[redacted]

实验课程：人工智能导论	年级：大三	实验成绩：
[redacted]		
		实验日期：2022/12/2

一、构建的模型

（一）代码：

```
#new_method.py
import torch
import torchvision
import torch.nn.functional as F
from torch import nn
from torch.utils.data import DataLoader
from torch.utils.tensorboard import SummaryWriter
#定义训练的设备：用GPU来运行
device = torch.device("cuda")
#准备数据集
train_data=torchvision.datasets.MNIST("./data",train=True,transform=torchvision.
transforms.ToTensor(),download=True)
test_data=torchvision.datasets.MNIST("./data",train=False,transform=torchvision.
transforms.ToTensor(),download=True)
# print("size_of_train_data:{}".format(len(train_data)))    60000张图片用来训练
# print("size_of_test_data:{}".format(len(test_data)))      10000张图片用来测试

#加载数据集
train_dataloader = DataLoader(train_data,batch_size=64)
test_dataloeader = DataLoader(test_data,batch_size=64)
#构建神经网络
class Mynn(nn.Module):
    def __init__(self):
        super(Mynn, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)
    def forward(self,x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return x
#测试所创建的神经网络
# if __name__ == '__main__':
#     mynn = Mynn()
```

```
#     input=torch.ones(64,1,28,28)
#     output = mynn(input)
#     print(output.shape)

#创建网络模型
mynn = Mynn()
mynn = mynn.to(device)
#损失函数：交叉熵损失
loss_fn = nn.CrossEntropyLoss()
loss_fn = loss_fn.to(device)
#优化器：随机梯度下降
learning_rate = 0.01 #学习速率
optimizer = torch.optim.SGD(mynn.parameters(),lr=learning_rate)

#设置训练网络的一些参数
total_train_num =0
total_test_num=0
epoch =10
writer = SummaryWriter("my_logs")#tensorboard用来展示结果

for i in range(epoch):
    print("------第{}轮训练开始-----".format(i+1))

    #开始训练
    mynn.train()
    for data in train_dataloader:
        imgs,targets = data
        imgs = imgs.to(device)
        targets = targets.to(device)
        outputs = mynn(imgs)
        loss = loss_fn(outputs,targets)#计算loss
        optimizer.zero_grad() #优化器优化模型
        loss.backward() #根据loss改变参数
        optimizer.step()
        total_train_num+=1#记录训练次数
        if total_train_num %500==0:#每500次训练后输出LOSS值
            print("训练次数：{}，LOSS：{}".format(total_train_num,loss.item()))

    #测试步骤开始：
    mynn.eval()
    total_test_loss=0
    total_accuracy=0
    with torch.no_grad():
        for data in test_dataloeader:
            imgs,targets = data
            imgs = imgs.to(device)
            targets = targets.to(device)
            outputs =mynn(imgs)
            loss= loss_fn(outputs,targets)#计算loss
            total_test_loss=total_test_loss+loss.item()
            accuracy = (outputs.argmax(1)==targets).sum()#计算预测准确的数据的个数
            total_accuracy = total_accuracy +accuracy#统计总共的预测准确的数据的个数
    #打印本次训练的结果：
    print("测试集上的LOSS：{}".format(total_test_loss))
    print("测试集上的准确率：
{}".format(total_accuracy/len(test_data)))#total_accuracy/len(test_data)是准确率
    writer.add_scalar("test_loss",total_test_loss,total_test_num)
```

```
    writer.add_scalar("test_accurary", total_accuracy/len(test_data),
total_test_num)
    total_test_num = total_test_num +1
#保存模型
torch.save(mynn,"mynn_pth")
print("模型已保存")
writer.close()
```

