机器学习 assignment2 实验报告

21307347
陈欣宇

一、问题描述

1.1 实验内容

探索神经网络在图像分类任务上的应用

1.2 实验要求

1) 在给定的训练数据集上，分别训练一个线性分类器（Softmax 分类器），多层感知机（MLP）和卷积神经网络（CNN）
2) 在 MLP 实验中，研究使用不同网络层数和不同神经元数量对模型性能的影响
3) 在 CNN 实验中，以 LeNet 模型为基础，探索不同模型结构因素（如：卷积层数、滤波器数量、Pooling 的使用等）对模型性能的影响
4) 分别使用 SGD 算法、SGD Momentum 算法和 Adam 算法训练模型，观察并讨论他们对模型训练速度和性能的影响
5) 比较并讨论线性分类器、MLP 和 CNN 模型在 CIFAR-10 图像分类任务上的性能区别
6) 学习一种主流的深度学习框架（如：Tensorfolw，PyTorch，MindSpore），并用其中一种框架完成上述神经网络模型的实验

二、实现过程

使用 PyTorch 框架完成对应的神经网络模型

2.1. 数据预处理

关键代码：使用示例读取代码，在最后将数据转为 tensor 类型

```
def load_data(dir):
    X_train = []
    Y_train = []
    for i in range(1, 6):
        with open(dir + r'/data_batch_' + str(i), 'rb') as fo:
            dict = pickle.load(fo, encoding='bytes')
        X_train.append(dict[b'data'])
        Y_train += dict[b'labels']
    X_train = np.concatenate(X_train, axis=0)
    with open(dir + r'/test_batch', 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    X_test = dict[b'data']
    Y_test = dict[b'labels']
    X_train = torch.tensor(X_train).float()
    Y_train = torch.tensor(Y_train)
    X_test  = torch.tensor(X_test).float()
    Y_test  = torch.tensor(Y_test)
    return X_train, Y_train, X_test, Y_test
```

主函数代码：

```
if __name__=='__main__':
    X_train, Y_train, X_test, Y_test = load_data(dir='./material')
```

```
    select_net = input("select 0--Linear 1--MLP 2--CNN:")
    if select_net==0:net = LinearNet()
    elif select_net==1:net = MLPNet()
    elif select_net==2:net = CNNNet()
    select_opt = input("select 0--SGD 1--SGD momentum 2--Adam:")
    if select_opt==0:optims = optim.SGD(net.parameters(),
lr=0.01,momentum=0)
    elif select_opt==1:optims = optim.SGD(net.parameters(),
lr=0.01,momentum=0.9)
    elif select_opt==2:optims = optim.Adam(net.parameters())
    epochslist,acclist,losslist =
train(net,optims,X_train,Y_train,X_test,Y_test)
    draw(epochslist,acclist,'epochs','acc')
    draw(epochslist,losslist,'epochs','loss')
```

过程：调用 load_data 读取数据，选择要加载的模型和优化器，调用 train 函数进行训练，放回训练过程的准确度 acc 和损失 loss

训练函数：

```
def train(net,optims,X_train,Y_train,X_test,Y_test):
    num = len(Y_train)
    num_label = 10
    batchsize = 256
    iterations = int(num/batchsize)
    etimes = 150
    epochslist,acclist,losslist=[],[],[]
    startTime = time.time()
    for i in range(iterations*etimes):
        idx = torch.randint(0,num,[batchsize])
        input_b = X_train[idx]
        output_b = net(input_b)
        Y_label = F.one_hot(Y_train[idx],num_label).float()
        loss = nn.CrossEntropyLoss()(output_b,Y_label)
        optims.zero_grad()
        loss.backward()
        optims.step()
        if (i+1) % iterations==0:
            acc = test(X_train,Y_train,net)
            print('epochs: %d, loss: %.2f, accuracy: %.2f' %
((i+1)/iterations, loss.item(), acc))
            epochslist.append((i+1)/iterations)
            acclist.append(acc)
            losslist.append(loss.item())
    endTime = time.time()
    # print('Total time:{:.4f}'.format(endTime-startTime))
    return epochslist,acclist,losslist
```

循环每次从训练集中随机抽取 batchsize 个训练数据，训练总数据累加为整个训练集数据量时算作一个 epochs，每个 epochs 对当前训练模型用测试集进行测试，记录此时的损失 loss 和准确度 acc。这里计算损失函数前先把标签 Y_train 转为 one hot 编码，之后计算 CrossEntropyLoss 损失，CrossEntropyLoss 相当于对输出进行 log_softmax 和 nlloss，故之后在 net 模型输出时不必再调用归一化函数。

Test 函数：

```
def test(X, Y, net):
    num_data = X.shape[0]
    num_correct = 0
    for i in range(num_data):
        input = X[i]
        output = net(input)
        label = output.max(1, keepdim=True)[1].item()
        true_label = Y[i].item()
        if label == true_label:
            num_correct += 1
    return num_correct / num_data
```

