaracy: 60.00%       Used time: 19.28483s
Train Epoch: [9 /24]    Loss:   0.001495        Accuaracy: 60.00%       Used time: 19.60127s
Train Epoch: [10/24]    Loss:   0.000077        Accuaracy: 60.00%       Used time: 19.25317s
Train Epoch: [11/24]    Loss:   0.000021        Accuaracy: 60.00%       Used time: 18.95115s
Train Epoch: [12/24]    Loss:   0.000011        Accuaracy: 60.00%       Used time: 18.98530s
Train Epoch: [13/24]    Loss:   0.000438        Accuaracy: 60.00%       Used time: 19.26959s
Train Epoch: [14/24]    Loss:   0.000007        Accuaracy: 60.00%       Used time: 19.29722s
Train Epoch: [15/24]    Loss:   0.000032        Accuaracy: 60.00%       Used time: 18.71414s
Train Epoch: [16/24]    Loss:   0.000175        Accuaracy: 60.00%       Used time: 18.69273s
Train Epoch: [17/24]    Loss:   0.000107        Accuaracy: 60.00%       Used time: 18.69157s
Train Epoch: [18/24]    Loss:   0.000323        Accuaracy: 60.00%       Used time: 18.68380s
Train Epoch: [19/24]    Loss:   0.000063        Accuaracy: 60.00%       Used time: 18.41901s
Train Epoch: [20/24]    Loss:   0.000550        Accuaracy: 60.00%       Used time: 19.03547s
Train Epoch: [21/24]    Loss:   0.000001        Accuaracy: 60.00%       Used time: 19.41139s
Train Epoch: [22/24]    Loss:   0.000043        Accuaracy: 60.00%       Used time: 19.21727s
Train Epoch: [23/24]    Loss:   0.000276        Accuaracy: 60.00%       Used time: 19.04504s
Train Epoch: [24/24]    Loss:   0.000503        Accuaracy: 60.00%       Used time: 19.07679s
```



发现虽然损失值loss已经趋于稳定，但是对测试集的预测效果却不好，稳定在了60%，说明了先前的测试出现80%和90%可能存在偶然性。

第六次测试：

将全连接层设置为3层时

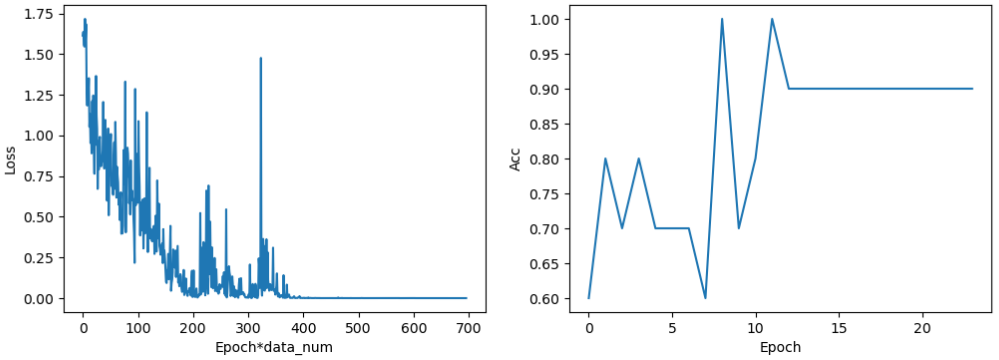
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN,self).__init__()
        self.conv1=nn.Sequential(
            nn.Conv2d(
                in_channels=3,
                #输出为16*226*226
                out_channels=16,
                kernel_size=3,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),#激活函数
            nn.MaxPool2d(kernel_size=2)#2x2的方块进行池化维度变为16*113*113
        )#
        self.conv2=nn.Sequential(
            nn.Conv2d(16,32,3,1,2),#16*113*123->32*115*115
```

```
            nn.ReLU(),
            nn.MaxPool2d(2)#32*115*115->32*57*57
        )
        self.fc1=nn.Linear(32*57*57,256)#32*57*57->5
        self.fc2=nn.Linear(256,32)
        self.fc3=nn.Linear(32,5)

    def forward(self,x):
        x=torch.relu(self.conv1(x))
        x=torch.relu(self.conv2(x))
        x=x.view(x.size(0),-1)
        x=self.fc1(x)
        x=self.fc2(x)
        x=self.fc3(x)
        return x
```

