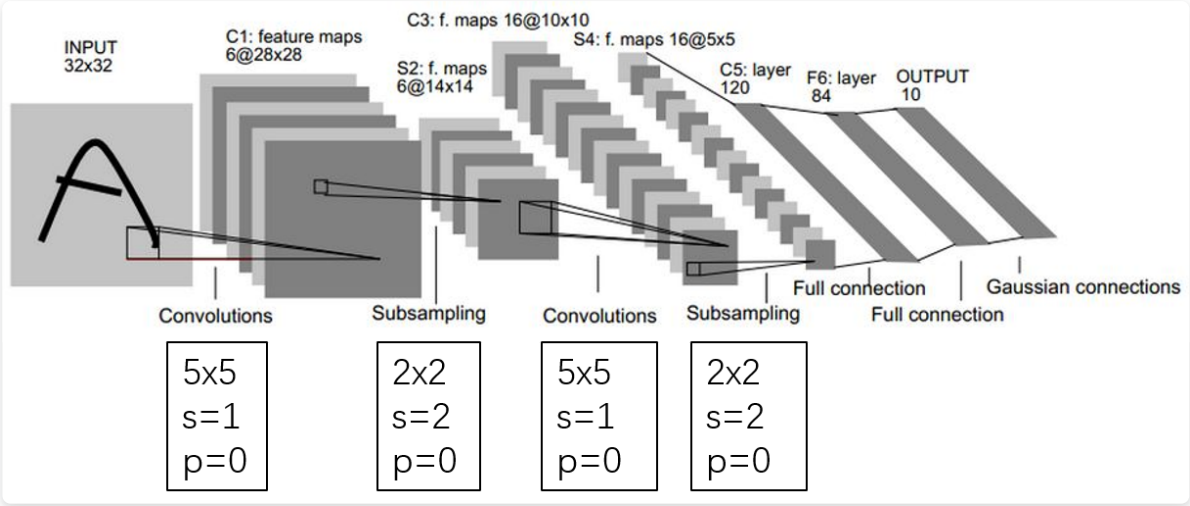
CNN-Lenet5实验

实验内容

- 实现卷积网络LeNet-5
- 实现图像多分类任务

实验模型



Lenet5卷积网络

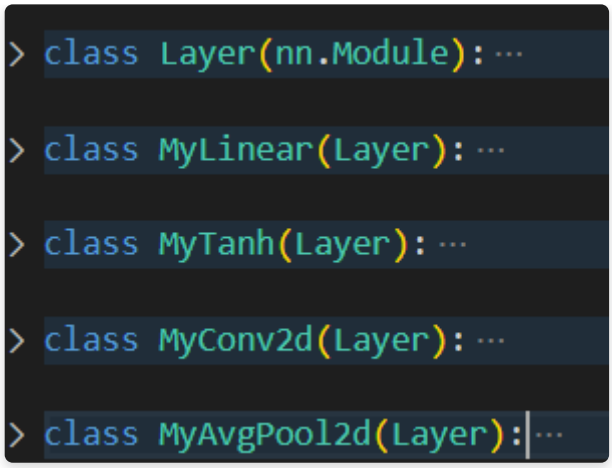
实验设计

数据集划分

数据集划分采用torchvision.datasets.MNIST类中自带的数据集划分方式，这里不做过多描述。

实验思路和代码框架

实验中通过继承torch..nn.Module,设计了Layer,MyLinear,MyTanh,MyConv2d和MyAvgPool2d五个类。这里对他们做简要说明。



实验中设计的几个类

1. Layer 类继承torch.nn.Module,并设置了一个cache用于存储临时值。

```
class Layer(nn.Module):
    def __init__(self):
        super(Layer,self).__init__()
        self.cache = None

    def _display(self):
        print(self.__class__.__name__)
```

Layer类

2. MyLinear类是对全连接层，即对nn.Linear功能的实现,主要实现forward函数的功能。

```
class MyLinear(Layer):
    def __init__(self, in_features: int, out_features: int, bias: bool = True):

    def reset_parameters(self) -> None:

    def forward(self, input: Tensor)->Tensor:
```

MyLinear类

函数功能说明:

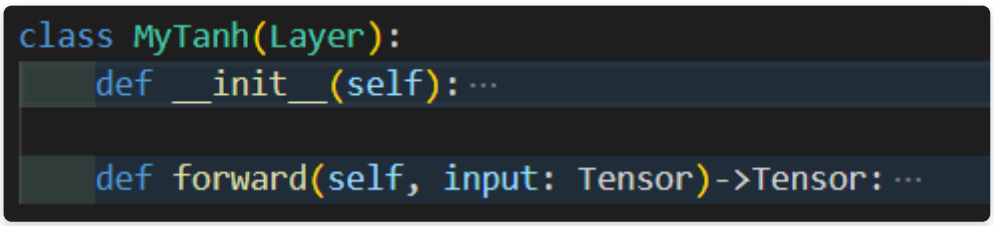
- `__init__`:初始化,生成一个全连接的线性层

  ```
  参数说明
  :param in_features: 输入特征数
  :param out_features: 输出特征数
  :param bias: 是否含有偏置量
  ```