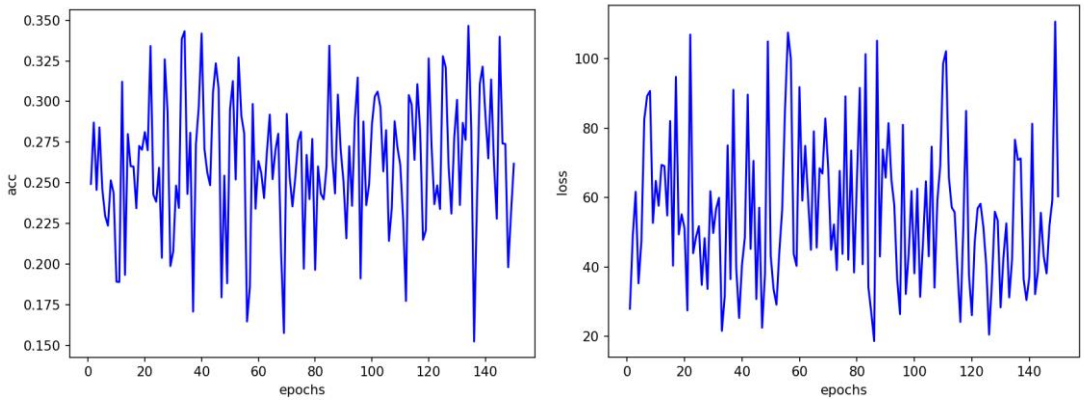
比对每个测试数据的预测值与正确值，统计该模型对于测试集的准确率。

2.2 线性分类器

网络模型如下，即只包含一个线性层，将输入转为[X,3*32*32]格式即可输入线性层直接输出

```
class LinearNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(3*32*32, 10)
    def forward(self, x):
        x = x.reshape(-1,3072)
        x = self.l1(x)
        return x
```

为方便对照，以下实验默认：设置 150 个 epochs 测试，batchsize 为 512，使用 Adam 模型在训练过程周期性用测试集进行测试准确度 acc，以及同时记录 loss 变化，结果如下：



总的看使用线性分类器训练该图像识别模型效果不是很好，准确度保持在较低位置起伏不定，在训练集的准确度为 29%，处于欠拟合状态。

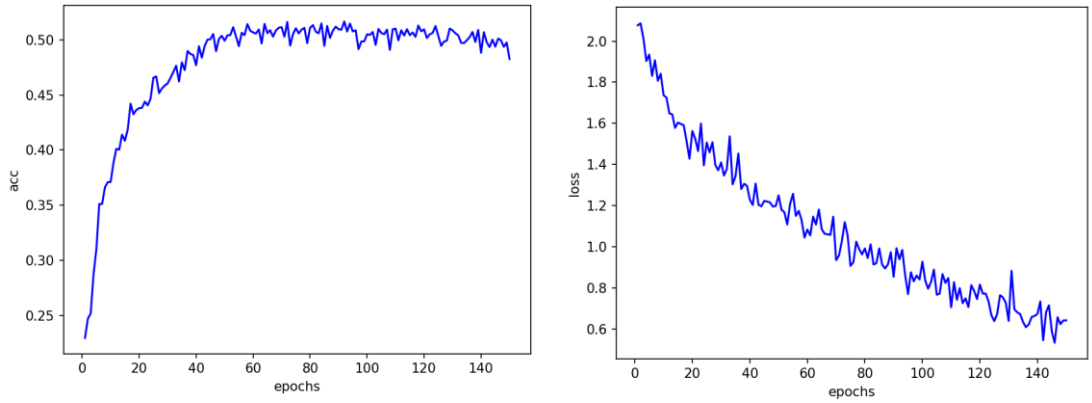
2.3 多层感知器(MLP)

2.3.1 模型示例

网络模型示例如下：这里使用了 4 个线性层，使用金字塔结构，每层神经元数逐渐递减

```
class MLPNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(3*32*32, 1024)
        self.l2 = nn.Linear(1024, 512)
        self.l3 = nn.Linear(512, 256)
        self.l4 = nn.Linear(256, 10)
    def forward(self, x):
        x = x.reshape(-1,3072)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = self.l4(x)
        return x
```

使用同样 batchsize=512，Adam 优化器，对以上 MLP 模型训练得到：

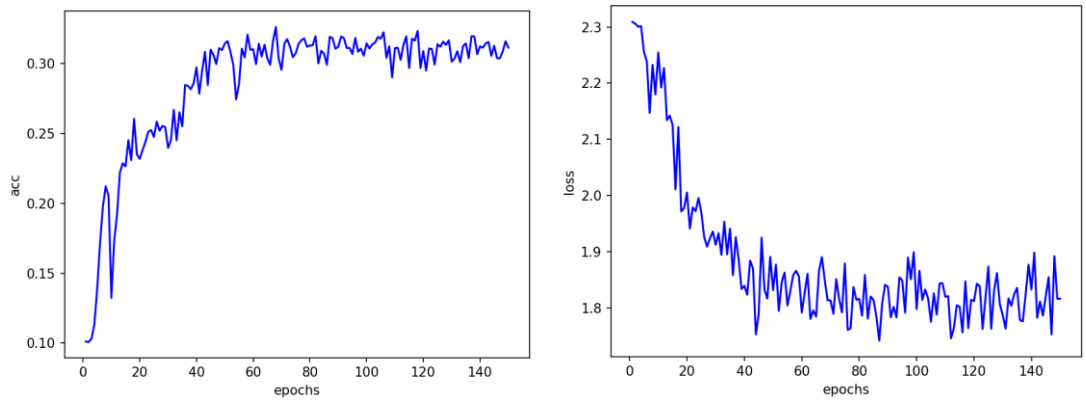


对于测试集数据，acc 在前 60epochs 有明显上升过程，在之后趋于稳定在 50%左右，而训练集的 loss 一直保持下降趋势，未收敛，但随着 loss 的下降测试集 acc 一直保持在 0.50 稳定波动，可能已经出现过拟合的情况。