- 注：构建的网络参考了一篇CSDN【2】上的文章。
- 参考资料：

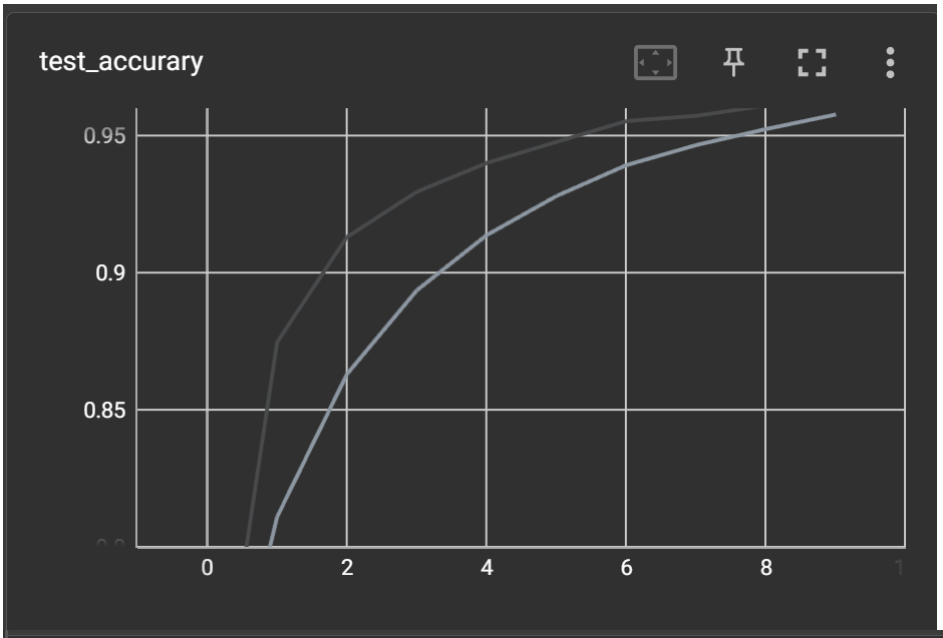
【1】PyTorch深度学习快速入门教程（绝对通俗易懂！）【小土堆】哔哩哔哩bilibili

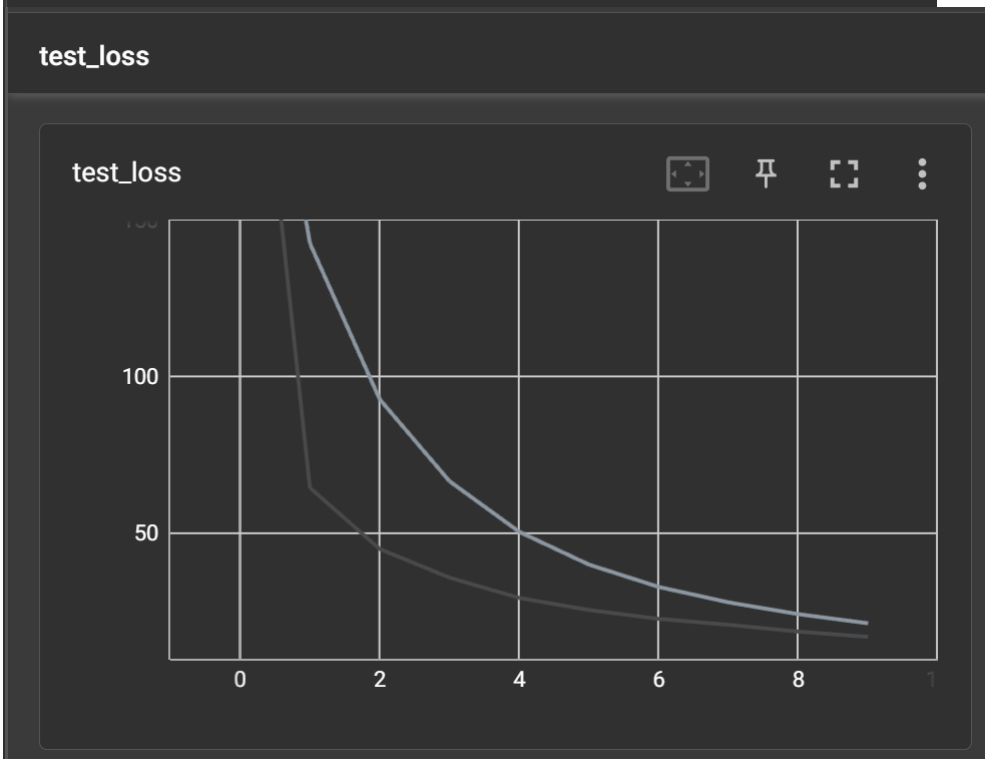
【2】(29条消息) 用PyTorch实现MNIST手写数字识别(非常详细)小锋学长生活大爆炸的博客-CSDN博客 pytorch mnist

（二）实验过程

- 使用python语言，pytorch框架。60000张图片用来训练，10000张图片用来测试。

（三）结果分析





- 经过10轮训练后，得到最高准确率约为96.72%

```
————第10轮训练开始————
训练次数：8500，LOSS：0.4438662528991699
训练次数：9000，LOSS：0.427619069814682
测试集上的总共的LOSS：16.482970500946976
测试集上的准确率：0.967199981212616
模型已保存
```

