实验三：CIFAR-10图像分类

18340238 邹雨桐

1 算法原理

1.1 线性分类器

线性分类器即Softmax分类，具体原理已在实验一中叙述，这里不再赘述.

1.2 多层感知机

多层感知机(MLP)是由感知机(PLA)推广而来，实际上就是全连接层，其一般有多个神经元层，层与层之间是全连接的，通常由输入层-隐藏层-输出层组成. 多层感知机通常放在复杂神经网路的最后一层做分类，但该任务本身就是一个分类任务，所以可以直接用多层感知机.

多层感知机中的神经元通常有激活函数，常见的激活函数有Sigmoid和Tanh.

1.3 卷积神经网络

1.3.1 卷积层

在图像分类中，全连接层的每一层网络都和相邻层全部连接，这样并不能考虑到图像的像素分布的位置信息，所以我们引入卷积层. 对图像的卷积通常是二维卷积，不妨设输入为$\boldsymbol{X} \in \mathbb{R}^{M \times N}$，卷积核为$\boldsymbol{W} \in \mathbb{R}^{K \times K}$，那么卷积过程为:

$$s_{ij} = \sum_m \sum_n x_{i+m,j+n} w_{mn}.$$

通常，卷积层之后需要激活神经元，常用的激活函数是ReLU函数.

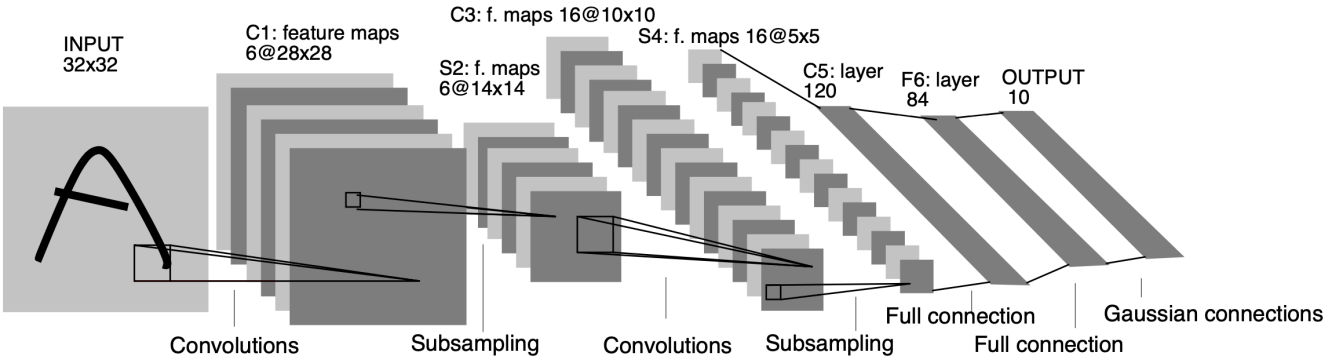
1.3.2 池化层

卷积层通常会将图像特征映射到高维的空间，因此参数量很大，为了减少计算量，我们可以采用池化层提取特征，并且还可以提升模型的稳定性.

常见的池化有最大池化与平均池化，根据前人的经验，最大池化的效果往往比平均池化好. 对于最大池化，即在某个固定的滑动二维窗口中，选取最大的值.

1.3.3 LeNet-5

本次实验中卷积神经网络的baseline是LeNet-5，其结构如下:



2 训练过程

2.1 数据预处理

本次实验的数据预处理非常简单，因为读入的数据已经是NumPy数组，因此只需要进行一些简单的处理.

- 对于Softmax分类器与MLP，输入的数据应该是一维的，但图像是三维的(包括三个通道)，所以将图像拼接起来就可以了.

```
data = data.reshape(len(data), -1)
```

- 本实验在数据预处理中还采用了两个非常重要的技巧：Shuffle与Mini Batch. 前者与Mini Batch结合使用可以使训练集的数据分布更均匀，避免学习到和数据分布相关的特征，降低模型的的泛化性；较大的batch size可以加快模型收敛，提高内存的利用率，同时迭代次数较少，梯度下降的方向较准确，但full batch size可能会出现大数据集内存不够用的现象，所以batch size的大小应该适中.