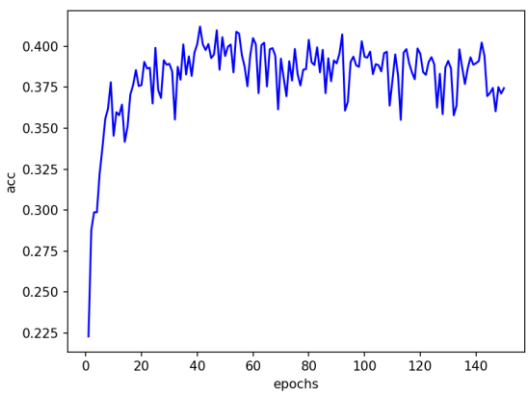
2.3.2 研究不同层数和不同数量的神经元对模型性能的影响

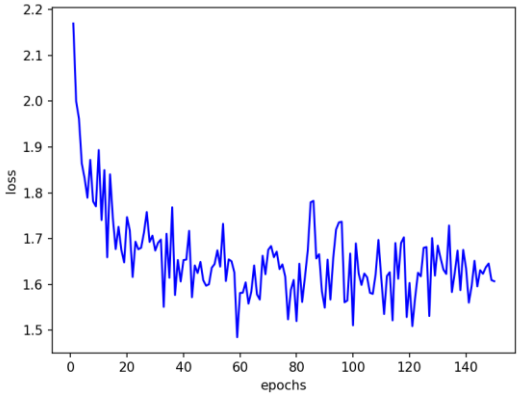
比较方式：训练不同层数的模型，每个模型中调整神经元数量，比较对模型性能的影响

(1) **1 个隐藏层**：调整为 512 个神经元进行训练，收敛速度慢，最终趋于 30%的准确度，在训练集中的准确率也仅有 33%。512 个神经元训练情况：



调整到 2048 个神经元训练情况：

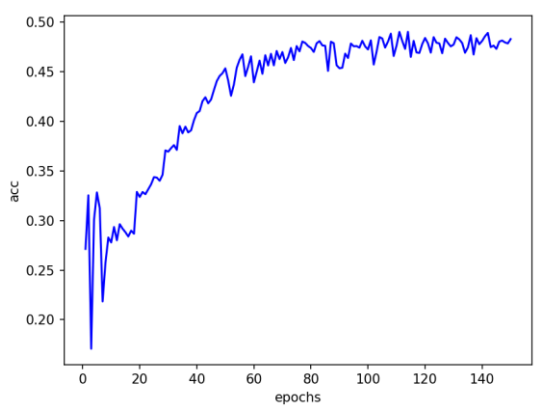


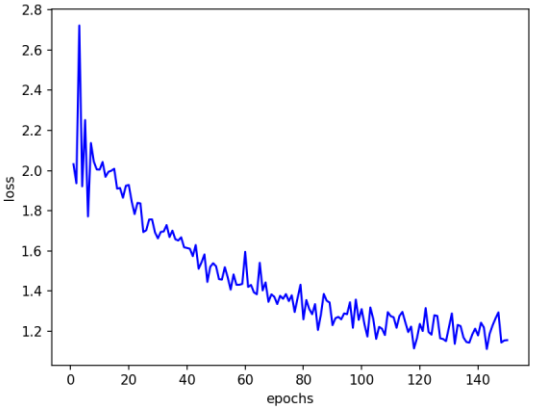


可见神经元的数量提升在一定程度上加快了单位 epoch 内的训练效率和准确度，但也使得训练时间大幅增加，在 2048 神经元的训练耗费了远超 512 神经元的大量时间，实际训练效益不高。

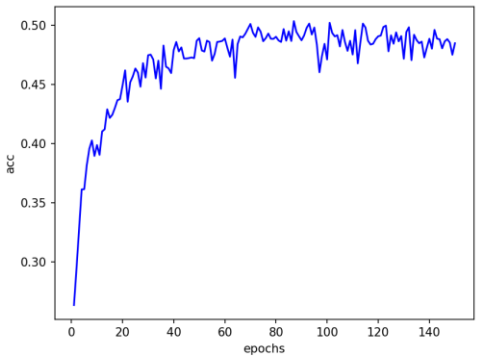
(2) **2 个隐藏层：**拟合速度总体也偏慢，但也最后趋于 50%的准确度

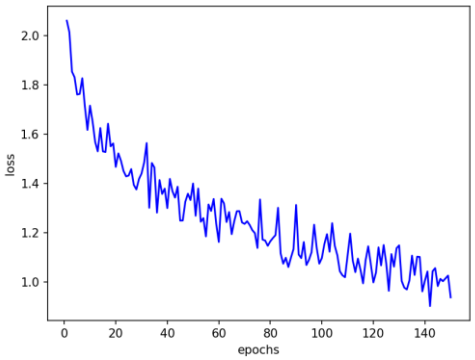
隐藏层分别使用 512、128 个神经元，训练结果如下：acc 收敛至 0.46 左右，loss 也已趋于收敛，训练结果在训练集中的准确度也只有 0.58，效果平平无奇。



