本科生实验报告

学生姓名：　丁晓琪

学生学号：　22336057

专业名称：　计科

一. 数据的提取

此处参考https://blog.csdn.net/weixin_41707744/article/details/104845856

1. CIFAR10数据集

- 概览：一共包含5个batch，一个测试集batch
- 每个batch包含10000张彩色三通道图像，大小为32*32
- 样本标签值：0-9，共10个

2. 数据的提取

- `load_CIFAR_batch(filename,option)`：提取一个batch内的样本的图像信息和标签。由于softmax线性分类器，MLP分类器和CNN分类器需要的数据的形状不同，当option==1时，图像信息为四维矩阵（样本数，通道数，长度，宽度），否则为二维矩阵（样本数，图像信息）

```
def load_CIFAR_batch(filename,option):
    with open(filename,'rb') as f:
        data_dict = p.load(f ,encoding='bytes')
        images = data_dict[b'data']
        labels = data_dict[b'labels']
        if option==1:
            images = images.reshape(10000, 3,32,32)
        else:
            images = images.reshape(10000, 3072)
        labels = np.array(labels)
        return images, labels
```

- `load_CIFAR_data(data_dir,option)`:提取出文件中五个训练batch的样本图像信息和标签，并将其拼为总的训练数据集，提取出测试集的样本图像信息和标签
  - 当option==1时，图像信息为四维矩阵（样本数，通道数，长度，宽度），否则为二维矩阵（样本数，图像信息）

```
# 完整读取数据的函数,通过多次调用批次读取数据的函数实现
def load_CIFAR_data(data_dir,option):
    images_train =[]
    labels_train = []
    for i in range(5):
        f = os.path.join(data_dir, 'data_batch_%d'%(i+1))
        print('loading', f)
        image_batch, label_batch = load_CIFAR_batch(f,option)
        images_train.append(image_batch)
        labels_train.append(label_batch)
        Xtrain = np.concatenate(images_train)
        Ytrian = np.concatenate(labels_train)
        del image_batch, label_batch
    Xtest, Ytest = load_CIFAR_batch(os.path.join(data_dir,
'test_batch'),option)
    return Xtrain, Ytrian, Xtest, Ytest
```

二. Pytorch使用基础

1. 数据处理和加载

- **Dataset**：表示数据集的接口，后面可以用Dataloader加载方便训练时样本的加载，以下是自定义数据集需要完成的操作
  - 需要自定义数据集类，并且继承pytorch的标准化数据集接口Dataset
  - 重写方法`__init __`: 对传入数据集中的数据和对数据的初始化处理，这里传入提取出来的样本图像信息和标签，将类型转为张量,并且对样本的图像信息归一化

```
    def __init__(self,X,Y):
        self.X = torch.from_numpy(X).float() / 255   #归一化到0-1
        self.Y = torch.from_numpy(Y).long()
```

  - 重写方法`__len__`: 获取数据集长度

```
    def __len__(self):#数据集中样本总数
        return len(self.X)
```

  - 重写方法`__getitem__`:根据下标获取数据集中对应的元素和标签

```
    def __getitem__(self,idx):
        return self.X[idx],self.Y[idx]
```

- **Dataloader:** 数据加载器，能够训练时自动加载出每一批次的数据和提供更多对批次中的数据的自定义选项
  - 参数：`dataset`数据集，`batch_size`每个批次要加载的样本数，`shuffle`是否在每次训练迭代重新排列数据
  - 本次实验的设置: `Test_loader`，由于在测试验证模型是在整个测试集上验证样本的正确性，所以测试集的数据加载器可以定义为一个批次里面包含整个整个数据集

```

Train_loader=DataLoader(dataset=Train_data,batch_size=100,shuffle=True)

Test_loader=DataLoader(dataset=Test_data,batch_size=10000,shuffle=True)
```

2. 神经网络的搭建

- 步骤:

```
class SoftmaxClassifier(nn.Module):
    def __init__(self,input_dim,num_labels):
        super(SoftmaxClassifier, self).__init__()
        self.linear=nn.Linear(input_dim,num_labels)

    def forward(self,x):
        return self.linear(x)
```

1. 创建神经网络构架类，需要继承基类`torch.nn.Module`
2. 在类的`__init__`方法中初始化神经网络的所有组件，需要调用`super().__init()__`继承`nn.Module`的初始化方法
3. 在`forward`函数中定义网络的前向传播法，input为网络输入，return为网络输出

- 基本网络组件：
  - `nn.Linear(input_dim,output_dim, bias)`：线性层，参数分别为输入的维度，输出的维度，偏置因子。$Y = XW + b$
  - `nn.ReLU()`：激活函数层，$ReLU(x) = max(0, x)$
  - `nn.Conv2d(in_channnels,out_channels,kernel_size,stride,padding)`：卷积层
    - `in_channels`：输入的通道数，实验中为3，数据集为RGB彩色图像
    - `out_channels`：输出的通道数，卷积核数
    - `kernel_size`：卷积核的大小
    - `stride`：卷积窗口移动步长
    - `padding`：填充0的行/列数
  - `nn.MaxPool2d(kernel_size)`：池化层
    - `kernel_size`：每kernel*kernel的窗口，选出最大的元素作为新的特征矩阵的组成元素
    - 窗口步长：默认为kernel_size

3. 神经网络的训练

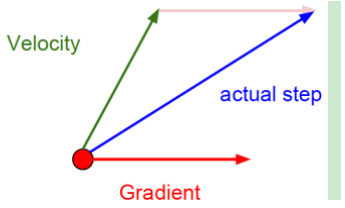
- 实例化神经网络模型
- 定义损失函数：本次实验中都选择`nn.CrossEntropyLoss(input,target)`J交叉熵损失
  - 交叉熵计算公式：$H(label, predict) = -\sum_{k=1}^{N}(label_k * logpredict_k)$
  - `input`：直接是整数索引，而不是独热编码。独热编码会在内部计算中完成
- 定义优化器：
  - `optim.SGD(model.parameters(),lr=)`：SGD随机梯度下降

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - lr * \nabla f(\boldsymbol{w}_t)$$