2.2 优化方法

本实验将对比三种优化方法的性能，分别是SGD，SGD Momentum，Adam.

2.3 训练框架

本实验采用PyTorch框架.

2.4 参数初始化

PyTorch的Module自带默认的初始化方法，可以在库源码的`linear.py`与`conv.py`中的`reset_parameters`函数中看到：

```
def reset_parameters(self) -> None:
    init.kaiming_uniform_(self.weight, a=math.sqrt(5))
```

默认的初始化方法是`kaiming_uniform_`.

除了默认的初始化方法以外，本实验还在MLP中对比了Xavier初始化与均匀分布初始化的效果：

```python
def init_linears(self, method):
    if(method == 'xavier'):
        nn.init.xavier_uniform_(self.linear1.weight)
        nn.init.zeros_(self.linear1.bias)
        nn.init.xavier_uniform_(self.linear2.weight)
        nn.init.zeros_(self.linear2.bias)
        nn.init.xavier_uniform_(self.linear3.weight)
        nn.init.zeros_(self.linear3.bias)
    elif(method == 'normal'):
        nn.init.normal_(self.linear1.weight)
        nn.init.normal_(self.linear1.bias)
        nn.init.normal_(self.linear2.weight)
        nn.init.normal_(self.linear2.bias)
        nn.init.normal_(self.linear3.weight)
        nn.init.normal_(self.linear3.bias)
        nn.init.normal_(self.linear4.weight)
        nn.init.normal_(self.linear4.bias)
```

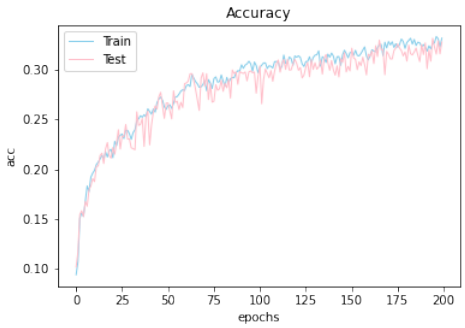
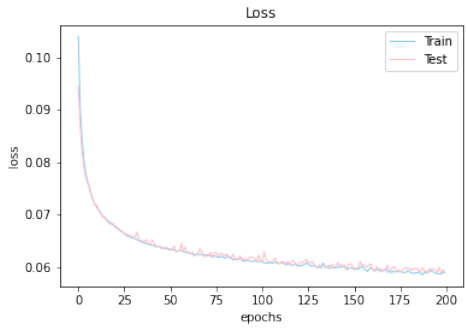
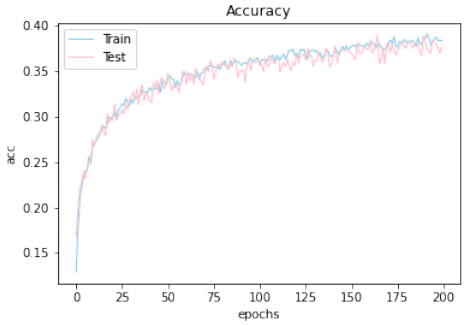
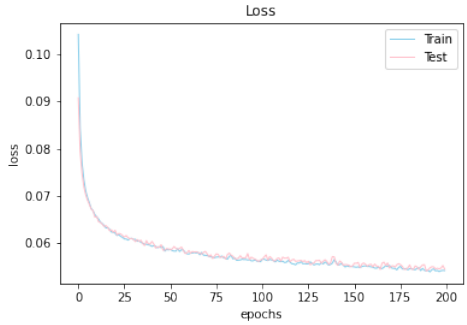
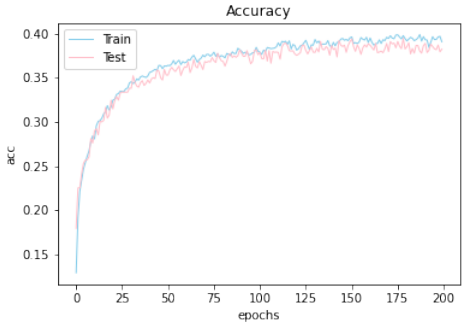
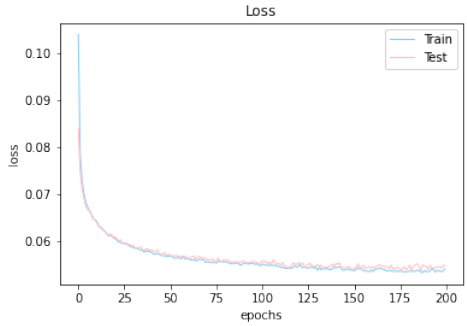
3 实验结果