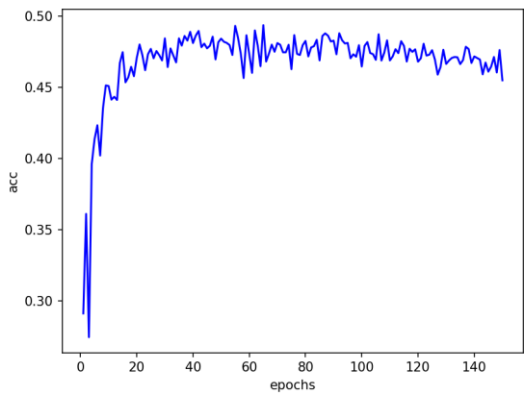


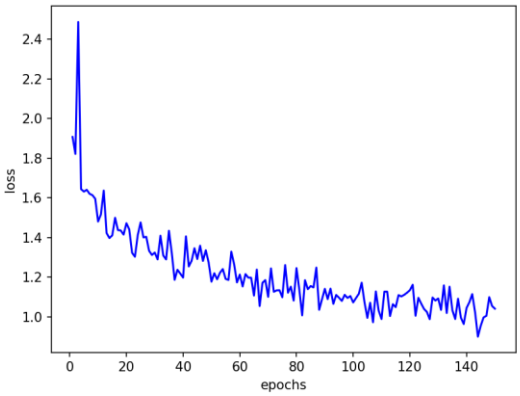
隐藏层使用 1024、256 个神经元，训练情况：最终在训练集准确率为 0.63





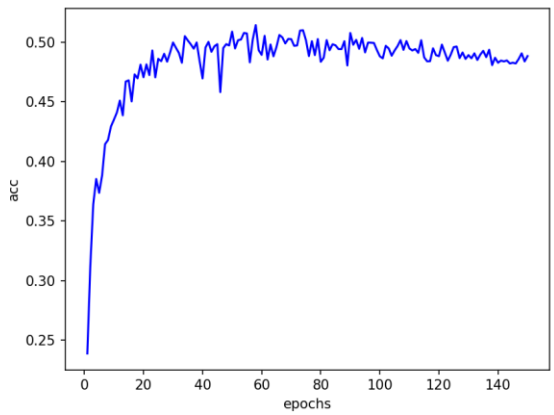
隐藏层使用 1024、512 个神经元，训练情况：很快到达 0.40 的准确度，收敛速度变快，在训练集也达到 0.6 左右的准确度，神经元数量的提升与前面的效果差不多，在数量达到一定程度后无法带来有效的增益。增加神经元数加快了收敛速度，但准确度无法得到有效提升，在 50%以下波动，没有有效提升模型的性能。

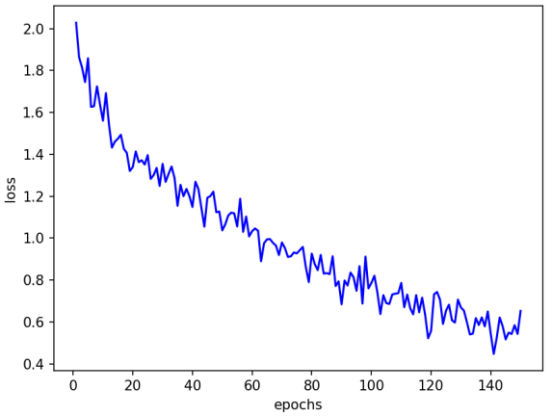




(3) **3 个隐藏层：**

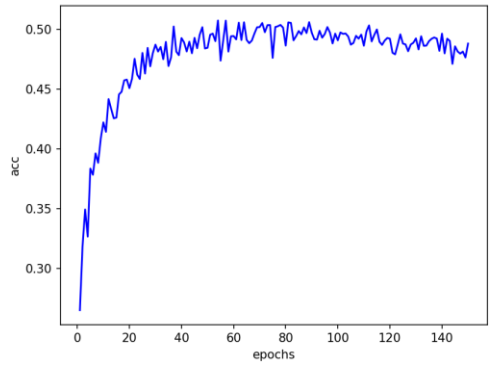
隐藏层使用 1024、512、128 个神经元，训练情况：训练集自身准确度 0.81，相较两层隐藏层有了提升，但测试集的准确度仍在 0.5 左右波动，仍是过拟合的问题，隐藏层数的增加没有缓解这个问题。