  - `optim.SGD(model.parameters(),lr=,Momentum=)`：SGD+Momentum动量

$$\boldsymbol{v}_t = \rho \boldsymbol{v}_{t-1} + \nabla f(\boldsymbol{w}_t)$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - lr \cdot \boldsymbol{v}_t$$



- `optim.Adam(model.parameters(),lr=)`：`betas (Tuple[float, float], optional)`：用于计算梯度的平均和平方的系数(默认: (0.9, 0.999))

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1)\nabla f(w_t)$$

$$s_t = \beta_2 \cdot s_{t-1} + (1-\beta_2)\left(\nabla f(w_t)\right)^2$$

$$w_{t+1} = w_t - lr \cdot m_t \oslash \sqrt{s_t}$$

- 权重更新步骤：

```
        optimizer.zero_grad() #清除梯度
        output=softmax_model(b_x)  #输入每一批次到模型，自动调用forward
        loss=loss_func(output,b_y) #计算损失函数
        loss.backward() #损失函数的反向传播
        optimizer.step() #参数优化
```

三. Softmax 线性分类器

1. 理论背景

- 对样本X为标签i的预测概率：

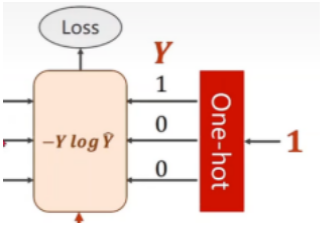
$$f_i(x) = softmax_i(xW) = \frac{e^{xw_i}}{\sum_{k=1}^{K} e^{xw_k}}$$

w_k是行向量

$$\begin{bmatrix} 0.2 & 1.2 \\ 0.3 & 0.2 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 2 \\ 0.3 \end{bmatrix}$$

w　　x

- 需要对真实标签读热向量转化



- Loss函数：

$$L(w_1, w_2, \ldots w_k) = -\frac{1}{N}\sum_{l=1}^{N}\sum_{k=1}^{K} y_k^{(l)} log[softmax_k(x^l W)]$$

- 梯度下降：

$$= -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w_k}\sum_{i=1}^{n}\sum_{j=1}^{C}y_i^{(j)}(w_j^T x_i - \ln\sum_{q=1}^{C}e^{w_q^T x_i})$$

$$= -\frac{1}{n}\sum_{i=1}^{n}(y_i^{(k)}x_i - \frac{e^{w_k^T x_i}x_i}{\sum_{q=1}^{C}e^{w_q^T x_i}})$$

$$= -\frac{1}{n}\sum_{i=1}^{n}x_i(y_i^{(k)} - \frac{e^{w_k^T x_i}}{\sum_{q=1}^{C}e^{w_q^T x_i}})$$

所以

$$\frac{\partial \boldsymbol{L}(w,x)}{\partial W} = [\frac{\partial \boldsymbol{L}(w,x)}{\partial w_1}, \frac{\partial \boldsymbol{L}(w,x)}{\partial w_2}, ..., \frac{\partial \boldsymbol{L}(w,x)}{\partial w_C}]$$

$$= -\frac{1}{n}\sum_{i=1}^{n}x_i(y_i - f(x_i))^T$$

2.模型结构

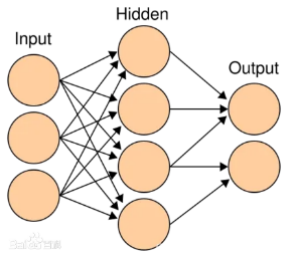
- 一个全连接层，一个softmax激活函数层
- 训练时使用的函数是交叉熵函数：隐含完成独热编码，比较模型预测的概率分布和真实标签的概率分布，并计算两者之间的差异，

```
# softmax线性分类器
class SoftmaxClassifier(nn.Module):
    def __init__(self,input_dim,num_labels):
        # function: 初始化网络结构,一个全连接层一个softmax激活函数层
        # input:input_dim，样本的总特征数；num_labels:样本的标签数
        super(SoftmaxClassifier, self).__init__()
        self.linear=nn.Linear(input_dim,num_labels)
        self.softmax=nn.Softmax(dim=1)

    def forward(self,x):
        # function:前向传播函数
        x=self.linear(x)
        output=self.softmax(x)
        return output
```

四. MLP分类器

1.理论背景



- MLP：有多层隐藏层的全连接网络层，每层线性层后面伴随着一个激活层
- MLP分类器: 第一层的输入维度为样本特征的个数，最后一层的输出维度为样本标签个数
- 预测样本标签的方法：将样本特征值输入网络，预测标签为输出值最大的索引

2. 模型结构

- 可增添或减少隐藏层数
- 第一层线性层：输入维度为样本特征数，输出维度为2048
- 第二层隐藏线性层：输入维度为2048，输出维度为1024
- 第三层线性输出层：输入维度为1024，输出维度为标签数为10
- 激活层：ReLU（max(0,x)）

```
class MLPClassifier(nn.Module):
    def __init__(self,input_dim,num_labels):
        super(MLPClassifier,self).__init__()
        self.linear1=nn.Linear(input_dim,2048)
        self.linear2=nn.Linear(2048,1024)
        self.linear3=nn.Linear(1024,10)
        self.relu=nn.ReLU()

    def forward(self,x):
        x=self.linear1(x)
        x=self.relu(x)
        x=self.linear2(x)
        x=self.relu(x)
        output=self.linear3(x)
        return output
```