3.1 MLP

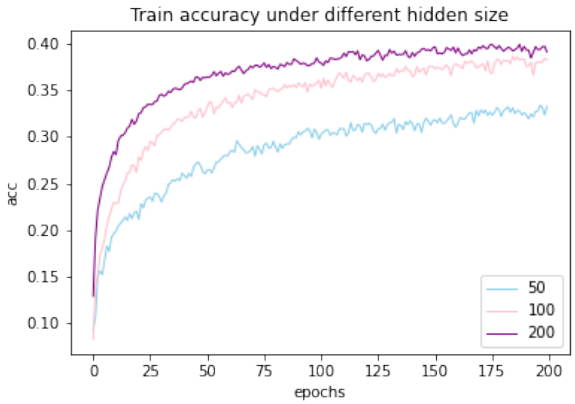
3.1.1 神经元数

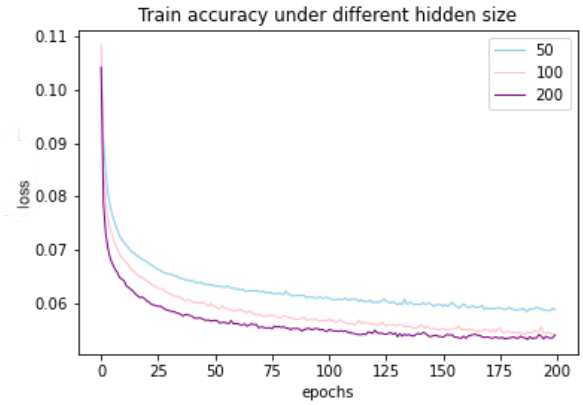
选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，隐藏层为1层，激活函数为Tanh，比较隐藏层维度分别为50、100、200时的网络性能.