- `reset_parameters`:权重初始化
- `forward`:前向传递

```
参数说明
:param input:输入
:return 输出
```

3. MyTanh类是对激活函数的实现，即激活函数tanh的实现，主要实现其forward功能。



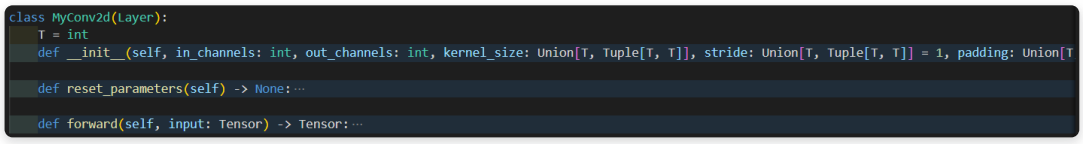
MyTanh类

函数功能说明:

- `__init__`:初始化
- `forward`:前向传递

```
参数说明
:param input:输入
:return 输出
```

4. MyConv2d类是对卷积层，即卷积层conv2d的实现，主要实现其forward功能。



MyConv2d类

函数功能说明:

- `__init__`:初始化,生成一个卷积层

```
参数说明
:param in_channels: 输入通道数
:param out_channels: 输出通道数数
:param kernel_size:  卷积核尺寸
:param stride: 移动步长
:param padding: 填充
```

- `reset_parameters`:权重初始化
- `forward`:前向传递,由于多重循环的方式时间复杂度过高，因此实验中借用unfold和fold函数将卷积转化为矩阵乘法，从而大大提高了卷积速率。

```
参数说明
:param input:输入
:return 输出
```

5. MyAvgPool2d类是对平均池化层，即均值池化层AvgPool2d的实现，主要实现其forward功能。同样，由于多重循环的性能较低，也采用了unfold将其转为矩阵运算，从而提高池化速率。

```
class MyAvgPool2d(Layer):
    T = int
    def __init__(self, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = None, padding: Union[T, Tuple[T, T]] = 0) -> None:···

    def forward(self, input: Tensor) -> Tensor:···
```

MyAvgPool2d类

函数功能说明:

- `__init__`:初始化
- `forward`:前向传递

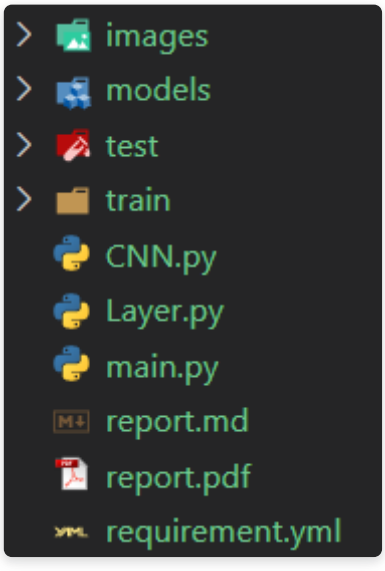
```
参数说明
:param input:输入
:return 输出
```

文件结构和代码运行说明

文件结构

提交文件目录如下:

- requirements.yml: 依赖的库(实验中只涉及torch,numpy和matplotlib等)
- **CNN.py**：实现的Lenet5模型
- **Layer.py**：继承nn.Module实现的层，包括卷积层，平均池化层，激活层和全连接层
- **report.md,report.pdf**：实验报告的markdown和pdf版本，方便查看
- models: 运行中若调用CNN中的save函数，将会将训练号的模型保存在该目录下
- train,test：训练和测试数据，无需赘述
- **main.py**: 可运行的文件



提交文件目录

运行说明

平台说明

- 开发工具: VSCode 1.50.1
- OS: Windows_NT x64 10.0.18363
- 编程语言: Python3.7.6
- 显卡: GeForce RTX 2060