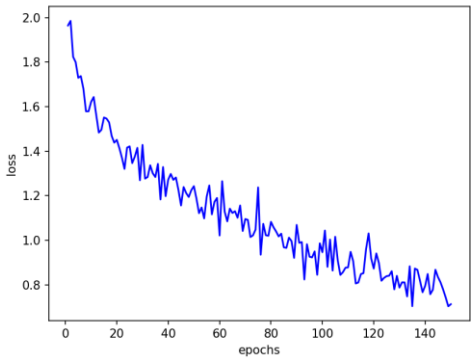




继续调大神经元数依旧会导致过拟合，尝试调小神经元数量

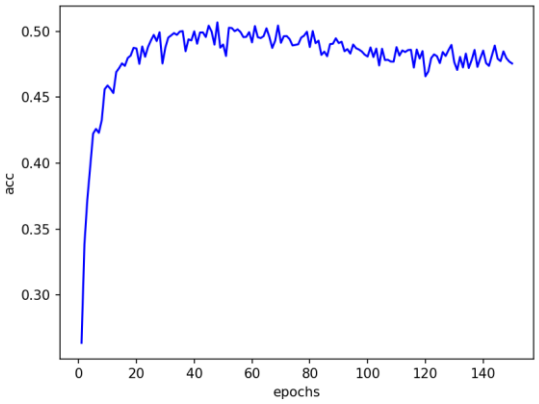
隐藏层使用 512、256、128 个神经元，神经元数量下降导致了最终训练集的准确度也下降为 0.74，但测试集的准确度仍然在 0.50 左右波动，且拟合速度与更多神经元数的模型相差不大。

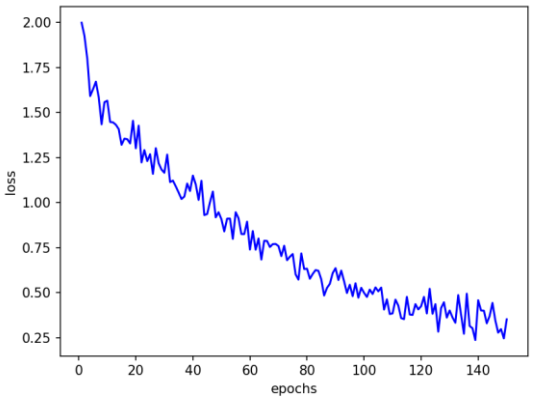




(4) 4 个隐藏层：

训练情况：训练集自身准确度 0.86，比 3 层隐藏层有一定提升，但由下图可见，测试集准确度最高值依旧没有很好的提升，MLP 模型的训练结果达到了一定瓶颈。



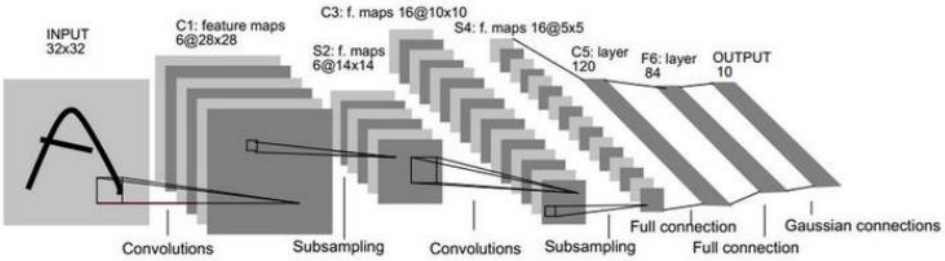


总的来看，增加隐藏层数量可以提高训练模型的准确度，但无法解决原本就存在的过拟合现象，神经元数量也需要合理设置，设置过多的神经元会导致训练时间大幅增加，且加剧过拟合现象。

2.4 卷积神经网络(CNN)

2.4.1 模型示例

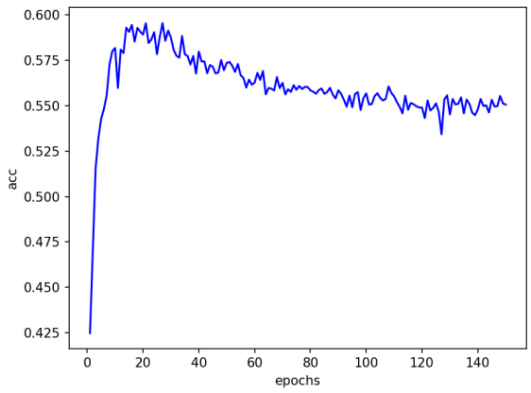
参考 LeNet 模型

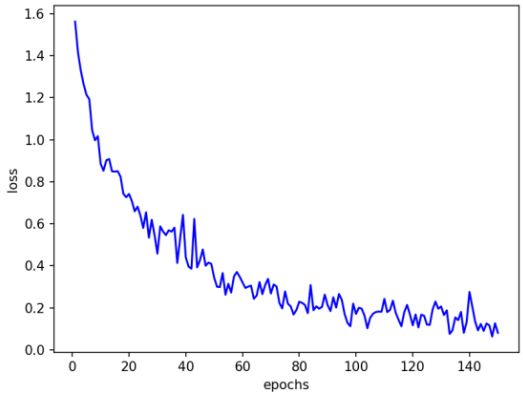


对应模型代码如下：

```
class CNNNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.c1 = nn.Conv2d(3, 6, 5)
        self.c2 = nn.Conv2d(6, 16, 5)
        self.l1 = nn.Linear(400, 120)
        self.l2 = nn.Linear(120, 84)
        self.l3 = nn.Linear(84, 10)
    def forward(self, x):
        x = x.reshape(-1,3,32,32)
        x = F.relu(self.c1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.c2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.reshape(-1,400)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = self.l3(x)
        return x
```