结果如下：

```
Initial Loss: 1.614029
Train Epoch: [1 /24]    Loss:   0.884985        Accuaracy: 60.00%       Used time: 19.81712s
Train Epoch: [2 /24]    Loss:   0.958412        Accuaracy: 80.00%       Used time: 19.09493s
Train Epoch: [3 /24]    Loss:   0.514289        Accuaracy: 70.00%       Used time: 18.93062s
Train Epoch: [4 /24]    Loss:   0.473199        Accuaracy: 80.00%       Used time: 19.17922s
Train Epoch: [5 /24]    Loss:   0.336053        Accuaracy: 70.00%       Used time: 18.85950s
Train Epoch: [6 /24]    Loss:   0.096063        Accuaracy: 70.00%       Used time: 18.90037s
Train Epoch: [7 /24]    Loss:   0.011749        Accuaracy: 70.00%       Used time: 19.76492s
Train Epoch: [8 /24]    Loss:   0.470084        Accuaracy: 60.00%       Used time: 19.64404s
Train Epoch: [9 /24]    Loss:   0.545800        Accuaracy: 100.00%      Used time: 19.30973s
Train Epoch: [10/24]    Loss:   0.030455        Accuaracy: 70.00%       Used time: 19.62170s
Train Epoch: [11/24]    Loss:   0.018862        Accuaracy: 80.00%       Used time: 18.97474s
Train Epoch: [12/24]    Loss:   0.052369        Accuaracy: 100.00%      Used time: 19.32078s
Train Epoch: [13/24]    Loss:   0.000879        Accuaracy: 90.00%       Used time: 19.53882s
Train Epoch: [14/24]    Loss:   0.001074        Accuaracy: 90.00%       Used time: 19.18425s
Train Epoch: [15/24]    Loss:   0.000625        Accuaracy: 90.00%       Used time: 19.73913s
Train Epoch: [16/24]    Loss:   0.003953        Accuaracy: 90.00%       Used time: 18.86161s
Train Epoch: [17/24]    Loss:   0.000006        Accuaracy: 90.00%       Used time: 18.82148s
Train Epoch: [18/24]    Loss:   0.000161        Accuaracy: 90.00%       Used time: 18.65720s
Train Epoch: [19/24]    Loss:   0.000042        Accuaracy: 90.00%       Used time: 18.79926s
Train Epoch: [20/24]    Loss:   0.000011        Accuaracy: 90.00%       Used time: 18.64511s
Train Epoch: [21/24]    Loss:   0.000011        Accuaracy: 90.00%       Used time: 18.61375s
Train Epoch: [22/24]    Loss:   0.000000        Accuaracy: 90.00%       Used time: 18.43729s
Train Epoch: [23/24]    Loss:   0.000018        Accuaracy: 90.00%       Used time: 18.73865s
Train Epoch: [24/24]    Loss:   0.000016        Accuaracy: 90.00%       Used time: 18.63912s
```