二、vgg16网络模型：

（一）代码：

- 我先尝试直接用pytorch中自带的vgg16的网络模型进行训练（下图只展示部分修改的代码，全部的代码可以在vgg16.py中查看）

```
#vgg16.py
#定义transform
resize = transforms.Resize((64,64)) #将图片放大
tensor = transforms.ToTensor()
trans_compose = transforms.Compose([
    tensor,
    resize
])

#构建神经网络
vgg16 = torchvision.models.vgg16(weights = None)
vgg16.features[0] = nn.Conv2d(1,64,kernel_size=3)#修改输入，以适应mnist数据集1通道的输入
vgg16.classifier[6]=nn.Linear(4096,10)#修改输出，最终分为10类
#测试所创建的神经网络,主要看输入和输出的格式是否满足要求
# if __name__ == '__main__':
#     img = torch.ones(64,1,28,28)
#     resize = transforms.Resize((64,64))
#     new_img = resize(img)
#     output = vgg16(new_img)
#     print("vgg16:",output.shape)

#创建网络模型
vgg16 =vgg16.to(device)
#损失函数：交叉熵损失
loss_fn = nn.NLLLoss()
loss_fn = loss_fn.to(device)
#优化器：随机梯度下降
learning_rate = 0.000001 #学习速率
optimizer = torch.optim.SGD(vgg16.parameters(),lr=learning_rate)
#设置训练网络的一些参数
total_train_num =0
total_test_num=0
epoch =10
writer = SummaryWriter("vgg_logs")
```

- 这个代码用的vgg16的网络是pytorch中自带的模型，但是由于这个神经网络原本是用于一个大数据集，所以对mnist数据集来说太过复杂，分类效果并不好。
    - 由于vgg16中有多个pooling，所以我先将mnist数据集中图形的大小从28×28拉伸到64×64
    - 运行过程中由于网络太过复杂，所以出现了梯度爆炸的情况，得到的LOSS值变为NAN。我尝试用减小学习率的方法来防止，最终将学习率控制在0.000001时，可以运行，但是学习速率非常慢（如图所示），经过10轮的学习后，准确率只有：7.43%

```
测试集上的总共的LOSS：-6.611109267920256
测试集上的准确率：0.07429999858140945
100%|██████████| 10/10 [28:27<00:00, 170.77s/it]
模型已保存
```

    - 因为学习率非常小，所以要得到高准确性就要多轮的学习，又因为图片被放大，所以训练的很慢。
- 因此，我参考了(29条消息) VGG网络MNIST分类任务Pytorch实现_K5niper的博客-CSDN博客这篇文章的内容，修改了网络的结构。

```
#vgg_2.py
#对数据集进行变换
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=(0.5,), std=(0.5,))
])
#构建神经网络
class Vgg_2(nn.Module):
    def __init__(self):
        super(Vgg_2, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 64, kernel_size=3, stride=1, padding=1),#将输入的channel
数变为1
            nn.BatchNorm2d(64),#增加归一化处理
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(64),#增加归一化处理
            nn.ReLU(),
            nn.MaxPool2d(stride=2, kernel_size=2),
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),#增加归一化处理
            nn.ReLU(),
            nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(128),#增加归一化处理
            nn.ReLU(),
            nn.MaxPool2d(stride=2, kernel_size=2)
        )
        self.classifier = nn.Sequential(
            nn.Linear(7 * 7 * 128, 256),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(p=0.5),
            nn.Linear(128, 10)
        )
        self._initialize_weights()

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, start_dim=1)
        x = self.classifier(x)
        return x

    def _initialize_weights(self):#学习参数的初始化
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(tensor=m.weight, mode="fan_out",
                                        nonlinearity="relu")
                if m.bias is not None:
                    nn.init.constant_(tensor=m.bias, val=0)
```

```
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(tensor=m.weight, val=1)
                nn.init.constant_(tensor=m.bias, val=0)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(tensor=m.weight, mean=0, std=0.01)
                nn.init.constant_(tensor=m.bias, val=0)
```