依旧使用 batchsize=512，Adam 优化器，对该 CNN 模型训练得到

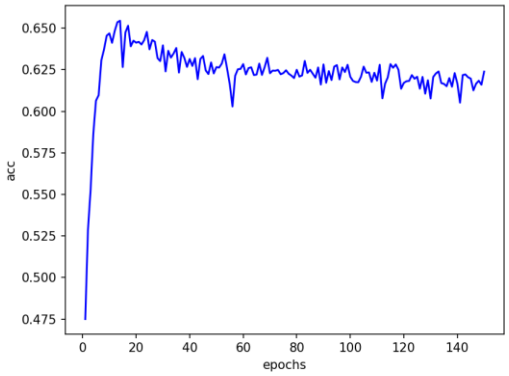


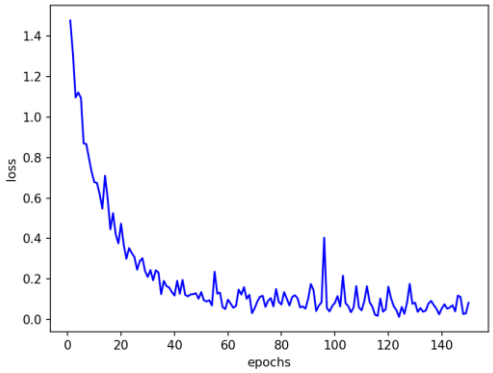


在该模型中，测试集准确度在 epochs=20 左右达到最高值，之后随着训练集 loss 的下降出现准确度降低的情况，说明该训练模型已经出现过拟合现象，最后 loss 也逐渐收敛，训练集准确度也达到了 0.97。

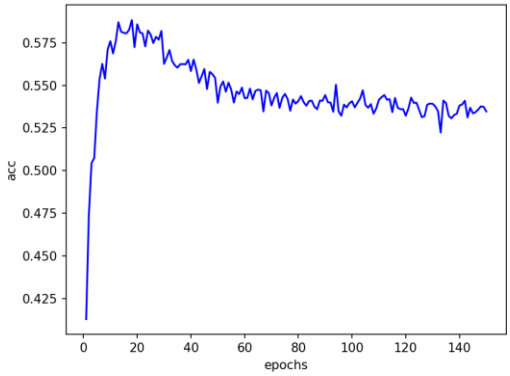
2.4.2 探索不同结构因素对模型性能的影响

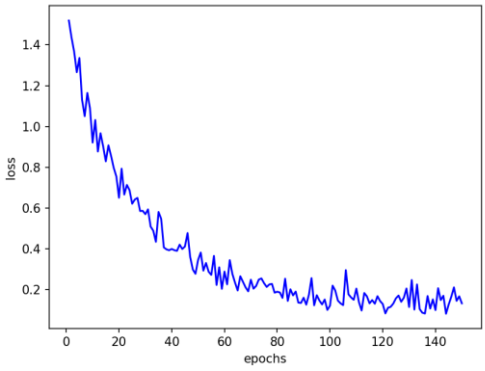
(1) 增加两个卷积层的滤波器数量，分别加至 16 和 64，其他结构与前文保持一致，运行：准确率提升到 0.64 左右，训练集准确度达到 0.99，准确度最大值变大，训练结果得到优化





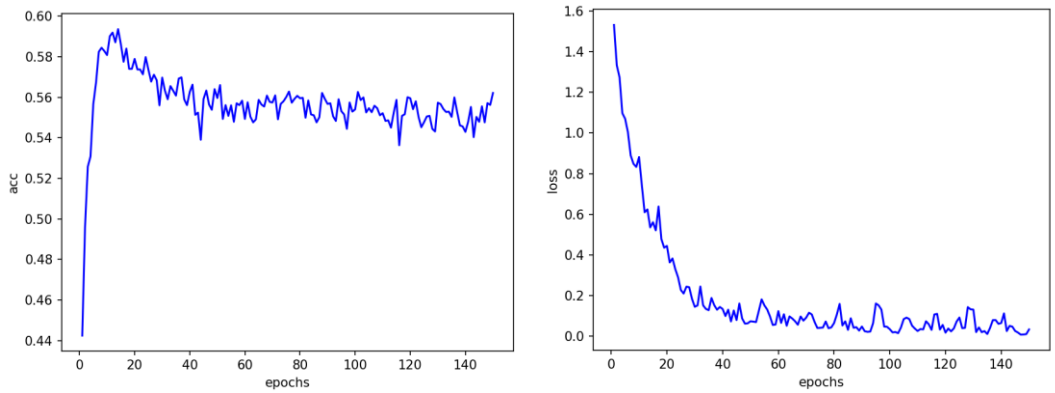
(2) 在原始 cnn 模型上，增加一个卷积层，训练集准确度达到 0.95，相较 2 个卷积层的性能没有提升，反而过拟合的情况变强，