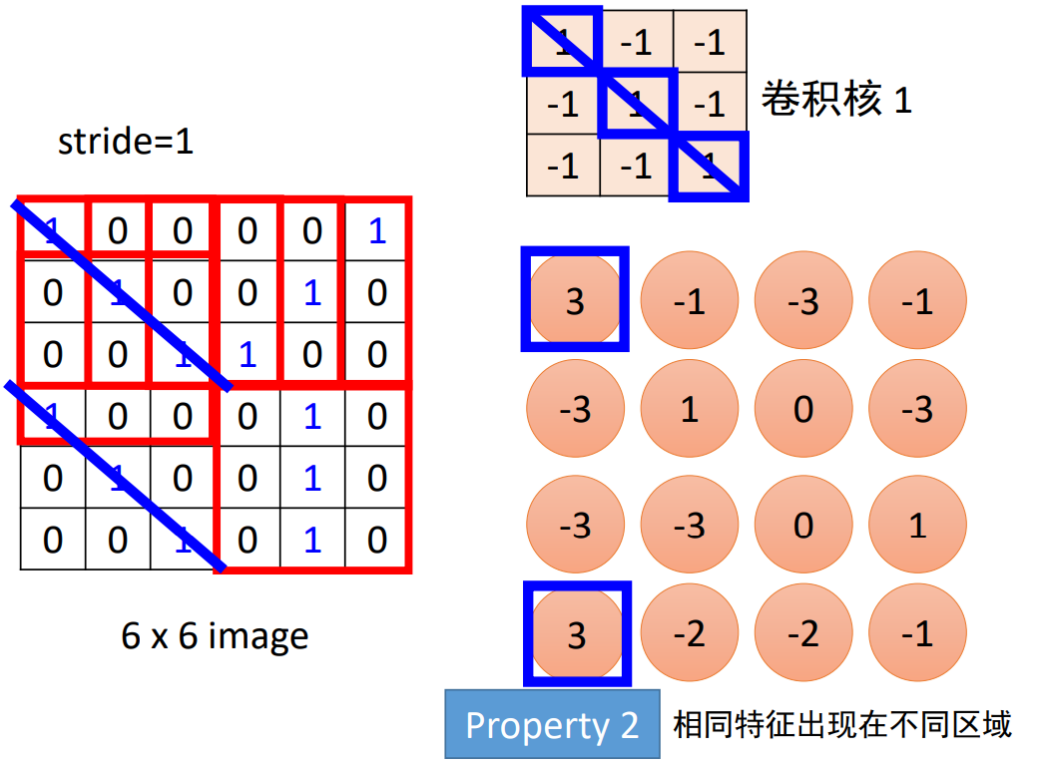
五. CNN分类器

1. 理论背景

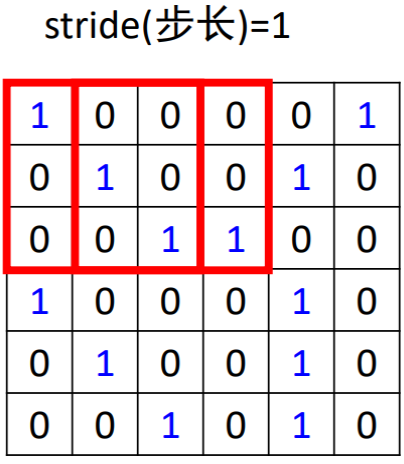
1. **卷积层**

- 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



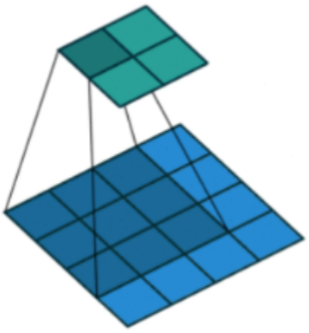
- 实现：基于相同特征可能会出现再不同区域的原则，所以要对图片矩阵的各个部分与卷积核做卷积操作得到新的矩阵
- 步长(stride)：在卷积操作中滑动卷积核的步幅，当步长较大时，卷积核在输入数据上滑动的速度会更快，这会导致输出特征图的尺寸减小。较大的步长可以减少模型的计算复杂度和内存消耗，但可能会丢失一些细节信息



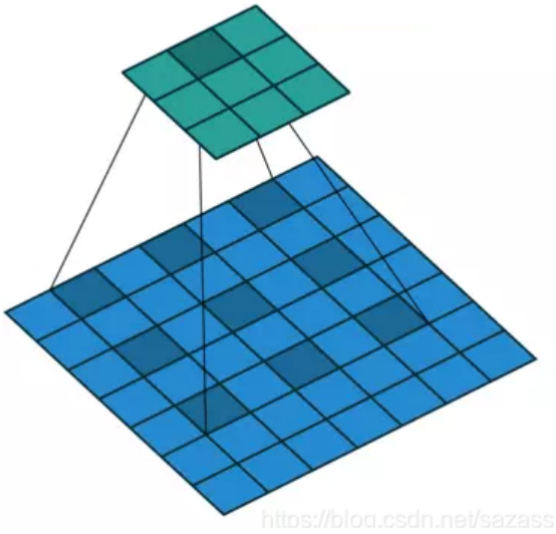
- 填充(Padding)：在输入图像的周围增加额外的像素值，以扩大输入图像的尺寸。由于在卷积操作中，卷积出来的特征矩阵的尺寸会缩小，为了更好地处理图像边缘信息，控制输出特征图的大小，需要Padding在图像边缘填充0或其他来调整