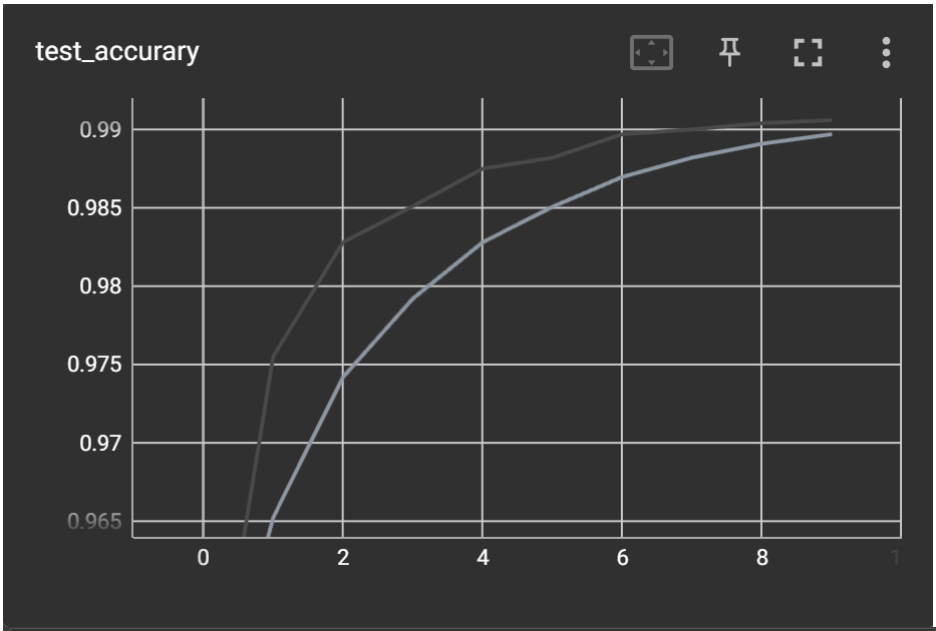
- 网络模型的改变主要体现在：
    - 减少了卷积层和pooling的数量，并增加了数据的归一化处理。
    - 增加了参数的初始化这一步骤

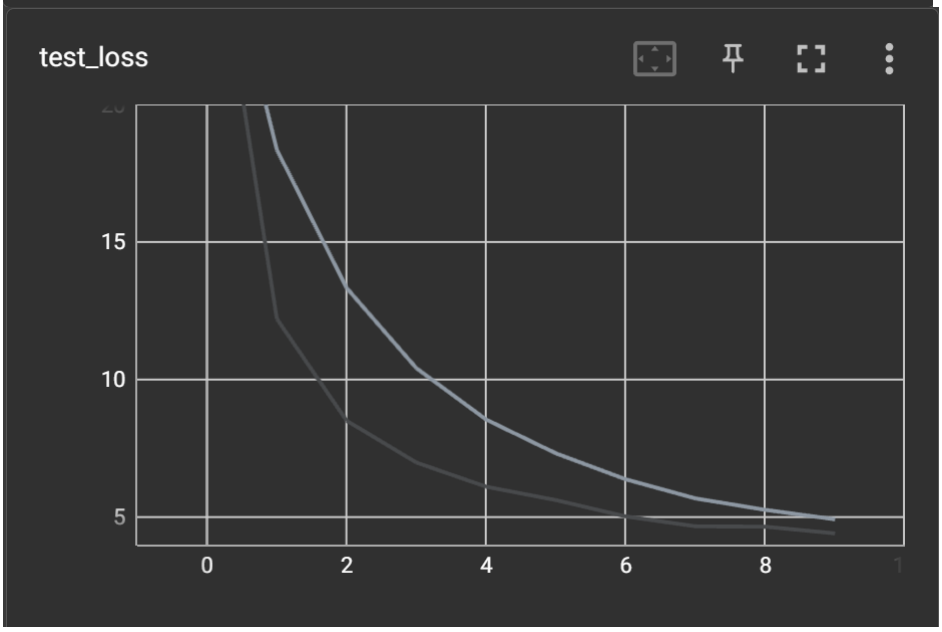
（二）实验过程：

- 使用了python语言，pytorch框架。60000张图片用来训练，10000张图片用来测试。
- 首先尝试pytorch中自带的vgg16的模型，后又对模型进行改进。

（三）结果分析：

- 仅对第二种vgg的改进方法进行结果分析：





- 在第10轮时，准确率达到了99.06%

```
————第10轮训练开始————
训练次数：8500，LOSS：0.1397731900215149
训练次数：9000，LOSS：0.023637883365154266
测试集上的LOSS：4.408009553309967
测试集上的准确率：0.9905999898910522
模型已保存
```

三、ResNet网络模型：

（一）代码：

- 先使用pytorch自带的resnet模型

```
#resnet.py
#构建神经网络
resnet = torchvision.models.resnet18()
#修改输入和输出
resnet.conv1 = nn.Conv2d(1,64,7,2,3,bias=False)
resnet.fc = nn.Linear(512,10)

#创建网络模型
resnet =resnet.to(device)
learning_rate = 0.001 #学习速率
#设置训练网络的一些参数
total_train_num =0
total_test_num=0
epoch =20
writer = SummaryWriter("resnet_logs")
```