- 训练结果

hidden_dim	Accuracy	Loss
50		
100		
200		

- 对比一下三者在训练集上的性能

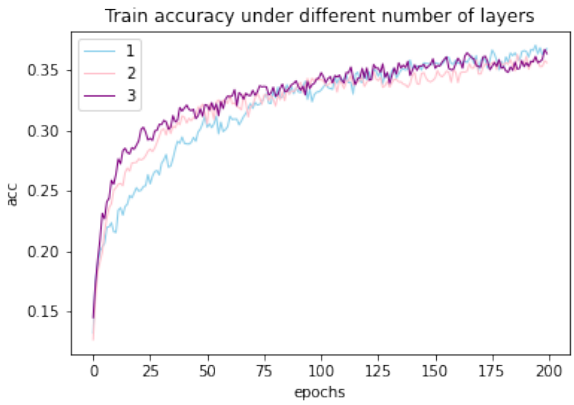


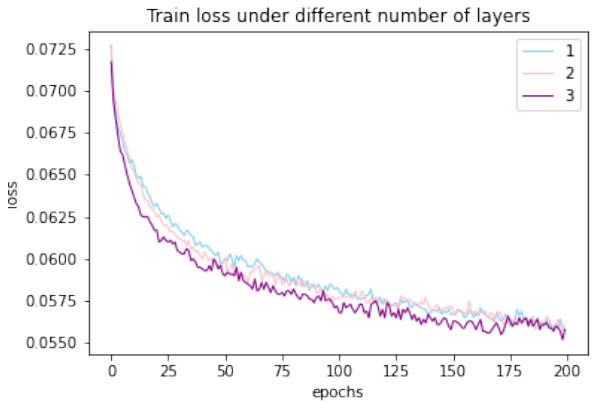


可以看到，随着隐藏层维度的增大，准确率明显上升，并且网络的收敛速度也有所加快.

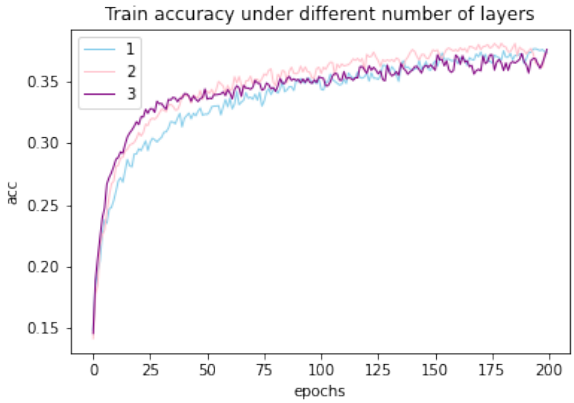
3.1.2 网络层数

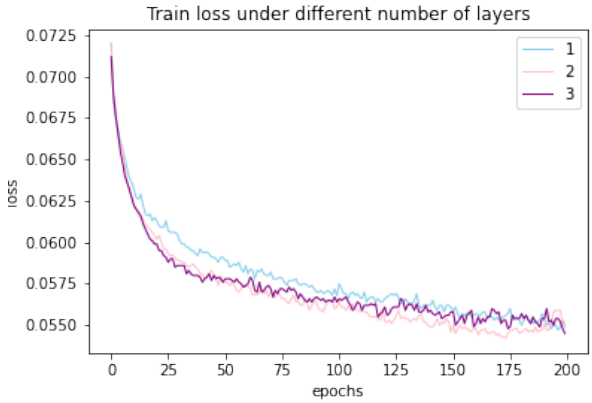
- 选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，隐藏层维度为50，激活函数为Tanh，比较隐藏层数分别为1、2、3时的网络性能.





- 更改隐藏层维度为100.

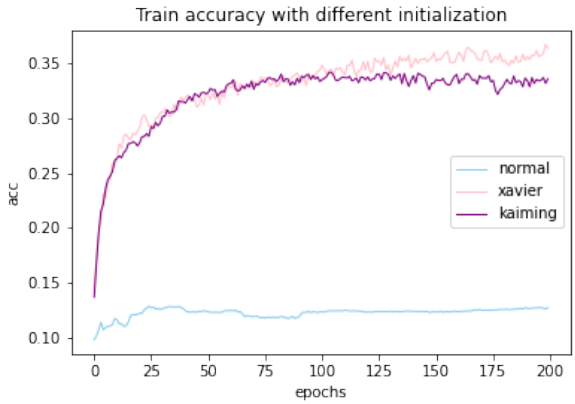


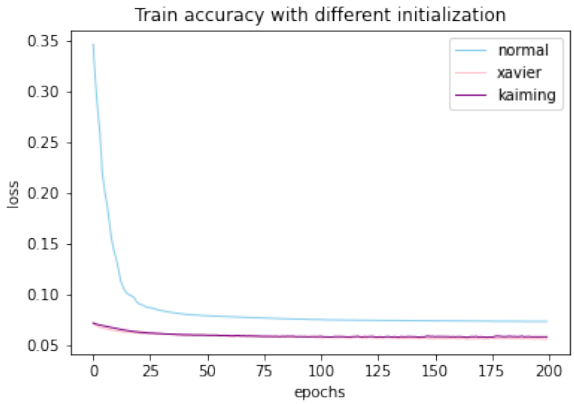


注意到在隐藏层维度相同时，随着隐藏层数的增加，准确率并没有明显提升，但是收敛速度略有加快.