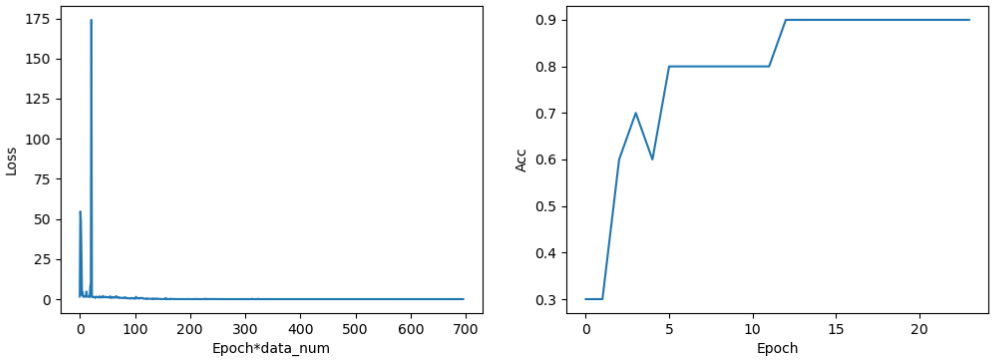


增加全连接层的层数后每一轮的训练时间略有增加，初始的损失值较低，最后的准确率也稳定在90%。

第七次测试：

增加卷积层得到的输出的通道数第一层16->64,第二层32->256，观察训练过程中的预测效果。

```
Initial Loss: 1.589155
Train Epoch: [1 /24]    Loss:   0.741914        Accuaracy: 30.00%       Used time: 79.46529s
Train Epoch: [2 /24]    Loss:   0.764255        Accuaracy: 30.00%       Used time: 77.54285s
Train Epoch: [3 /24]    Loss:   0.552909        Accuaracy: 60.00%       Used time: 79.25173s
Train Epoch: [4 /24]    Loss:   0.360540        Accuaracy: 70.00%       Used time: 72.68069s
Train Epoch: [5 /24]    Loss:   0.128004        Accuaracy: 60.00%       Used time: 70.22303s
Train Epoch: [6 /24]    Loss:   0.062340        Accuaracy: 80.00%       Used time: 72.49130s
Train Epoch: [7 /24]    Loss:   0.003526        Accuaracy: 80.00%       Used time: 74.74228s
Train Epoch: [8 /24]    Loss:   0.034104        Accuaracy: 80.00%       Used time: 70.87366s
Train Epoch: [9 /24]    Loss:   0.001455        Accuaracy: 80.00%       Used time: 70.33141s
Train Epoch: [10/24]    Loss:   0.001843        Accuaracy: 80.00%       Used time: 70.46871s
Train Epoch: [11/24]    Loss:   0.000640        Accuaracy: 80.00%       Used time: 70.16110s
Train Epoch: [12/24]    Loss:   0.000038        Accuaracy: 80.00%       Used time: 70.24132s
Train Epoch: [13/24]    Loss:   0.000542        Accuaracy: 90.00%       Used time: 70.33320s
Train Epoch: [14/24]    Loss:   0.000131        Accuaracy: 90.00%       Used time: 72.37281s
Train Epoch: [15/24]    Loss:   0.001977        Accuaracy: 90.00%       Used time: 70.20885s
Train Epoch: [16/24]    Loss:   0.000280        Accuaracy: 90.00%       Used time: 70.30303s
Train Epoch: [17/24]    Loss:   0.000054        Accuaracy: 90.00%       Used time: 70.52009s
Train Epoch: [18/24]    Loss:   0.000132        Accuaracy: 90.00%       Used time: 70.99238s
Train Epoch: [19/24]    Loss:   0.000971        Accuaracy: 90.00%       Used time: 71.05793s
Train Epoch: [20/24]    Loss:   0.000184        Accuaracy: 90.00%       Used time: 70.92406s
Train Epoch: [21/24]    Loss:   0.000100        Accuaracy: 90.00%       Used time: 70.31371s
Train Epoch: [22/24]    Loss:   0.000059        Accuaracy: 90.00%       Used time: 72.72639s
Train Epoch: [23/24]    Loss:   0.000128        Accuaracy: 90.00%       Used time: 82.92474s
Train Epoch: [24/24]    Loss:   0.000005        Accuaracy: 90.00%       Used time: 82.41700s
```