- 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

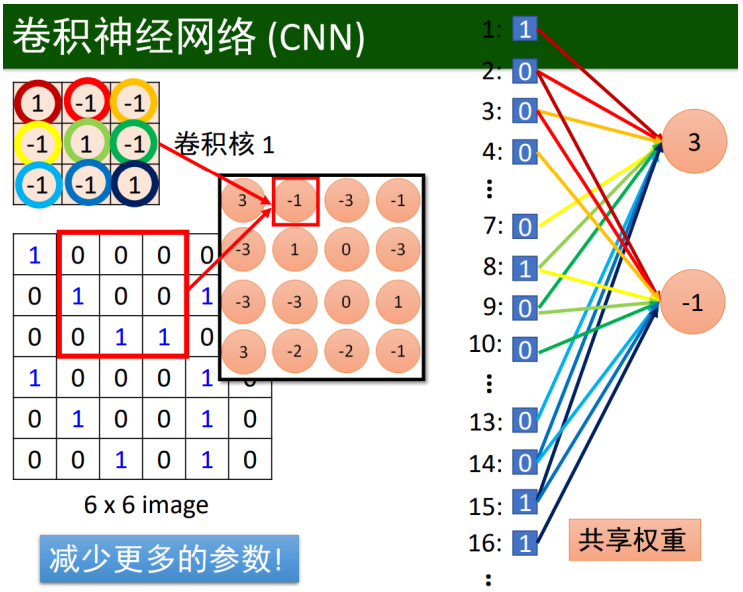
（1）dilation=0的话，效果如图：



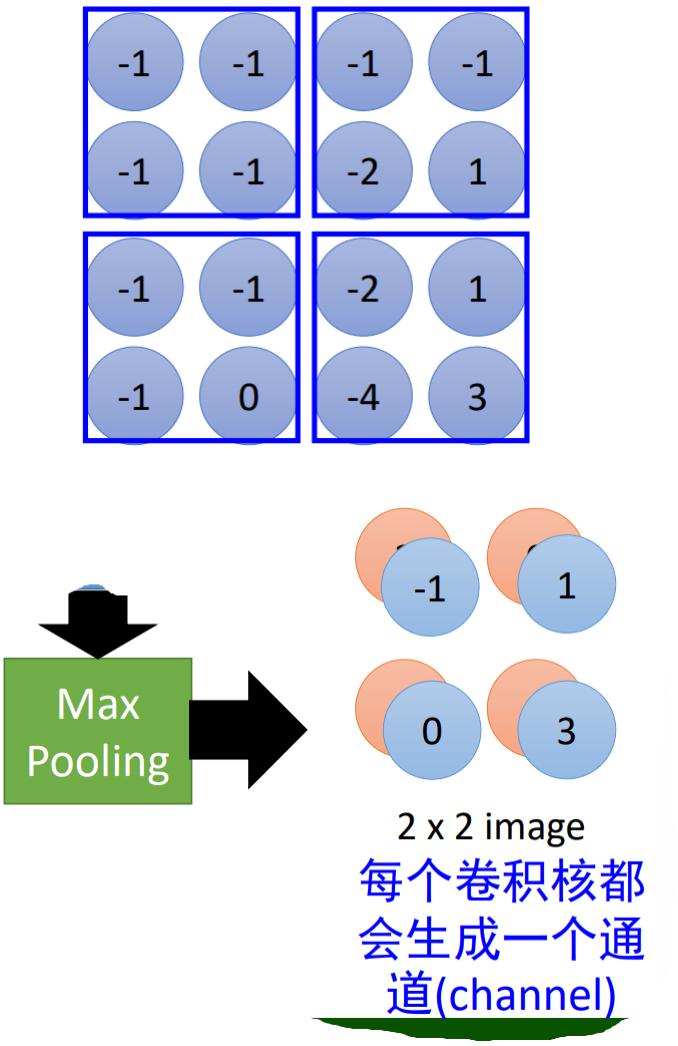
（2）dilation=1，那么效果如图：
称为扩张卷积（也叫空洞卷积）



- 综合：



- 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

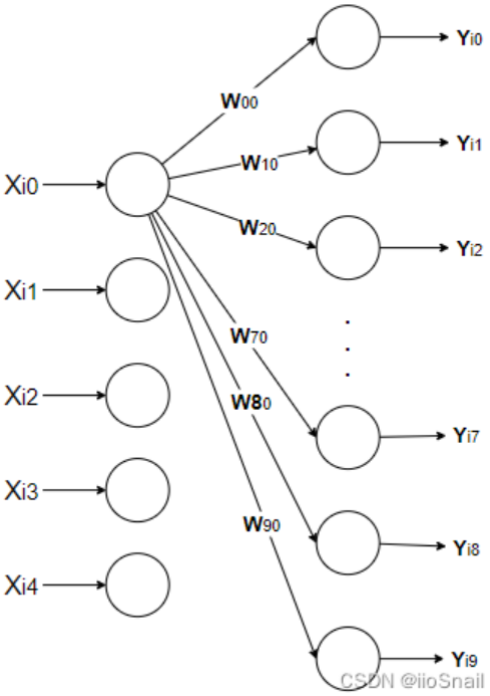


2. **全连接层**

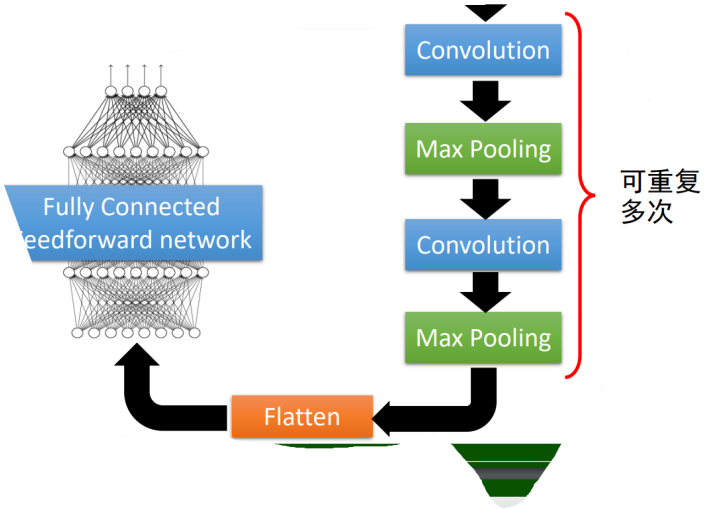
- 输入：经过卷积层提取特征和减少参数后的输入
- 输出：对输入的线性变化，在实验中是要对重要图片分类，所以这里的输出是有5个元素的向量，表征各种种类的概率（未归一化）。输出的数值越大，表示属于对应种类的概率越大

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

- 结构：



3. **总体网络结构**



2. 模型结构

- 包含两个卷积层，最大池化层，全连接层

```
class CNNClassifier(nn.Module):
    def __init__(self):
        # function:初始化网络结构
        super(CNNClassifier,self).__init__()#类继承的初始化
        # 卷积层1
        self.conv1=nn.Sequential( #输入 3*32*32
            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 (32-5+4)/1+1=32,16*32*32
            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*16*16
        )
```

```
        # 卷积层2
        self.conv2=nn.Sequential(
            nn.Conv2d(16,32,5,1,2),
            nn.ReLU(),
            nn.MaxPool2d(2),#输出 32*8*8
        )
        # 两层全连接层
        self.linear1=nn.Linear(32*8*8,1024)
        self.linear2=nn.Linear(1024,10)

    def forward(self,x):
        # function: 前向传播函数
        x=self.conv1(x)
        x=self.conv2(x)
        x=x.view(x.size(0),-1)  #保留第一维度，剩下的都压成一维
        x=self.linear1(x)
        output=self.linear2(x)
        return output
```