- 发现效果不太好，所以进行改进：参考文章(29条消息) pytorch实现简单的ResNet并对MNIST进行分类_Sword丶的博客-CSDN博客

```
#构建神经网络
class ResidualBlock(nn.Module):
    """
    每一个ResidualBlock,需要保证输入和输出的维度不变
    所以卷积核的通道数都设置成一样
    """
    def __init__(self, channel):
        super().__init__()
        self.conv1 = nn.Conv2d(channel, channel, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(channel, channel, kernel_size=3, padding=1)

    def forward(self, x):
        """
        ResidualBlock中有跳跃连接;
        在得到第二次卷积结果时,需要加上该残差块的输入,
        再将结果进行激活,实现跳跃连接 ==> 可以避免梯度消失
        在求导时,因为有加上原始的输入x,所以梯度为: dy + 1,在1附近
        """
        y = F.relu(self.conv1(x))
        y = self.conv2(y)
        return F.relu(x + y)
```

```python
class ResNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=5)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=5)
        self.res_block_1 = ResidualBlock(16)
        self.res_block_2 = ResidualBlock(32)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(512, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = self.res_block_1(x)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = self.res_block_2(x)
        x = x.view(in_size, -1)
        x = self.fc1(x)
        return F.log_softmax(x, dim=1)
```