3.1.3 参数初始化

选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，隐藏层维度为50，隐藏层数为3，激活函数为Tanh，比较参数初始化方法分别为Normal、Xavier、Kaiming时的网络性能.





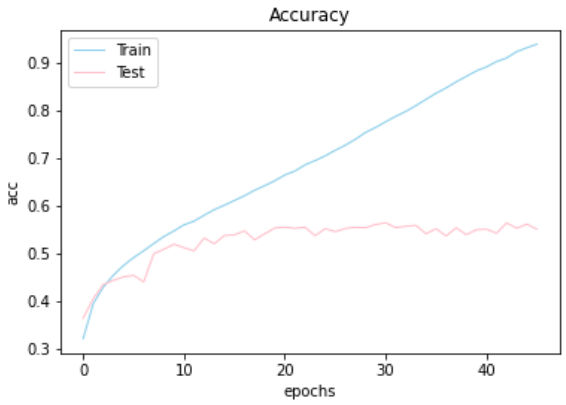
结果显示如果只用普通的均匀分布初始化，不仅收敛较慢，而且准确率也很低，Xavier初始化与PyTorch框架默认的Kaiming初始化收敛速度相似，但是Xavier初始化的正确率略高于Kaiming初始化.

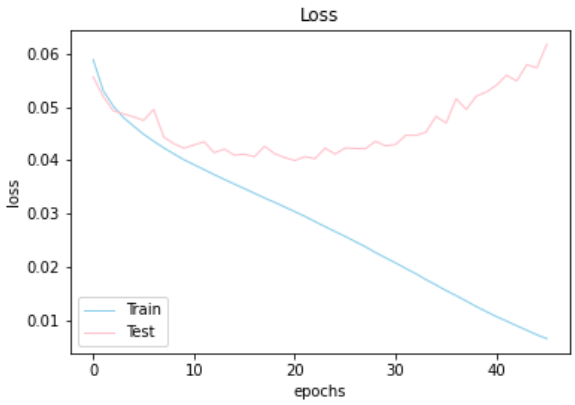
3.2 LeNet-5

3.2.1 池化层

池化层的作用通常是特征提取，以及提高计算效率，按照经验，如果不添加池化层，则很可能因为参数过多而造成过拟合. 我们删掉池化层来验证一下结果是不是这样.

选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，激活函数为ReLU，优化器为SGD.