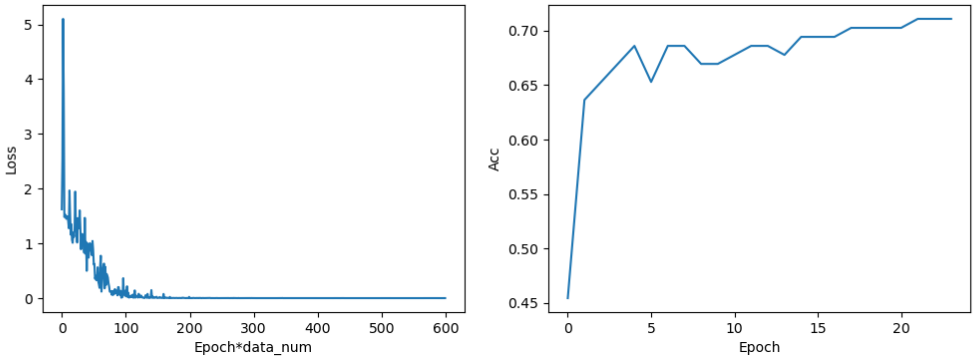


可以看到增加了卷积层的输出层数后每一轮训练所花费的时间大大增加了，最终准确率稳定在90%。

第八次测试：

接着将部分训练集数据放到测试集并进行训练和预测：

```
Initial Loss: 1.622704
Train Epoch: [1 /24]    Loss:   1.015582        Accuaracy: 45.45%       Used time: 13.51513s
Train Epoch: [2 /24]    Loss:   0.886859        Accuaracy: 63.64%       Used time: 13.09243s
Train Epoch: [3 /24]    Loss:   0.210749        Accuaracy: 65.29%       Used time: 12.98322s
Train Epoch: [4 /24]    Loss:   0.135501        Accuaracy: 66.94%       Used time: 13.25386s
Train Epoch: [5 /24]    Loss:   0.046283        Accuaracy: 68.60%       Used time: 13.36269s
Train Epoch: [6 /24]    Loss:   0.004104        Accuaracy: 65.29%       Used time: 13.39165s
Train Epoch: [7 /24]    Loss:   0.002755        Accuaracy: 68.60%       Used time: 13.61691s
Train Epoch: [8 /24]    Loss:   0.022073        Accuaracy: 68.60%       Used time: 13.40893s
Train Epoch: [9 /24]    Loss:   0.000840        Accuaracy: 66.94%       Used time: 13.55631s
Train Epoch: [10/24]    Loss:   0.000673        Accuaracy: 66.94%       Used time: 14.16982s
Train Epoch: [11/24]    Loss:   0.000330        Accuaracy: 67.77%       Used time: 13.54820s
Train Epoch: [12/24]    Loss:   0.000312        Accuaracy: 68.60%       Used time: 13.44806s
Train Epoch: [13/24]    Loss:   0.001193        Accuaracy: 68.60%       Used time: 13.50446s
Train Epoch: [14/24]    Loss:   0.000188        Accuaracy: 67.77%       Used time: 13.49104s
Train Epoch: [15/24]    Loss:   0.000400        Accuaracy: 69.42%       Used time: 13.43511s
Train Epoch: [16/24]    Loss:   0.000020        Accuaracy: 69.42%       Used time: 13.50904s
Train Epoch: [17/24]    Loss:   0.000349        Accuaracy: 69.42%       Used time: 13.60662s
Train Epoch: [18/24]    Loss:   0.000154        Accuaracy: 70.25%       Used time: 13.63124s
Train Epoch: [19/24]    Loss:   0.000115        Accuaracy: 70.25%       Used time: 13.55917s
Train Epoch: [20/24]    Loss:   0.000016        Accuaracy: 70.25%       Used time: 13.59252s
Train Epoch: [21/24]    Loss:   0.000599        Accuaracy: 70.25%       Used time: 13.79064s
Train Epoch: [22/24]    Loss:   0.000592        Accuaracy: 71.07%       Used time: 13.59199s
Train Epoch: [23/24]    Loss:   0.000039        Accuaracy: 71.07%       Used time: 13.44086s
Train Epoch: [24/24]    Loss:   0.000304        Accuaracy: 71.07%       Used time: 13.48584s
```



观察到增加测试集样例后准确率趋于70%，与第五次测试相比准确率更稳定。

综上，在以上的测试中，损失函数最终都能够收敛，训练24轮的CNN准确率能够达到60%，在进行增加全连接层的层数和增加卷积层输出的通道数确实能够提高CNN的效果，但是也会让时间增加。增加预测集的数据量，可以使最后的预测更有稳定性。

【由于电脑性能（未使用GPU），使用三层卷积-池化层进行训练运行速度大大降低，因此在这里不进行测试。以上图片存于result文件夹中，命名对应训练的次序】

四、 参考资料

[1][自定义的卷积神经网络模型CNN，对图片进行分类并使用图片进行测试模型-适合入门，从模型到训练再到测试，开源项目_卷积神经网络模型,实现图像的分类任务-CSDN博客(https://blog.csdn.net/qq_44757503/article/details/134143140)

[2]CNN.pdf

[3]week12 CNN.pdf