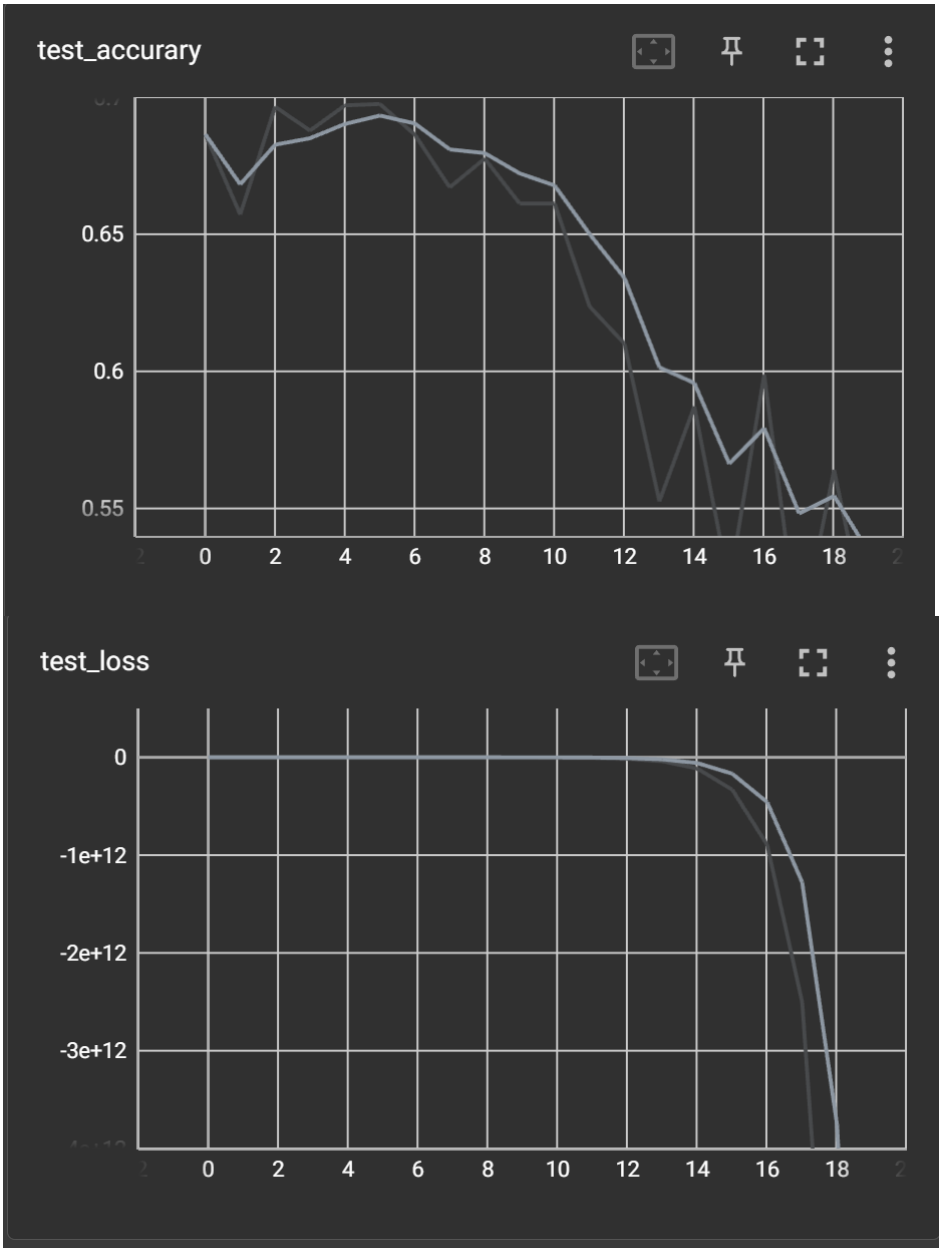
（二）实验过程：

- 使用了python语言，pytorch框架。60000张图片用来训练，10000张图片用来测试。
- 首先尝试pytorch中自带的rennet的模型，后又对模型进行改进。

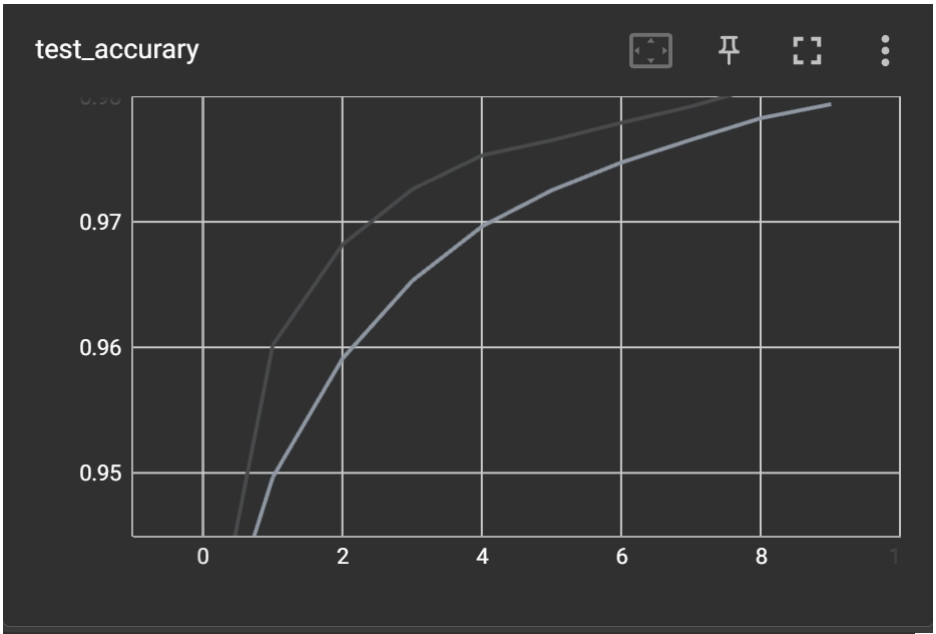
（三）结果分析：

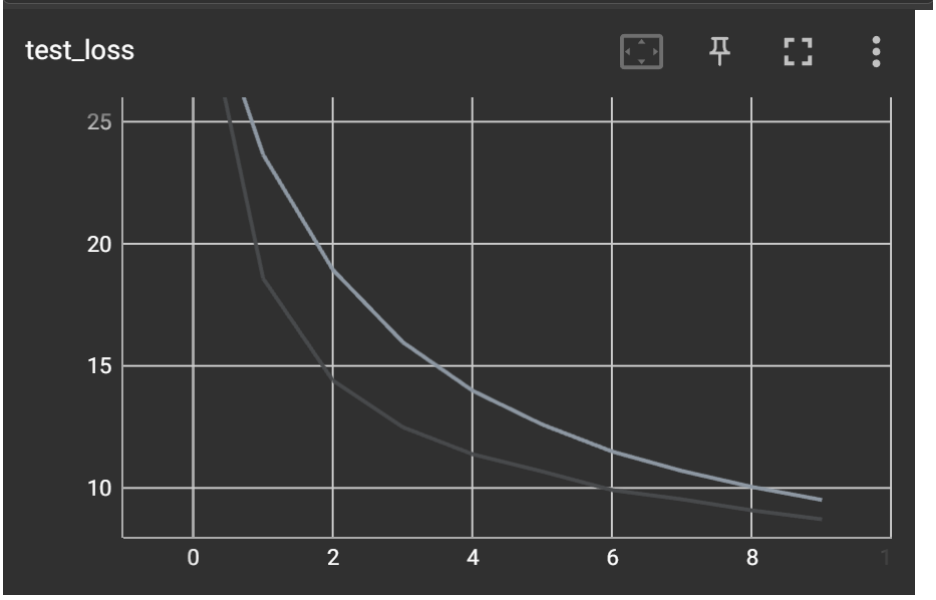
- pytorch自带的resnet模型，运行结果：在第6轮的时候，得到最高的准确率：约为69.75%。之后准确性会逐步下降。

```
————第6轮训练开始————
训练次数：4700，LOSS：-24448.03515625
训练次数：4800，LOSS：-28389.47265625
训练次数：4900，LOSS：-27522.92578125
训练次数：5000，LOSS：-32892.80078125
训练次数：5100，LOSS：-37207.9375
训练次数：5200，LOSS：-37016.8046875
训练次数：5300，LOSS：-46890.5078125
训练次数：5400，LOSS：-49476.86328125
训练次数：5500，LOSS：-60518.73046875
训练次数：5600，LOSS：-66162.265625
测试集上的总共的LOSS：-10650395.05859375
测试集上的准确率：0.6974999904632568
```