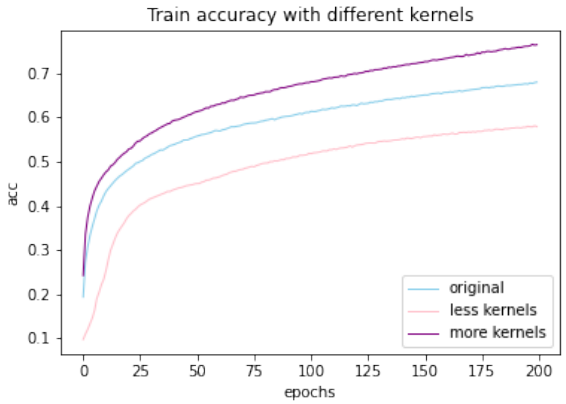


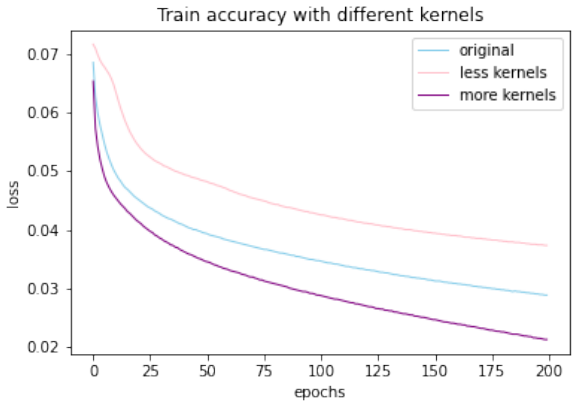
减掉pooling层，仅仅训练了40个epoch就出现了非常严重的过拟合，可见特征提取的重要性.

3.2.2 卷积核数量

将各层卷积核数量减半/加倍，与LeNet-5性能的对比.

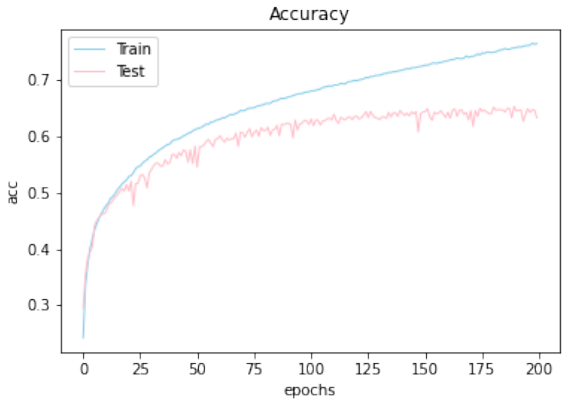
选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，激活函数为ReLU，优化器为SGD.

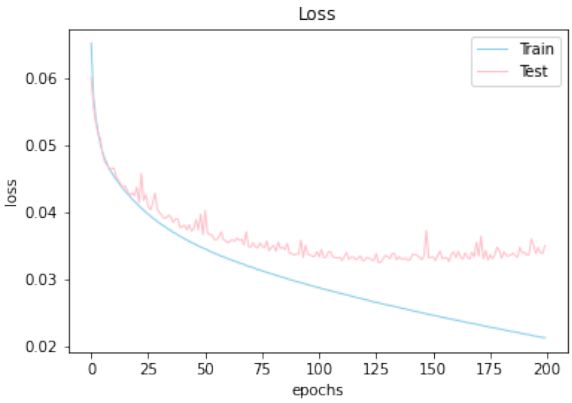




可见随着核数的增加，模型在训练集上的效果有所增加.

但是增大核数带来的问题是，降低了模型的泛化性，将各层卷积核数加倍以后，在测试集上的效果非常不稳定，而且有轻微的过拟合. 很显然，这是因为模型太复杂了.

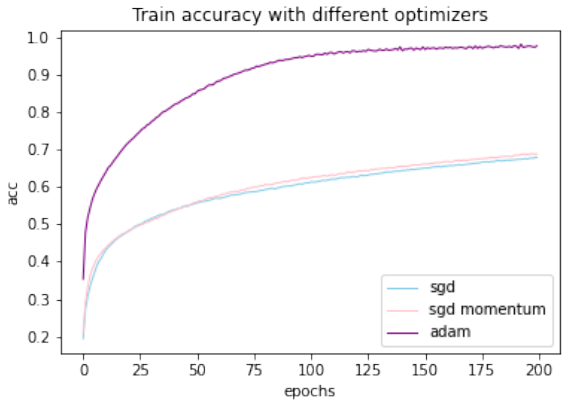


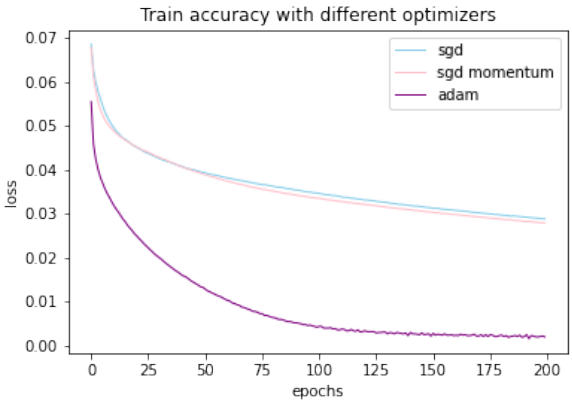


3.2.3 优化器

在LeNet-5上对比三个不同优化器的性能：SGD、SGD Momentum、Adam.