(3) pooling 的使用

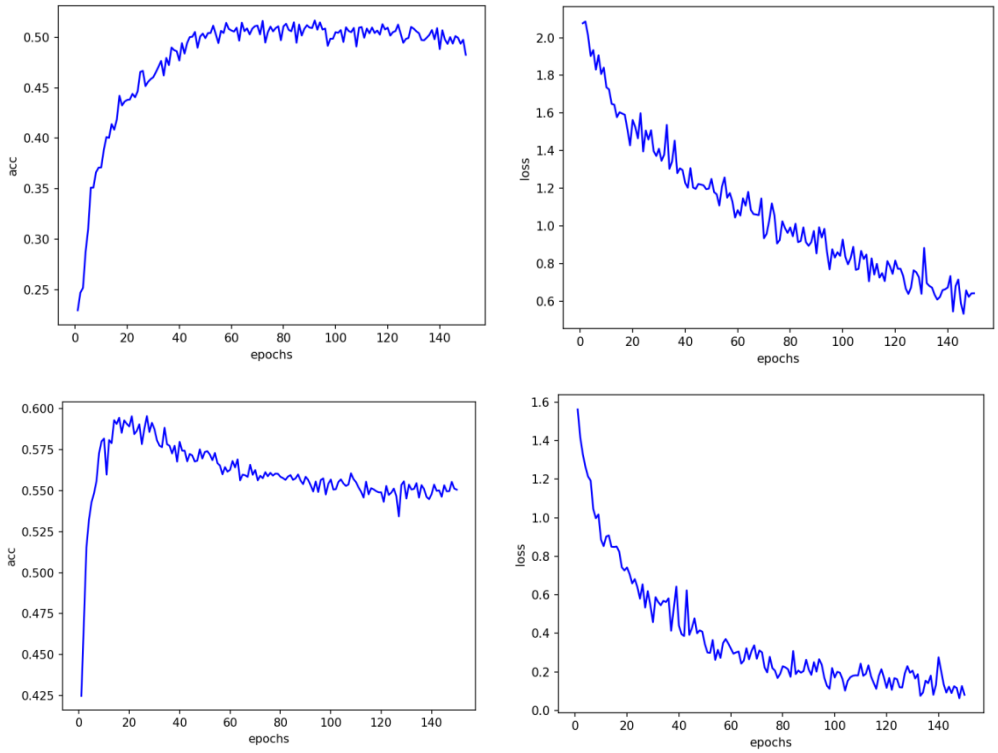
前面的模型都包括了 pool 层的使用，下面去掉第一个 pool 层，观察对训练结果有无影响训练情况：最终训练集准确度为，在去掉一个 pool 的情况下，模型参数增多，训练时间也明显变长，训练结果与初始模型并无明显区别，训练集准确度达到 0.99 过拟合现象也有所加重。



2.5 神经网络模型性能比较

比较以上线性分类器、MLP、CNN 三类神经网络模型，对比几个模型中初始示范的例子，均采用 Adam 算法，可以看出线性分类器在该图像处理问题中的准确度和训练集 loss 都远差于另外两个模型。主要对比 MLP 和 CNN 的性能，相较于 MLP，CNN 的准确度收敛迅速，但很快出现过拟合现象，准确率反而出现下降趋势，在训练集 loss 上，CNN 的 loss 下降更快且在 150epochs 内趋于收敛，最终训练集的准确度分别达到 0.81 和 0.96，出现不同程度的过拟合现象。

如下 4 个图展示 MLP 和 CNN 的 acc 和 loss 曲线:



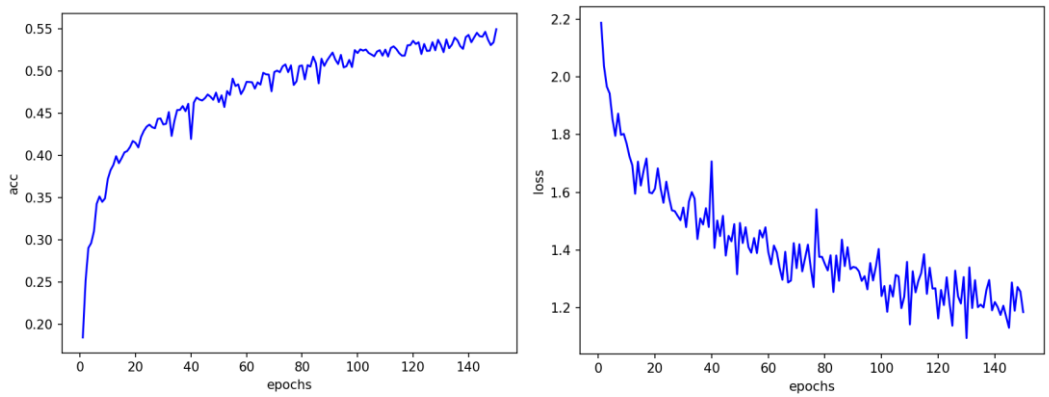
2.6 不同优化算法性能比较

选用神经网络模型为默认的 LeNet 模型，训练参数保持与前面实验一致，便于比较，分别使用不同的优化器，在 batchsize=150 范围内进行比较。

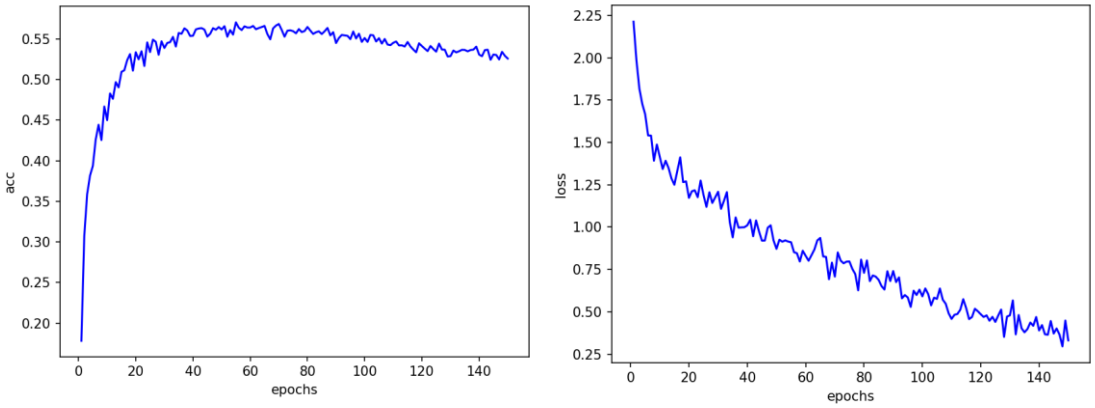
```
optim.SGD(net.parameters(), lr=0.001,momentum=0)
optim.SGD(net.parameters(), lr=0.001,momentum=0.9)
optim.Adam(net.parameters())
```

(1) SGD 算法: 最终训练集的准确度为 0.59，主要是因 SGD 训练的收敛速度慢，在 150epochs

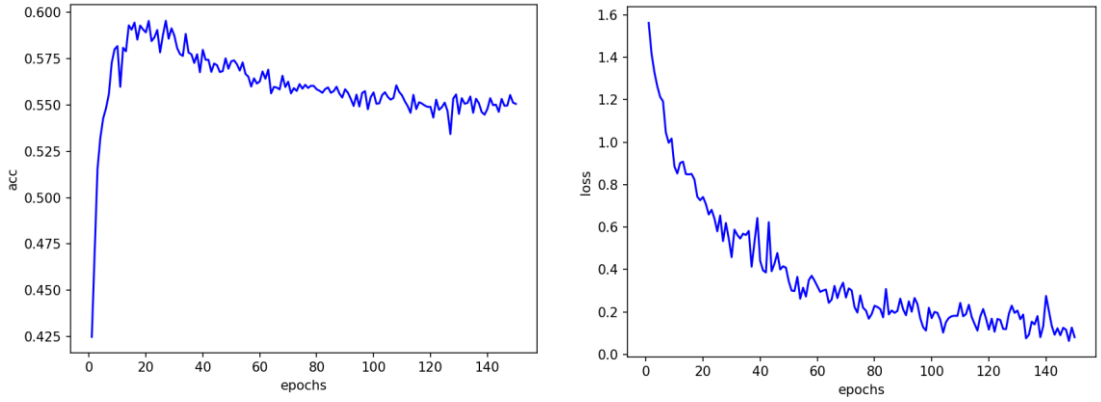
时仍未完全收敛。



(2) SGD Momentum 算法：最终训练集的准确度为 0.86，在测试集的准确度以达到收敛，拟合速度相比 SGD 算法变快，且 loss 曲线的抖动剧烈程度变小，但在最后测试集出现了微小的下降趋势。



(3) Adam 算法：最终训练集的准确度为 0.97，训练集基本训练完成，训练效率快，loss 的下降速度比 SGD Momentum 快许多，但带来结果的过拟合现象也十分明显。



总的来看，Adam 算法表现的最佳准确度以及训练速度都优于其他两类算法，最终训练集的测试结果已经十分准确，出于暂时无法解决的过拟合现象，Adam 算法也由于训练速度快表现得更为严重，但另外 SGD 和 SGD Momentum 也会在训练到一定程度上表现出这种现象，因此这里并不作为 Adam 劣于其他两类算法的依据。

三、实验总结

本次实验从实现代码到后面漫长的训练比对模型算法，都花了较长的时间，主要探究了不同的神经网络模型，不同的优化算法下训练性能的差异，以及在调参中比对不同网络层数和神经元数量带来的影响，不同的模型基本上都需要在一个适中的量上才能达到最佳的训练效

果。对于最佳的训练结果，在测试集上最好的训练结果准确率也才 60%左右，过拟合的问题还有待解决，这还是很需要自己探索和改进的地方。