- 改进的resnet模型：在第10轮的时候，准确率达到约98.10%





————第10轮训练开始————
训练次数：8500，LOSS：0.08287594467401505
训练次数：9000，LOSS：0.035566654056310654
测试集上的总共的LOSS：8.717336808258551
测试集上的准确率：0.9809999465942383
模型已保存

四、LeNet网络模型：

（一）代码：

```
#构建神经网络
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Sequential(  # input_size=(1*28*28)
            nn.Conv2d(1, 6, 5, 1, 2),  # padding=2保证输入输出尺寸相同
            nn.ReLU(),  # input_size=(6*28*28)
            nn.MaxPool2d(kernel_size=2, stride=2),  # output_size=(6*14*14)
        )
```

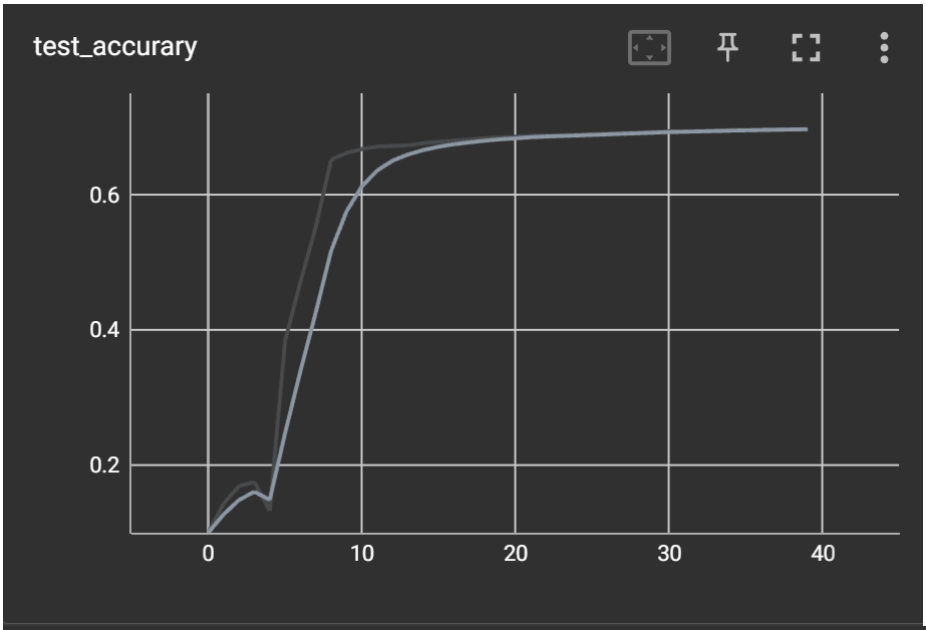
```
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.ReLU(),  # input_size=(16*10*10)
            nn.MaxPool2d(2, 2)  # output_size=(16*5*5)
        )
        self.fc1 = nn.Sequential(
            nn.Linear(16 * 5 * 5, 120),
            nn.ReLU()
        )
        self.fc2 = nn.Sequential(
            nn.Linear(120, 84),
            nn.ReLU()
        )
        self.fc3 = nn.Linear(84, 10)

    # 定义前向传播过程，输入为x
    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        # nn.Linear()的输入输出都是维度为一的值，所以要把多维度的tensor展平成一维
        x = x.view(x.size()[0], -1)
        x = self.fc1(x)
        x = self.fc2(x)
        x = self.fc3(x)
        return F.softmax(x, dim=1)
```