六. 模型训练和结果分析

1. 训练流程

统一调用函数`train_test`进行训练和测试模型

1. 根据变量`choice`选择和实例化模型，统一定义损失函数为交叉熵函数，根据变量`option,lr,momentum,betas`选择不同优化器和优化器的超参数设置
2. 模型训练: 一共迭代`epoches`。每次迭代过程中，从训练数据加载器`Train_loader`加载出所有批次，前面传播，反向传递更新模型参数，每次迭代结束在测试集上验证一次模型的正确率。每次迭代都要记录下loss和accuracy

```
def
train_test(choice,Train_loader,Test_loader,option,lr=0.0005,momentum=0.9,bet
as=(0.9,0.999),EPOCHES=100):

    # function:训练测试Softmax线性分类器
    # input:
    #  -choice: 0为softmax线性分类器，1为MLP分类器，2为CNN分类器
    #  -Train_loader,Test_Loder: 训练数据和测试数据加载器
    #  -option：选择优化器，0:SGD；1:SGD+momentum超参数；2:ADam
    #  -lr:学习率
    #  -momentum：SGD+momentum的超参数
    #  -betas: Adam的超参数
    #  -EPOCHES：训练时迭代次数

    #1. 定义模型，损失函数优化器
    if choice==0:
        model=SoftmaxClassifier(3072,10)
    elif choice==1:
        model=MLPClassifier(3072,10)
    elif choice==2:
        model=CNNClassifier()
    loss_func=nn.CrossEntropyLoss()

```

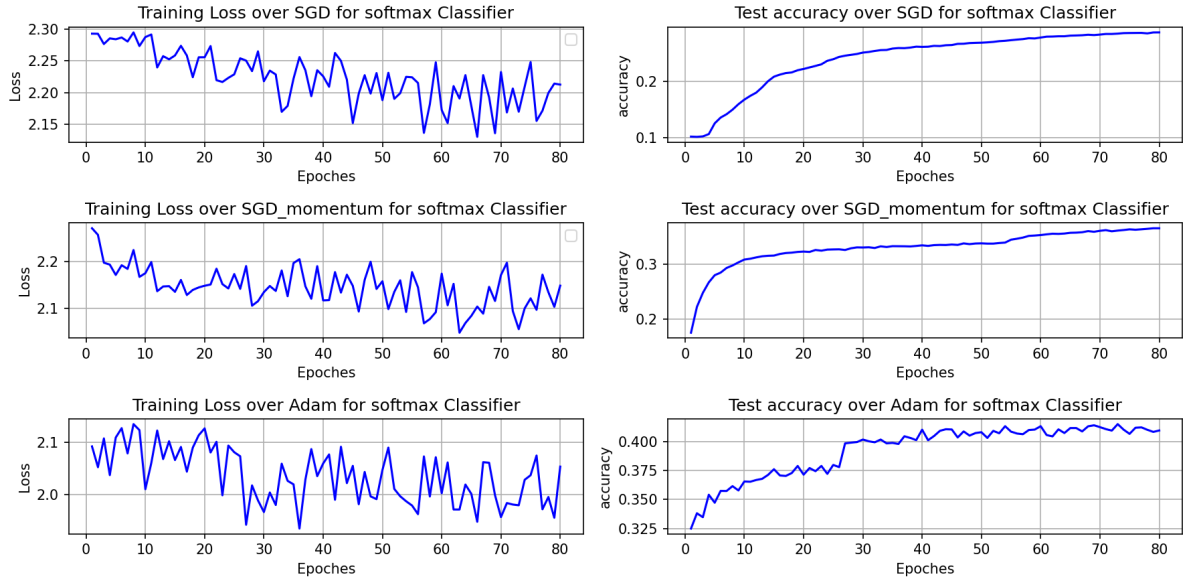
```
    if option==0:
        optimizer=optim.SGD(model.parameters(),lr=lr)
    elif option==1:
        optimizer=optim.SGD(model.parameters(),lr=lr,momentum=momentum)
    else:
        optimizer=optim.Adam(model.parameters(),lr=lr,betas=betas)
    model = model.to(device)#gpu训练
    result={'loss':[],'accuracy':[]}
    #2.迭代训练
    for epoch in range(EPOCHES):
     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
        optimizer.zero_grad() #清除梯度
        b_x = b_x.to(device)  #torch.Size([64, 3072])
        b_y = b_y.to(device)  #torch.Size([64])
        output=model(b_x)  #输入每一批次到模型，自动调用forward
        loss=loss_func(output,b_y) #计算损失函数
        loss.backward() #损失函数的反向传播
        optimizer.step() #参数优化
     #3. 每次迭代结束后都在测试集上测试
     for step2,(test_x,test_y) in enumerate(Test_loader): #一个批次弄完
                test_x = test_x.to(device)
                test_y = test_y.to(device)
                test_output = model(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))
                print('Epoch: ', epoch, '| train loss: %.4f' % loss.data.detach().cpu().numpy(), '| test accuracy: %.2f' % accuracy)
                result['accuracy'].append(accuracy)
                result['loss'].append(loss.data.detach().cpu().numpy())
    return result
```

2. 参数，超参数设置

- 网络的参数是直接采用默认的初始化方法，由查询资料可得`nn.Linear`和`nn.Conv2D`的参数的默认初始化，都是在 [-limit, limit] 之间的均匀分布（Uniform distribution），其中 limit 是`1. / sqrt(fan_in)`，`fan_in`是指参数张量（tensor）的输入单元的数量
- 超参数设置：
  - 学习率：选择优化器的默认学习率0.001时发现会有可能会出现震荡现象，则改为0.0005
  - 训练时迭代次数：在初次实验训练中发现SGD优化器和SGD-momentum优化器的收敛速度慢，需要比ADam多的训练迭代次数，而使用Adam优化器则很快收敛
  - SGD-momentum优化器的momentum设置：采用常见的0.9
  - Adam优化器的两个超参数betas的设置：采用默认也是常见的0.9和0.999的组合