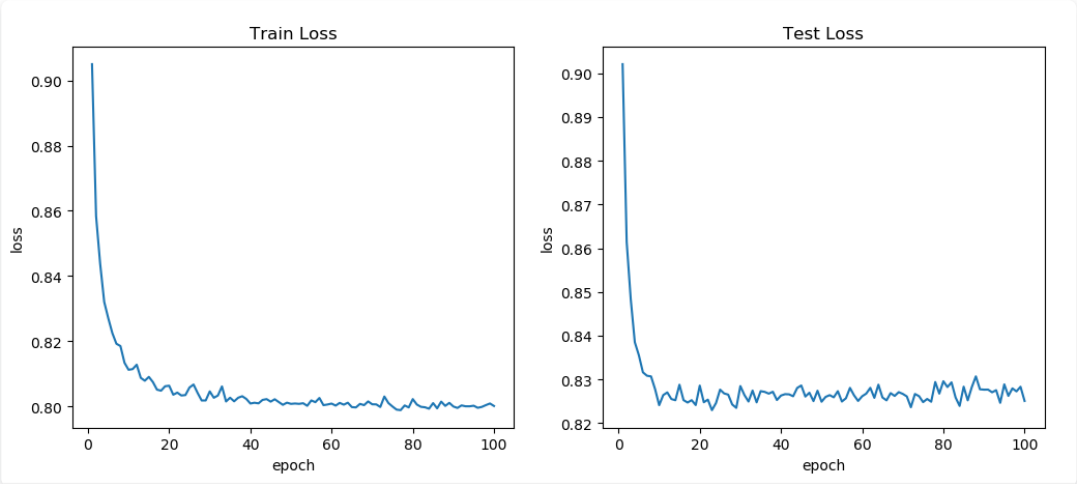
运行和运行结果

实际上，仅10次以内的迭代就可以获得正确率98%以上的分类性能，为了方便绘图，对参数设置如下（可在main函数中自由更改）：

```
# 参数设置
batch_size = 256
epoch = 100
lr = 0.002
# 采用Adam优化器
optimizer = Adam(model.parameters(), lr = lr ,betas = (0.9, 0.999), eps = 1e-6)
```

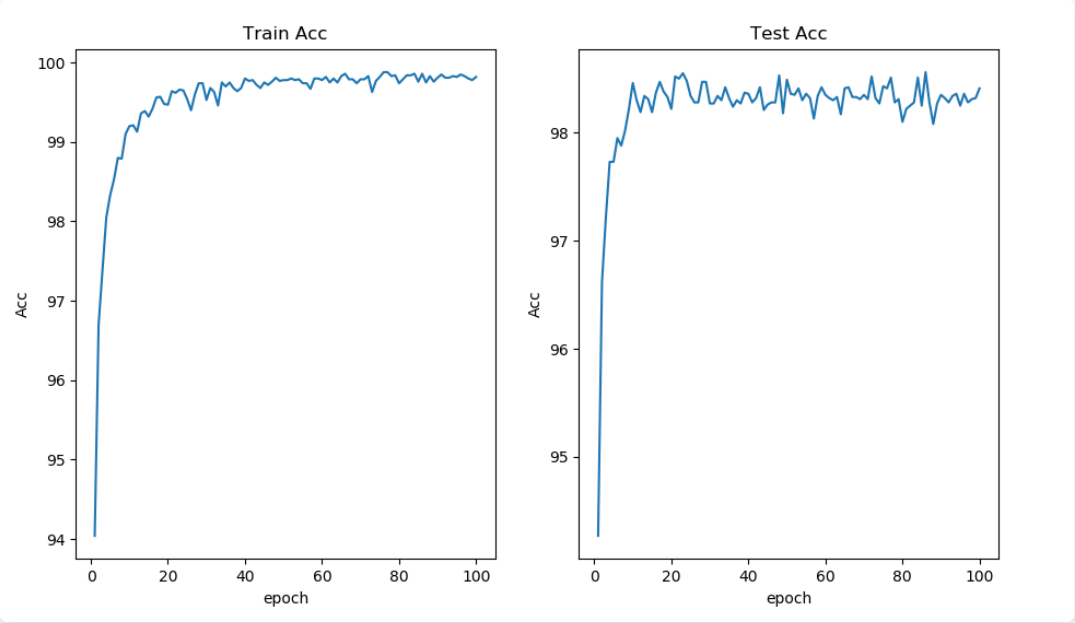
随后运行程序即可，如果希望保存模型，对`model.save()`取消注释即可，模型将默认保存在models目录下(路径也可以在save函数中国更改)。程序运行过程中将会

- 训练集和测试集上loss变化曲线，在一组实验中显示如下:



训练集，测试集loss曲线

- 训练集和测试集准确率变化曲线，在一组实验下显示如下:



训练集，测试集准确度曲线

训练结束后，终端会打印出最终的准确率结果。

```
训练集 accuracy: 99.82%
测试集 accuracy: 98.41%
```

终端输出结果

写在最后:

由于开发IDE使用Visual Studio Code, 并没有使用Pycharm，若存在运行方面的问题，请与我联系。