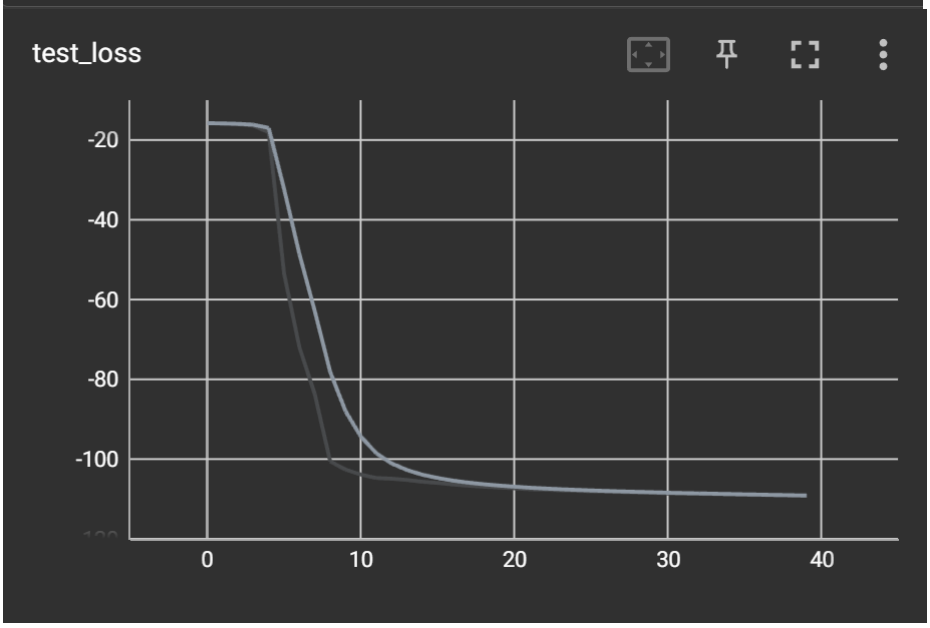
（二）实验过程：

- 使用了python语言，pytorch框架。60000张图片用来训练，10000张图片用来测试。
- 参考文章(29条消息) pytorch用LeNet5识别Mnist手写体数据集(训练+预测单张输入图片代码)_ZJE_ANDY的博客-CSDN博客

（三）结果分析：

- 在第40轮时，模型在测试集上的准确率达到约69.72%






```
————第40轮训练开始————
训练次数：37000，LOSS：-0.7973484992980957
训练次数：37500，LOSS：-0.6562403440475464
测试集上的总共的LOSS：-109.2149600982666
测试集上的准确率：0.6972000002861023
模型已保存
```

五、python实现卷积层：

（一）代码：

```python
import numpy as np

def my_conv(image,kernel,stride,padding):
    img_batch,img_channel,img_height,img_width = image.shape
    ker_num,ker_channel,ker_height,ker_width = kernel.shape
    #计算输出图片的size
    out_height = (stride+img_height+2*padding-ker_height)//stride
    out_width = (stride+img_width+2*padding-ker_width)//stride
```

```
    out=np.zeros((img_batch,ker_num,out_height,out_width))
    #原图片加padding
    image_padding =
np.zeros((img_batch,img_channel,img_height+2*padding,img_width+2*padding))

 image_padding[:,:,padding:padding+img_height,padding:padding+img_width]=image
    #计算输出的矩阵的值
    for f in range(ker_num):
        for i in range(out_height):
            for j in range(out_width):

 out[:,f,i,j]=np.sum(image_padding[:,:,i*stride:i*stride+ker_height,j*stride:j*s
tride+ker_width]*
                                    kernel[f,:,:,:],axis=(1,2,3))
    return out

#测试
if __name__=="__main__":
    stride = 1
    padding = 0
    #x是输入的图片，w是卷积核
    x_shape = (1, 1, 4, 4)
    w_shape = (1, 1, 2, 2)
    x = np.ones(x_shape)
    w = np.ones(w_shape)
    print("----image----\n",x)
    print(x.shape)
    print("----kernel----\n",w)
    print(w.shape)
    out = my_conv(x, w,stride,padding)
    print("----output:----\n",out)
    print(out.shape)
```

（二）测试结果：

- 结果正确：

```
E:\anaconda\envs\pytorch\python.exe D:
----image----
 [[[[1. 1. 1. 1.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]
   [1. 1. 1. 1.]]]]
(1, 1, 4, 4)
----kernel----
 [[[[1. 1.]
   [1. 1.]]]]
(1, 1, 2, 2)
----output:----
 [[[[4. 4. 4.]
   [4. 4. 4.]
   [4. 4. 4.]]]]
(1, 1, 3, 3)

Process finished with exit code 0
```