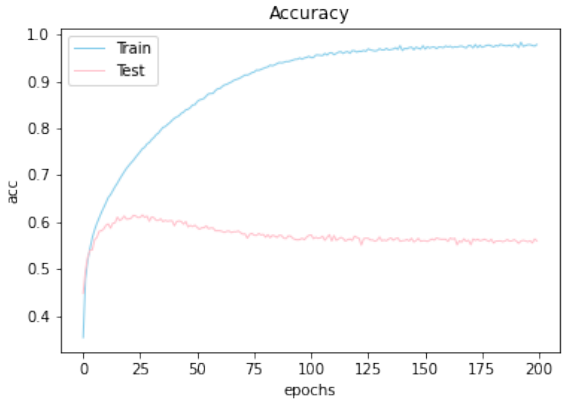
选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，激活函数为ReLU，SGD Momentum中momentum = 0.09.

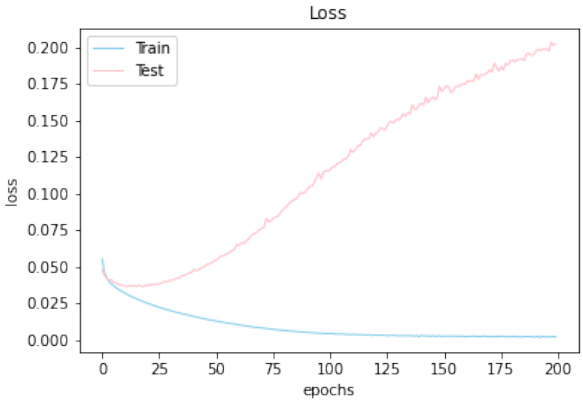




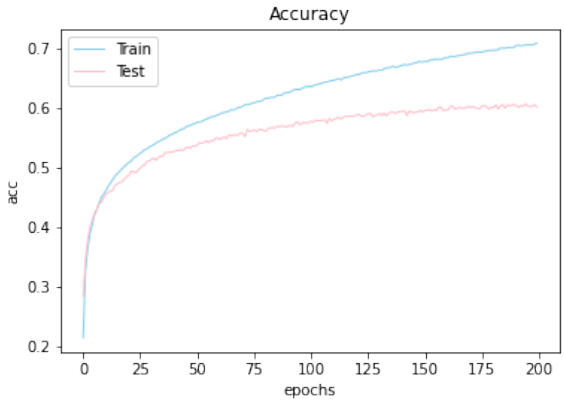
在训练集上，SGD Momentum的效果略好于SGD，而Adam优化器在收敛速度与准确率上都远远超过SGD优化器.

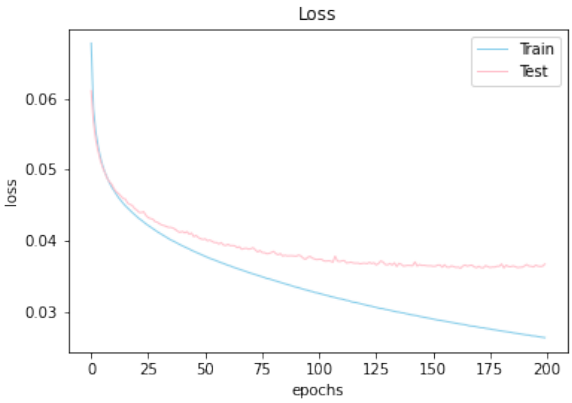
但是，在测试集上，使用与SGD同样的超参数时，Adam优化器会出现严重的过拟合.