3. Softmax线性分类器实验结果分析

为了方便比较不同优化器的表现，这里训练时统一迭代80次

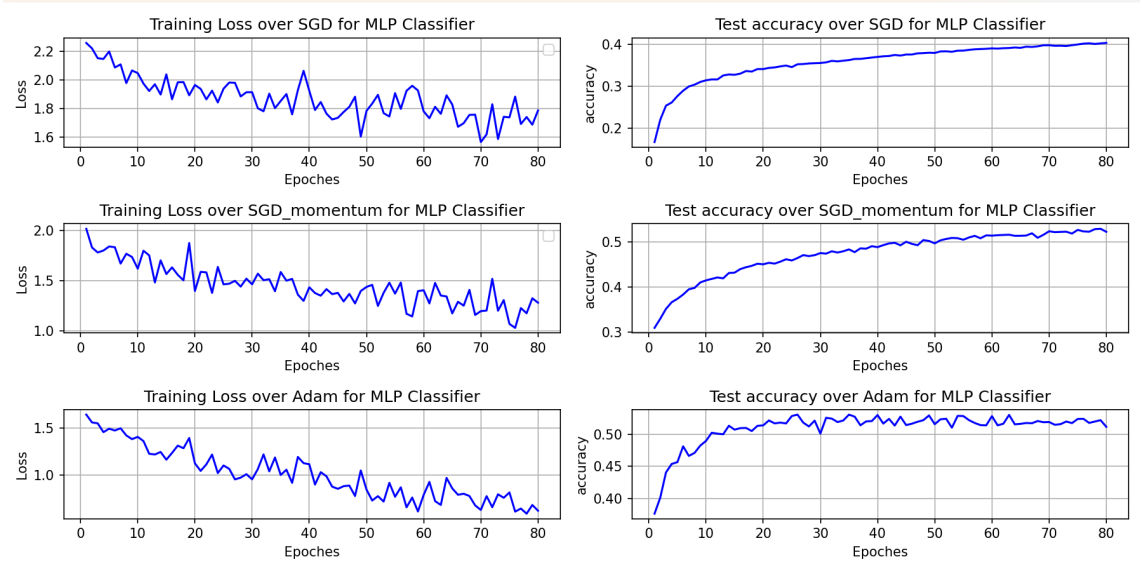


	SGD	SGD-momentum	Adam
最高正确率	0.29	0.36	0.41
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第40次迭代时收敛
loss下降情况	总体趋势为下降	总体趋势为下降	波动较大，后面达到收敛后开始震荡

4. MLP分类器实验结果及其分析

- 模型为两层结构时：

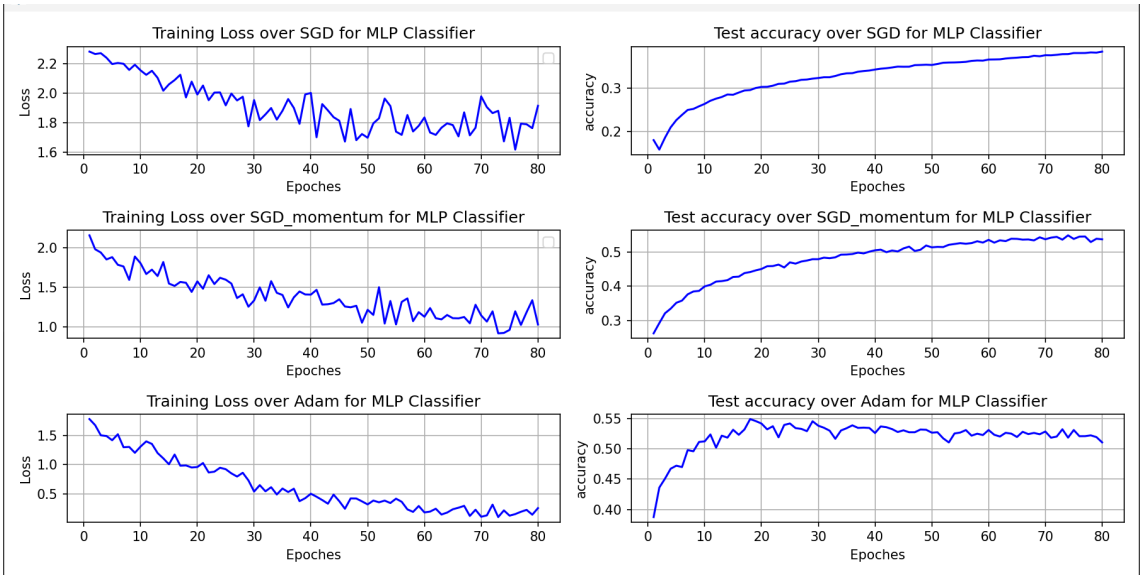
```
        self.linear1=nn.Linear(input_dim,1024)
        #self.linear2=nn.Linear(2048,1024)
        self.linear2=nn.Linear(1024,10)
        self.relu=nn.ReLU()
```



	SGD	SGD-momentum	Adam
最高正确率	0.40	0.53	0.53
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第24次迭代时收敛
loss下降情况	总体趋势为下降，但是后期有较大波动	总体趋势为下降	总体趋势为下降

- 模型结构为三层网络时

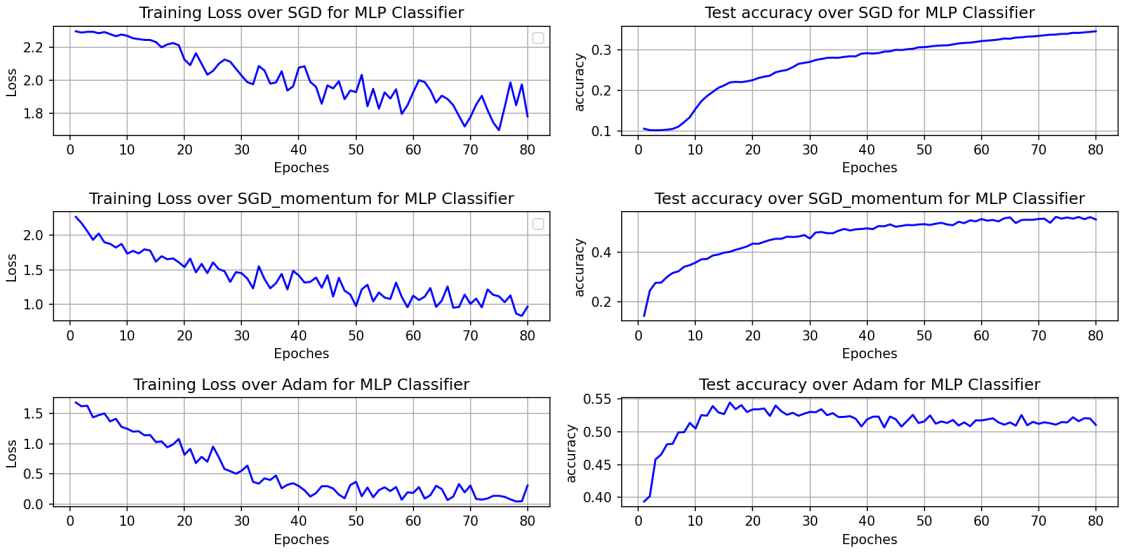
```
        self.linear1=nn.Linear(input_dim,2048)
        self.linear2=nn.Linear(2048,1024)
        self.linear3=nn.Linear(1024,10)
        self.relu=nn.ReLU()
```