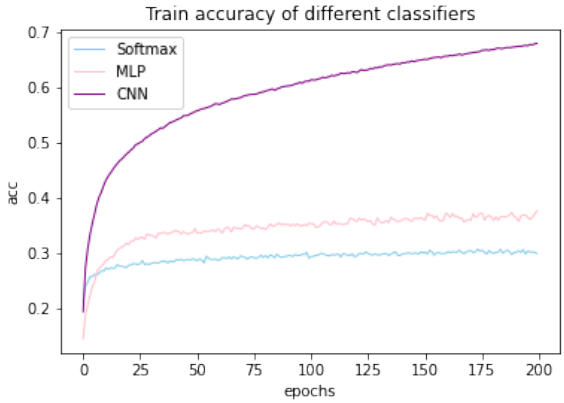
将Adam优化器的学习率调整为1e-5，没有过拟合现象轻微很多，并且准确率较高，收敛速度较快.

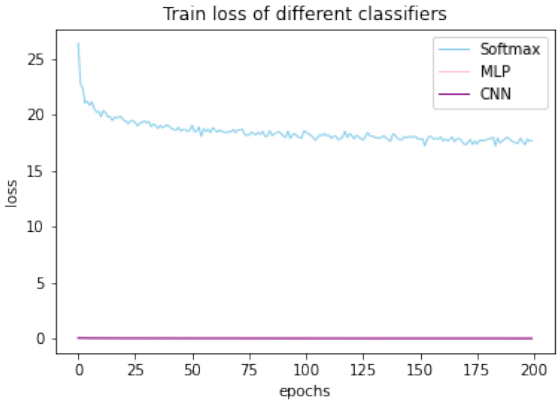


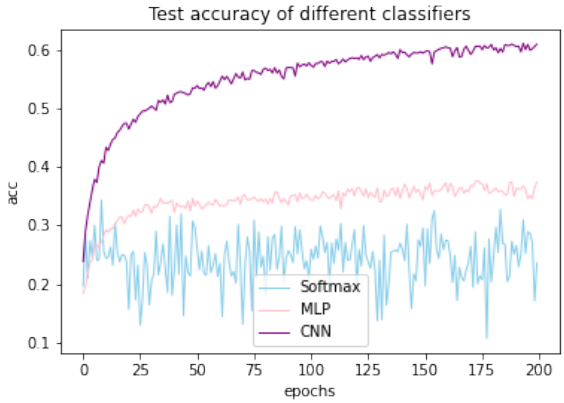


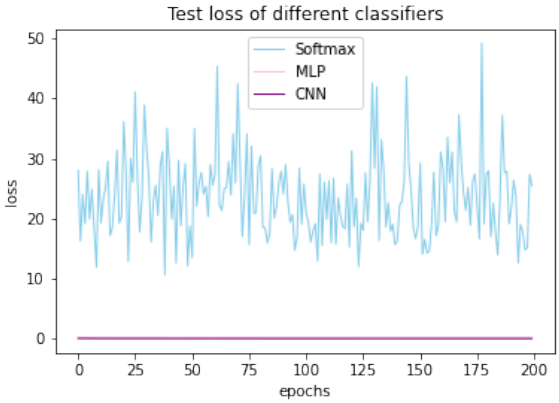
3.3 分类器性能对比

选取超参数batch_size = 32，learning_rate = 1e-4，weight_decay = 1e-4，优化器为SGD，对比三种不同分类器Softmax、MLP、CNN的性能.









Softmax分类器的性能最差，并且泛化性也很差，在测试集上的效果非常不稳定. MLP的效果略好于Softmax分类器，而CNN的效果最好，可以很好地捕捉到图像的特征并分类，并且在测试集上的效果也很优秀.