	SGD	SGD-momentum	Adam
最高正确率	0.38	0.55	0.55
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第17次迭代时收敛
loss下降情况	总体趋势明显为下降，但是后期有较大波动	总体趋势明显为下降	总体趋势明显为下降，且波动较小

- 使用模型为四层网络

```
        self.linear1=nn.Linear(input_dim,2048)
        self.linear2=nn.Linear(2048,1024)
        self.linear3=nn.Linear(1024,500)
        self.linear4=nn.Linear(500,10)
        self.relu=nn.ReLU()
```



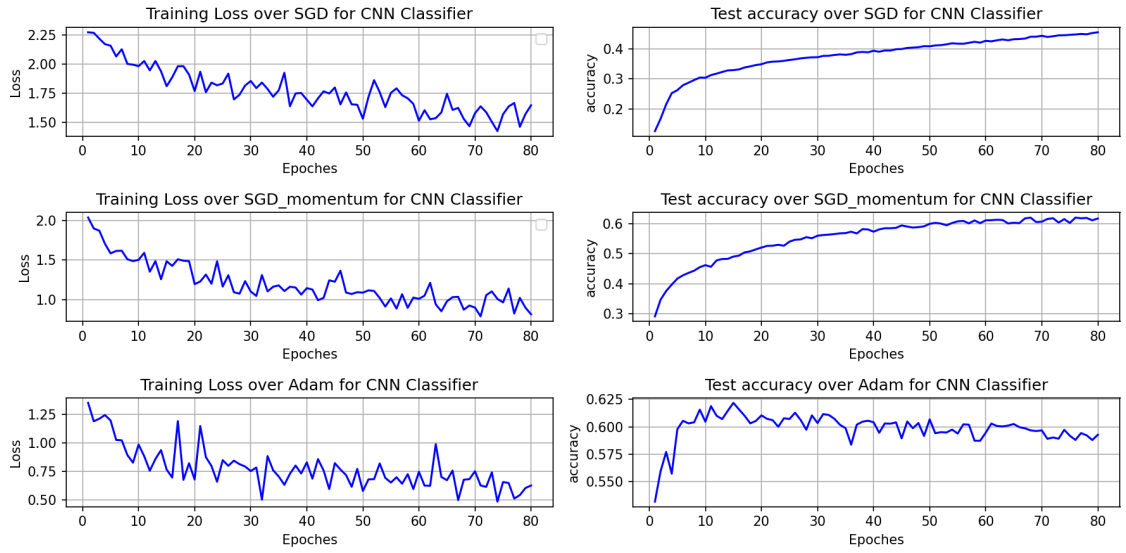
	SGD	SGD-momentum	Adam
最高正确率	0.36	0.54	0.55
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第12次迭代时收敛，在约15次正确率开始下降，训练迭代次数过多，导致过拟合
loss下降情况	总体趋势明显为下降	总体趋势明显为下降	总体趋势明显为下降，且波动较小

5. CNN分类器实验结果分析

- 模型结构：一层卷积层（后跟一层池化层），两层全连接层

```
        # 卷积层1
        self.conv1=nn.Sequential( #输入 3*32*32
            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 (32-5+4)/1+1=32,16*32*32
            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*16*16
        )

    self.linear1=nn.Linear(16*16*16,1024)
        self.linear2=nn.Linear(1024,10)
```

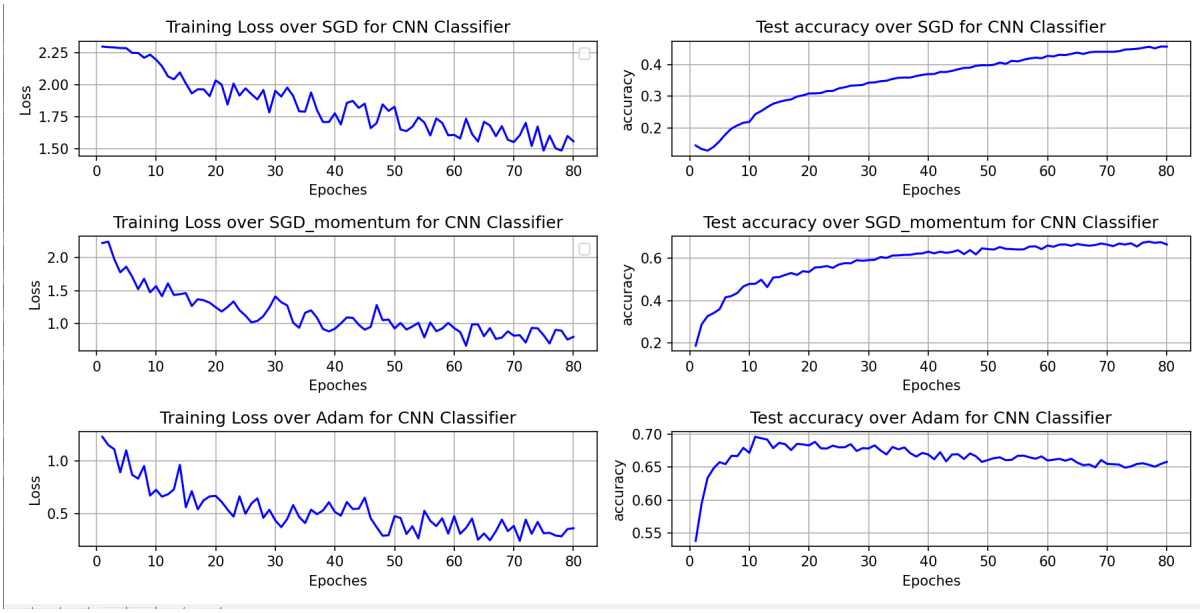


	SGD	SGD-momentum	Adam
最高正确率	0.46	0.62	0.62
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第10次迭代时收敛，在约15次正确率开始下降，训练迭代次数过多，导致过拟合
loss下降情况	总体趋势明显为下降	总体趋势明显为下降	总体趋势明显为下降

- 模型结构：两次卷积层（每层后面跟有一层池化层），两层全连接层

```
        # 卷积层1
        self.conv1=nn.Sequential( #输入 3*32*32
```

```
            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 (32-5+4)/1+1=32,16*32*32
            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*16*16
        )
        # 卷积层2
        self.conv2=nn.Sequential(
            nn.Conv2d(16,32,5,1,2),
            nn.ReLU(),
            nn.MaxPool2d(2),#输出 32*8*8
        )
        # 两层全连接层
        self.linear1=nn.Linear(32*8*8,1024)
        self.linear2=nn.Linear(1024,10)
```

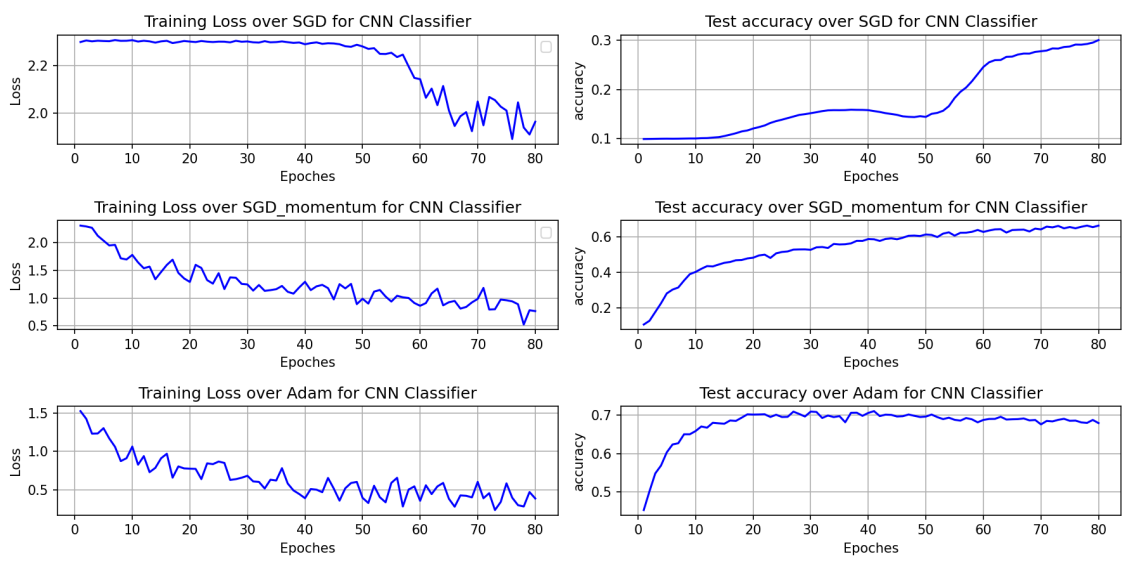


	SGD	SGD-momentum	Adam
最高正确率	0.46	0.68	0.70，正确率在第十次后有所下降，考虑训练迭代次数过多导致过拟合
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第10次迭代时收敛
loss下降情况	总体趋势明显为下降	总体趋势明显为下降	总体趋势明显为下降，且总体的loss值比其他优化器低

- 模型结构：3个卷积层，两个全连接层

```
        # 卷积层1
        self.conv1=nn.Sequential( #输入 3*32*32
```

```
            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 (32-5+4)/1+1=32,16*32*32
            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*16*16
        )
        # 卷积层2
        self.conv2=nn.Sequential(
            nn.Conv2d(16,32,5,1,2),
            nn.ReLU(),
            nn.MaxPool2d(2),#输出 32*8*8
        )
        #卷积层3
        self.conv3=nn.Sequential(
            nn.Conv2d(32,16,5,1,2),
            nn.ReLU(),
            nn.MaxPool2d(2),#输出16*4*4
        )
        # 两层全连接层
        self.linear1=nn.Linear(16*4*4,100)
        #self.linear1=nn.Linear(32*8*8,1024)
        self.linear2=nn.Linear(100,10)
```



	SGD	SGD-momentum	Adam
最高正确率	0.30	0.66	0.71，正确率在约第50次后有所下降，考虑训练迭代次数过多导致过拟合
收敛情况	80次迭代仍未收敛	80次迭代仍未收敛	在约第26次迭代时收敛

	SGD	SGD-momentum	Adam
loss下降情况	总体趋势明显为下降，一开始下降趋势较为缓慢	总体趋势明显为下降	总体趋势明显为下降，且总体的loss值比其他优化器低

6. 总结

- Adam优化器表现最佳，无论在什么分类器上，收敛速度远远快于其他优化器，且训练模型正确率高；SGD优化器表现最差，无论在什么分类器上，收敛最慢，模型正确率较低；SGD-momentum表现一般，收敛速度比SGD快，但还是远低于Adam优化器
- 在MLP中，网络层数越多且隐藏层包含的因子越多，模型的正确率会有所提高，且训练过程中收敛的速度会更快一点，但是每次迭代的训练时间会变长
- 在CNN中卷积层数在一定范围内增加，能够提高模型的正确率，卷积层数过多时，正确率提升不大，而且训练时长加长，两层卷积层在正确率和开销上比较合适
- CNN分类器的表现优于Softmax线性分类器，优于MLP分类器，在训练过程中最早达到收敛，在测试数据集上正确率最高，但是训练时间较长
- MLP分类器的表现优于Softmax线性分类器，收敛速度快，正确率高