# Computer Vision Final Project


### 姓名：郑淇晟 学号：200110614

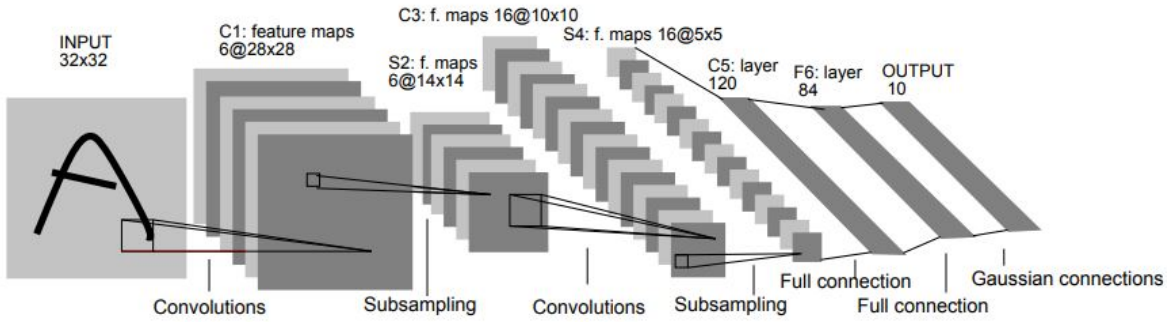

# 任务介绍

根据理论课以及前几次作业掌握的只是和技能，用Pytorch深度学习框架基于MNIST数据集实现一个手写数字识别模型，并探究不同的超参数对模型的影响。

# 模型结构

本实验参考LeNet5模型结构，设计了一个卷积神经网络，模型的结构与LeNet5基本一致，但在激活函数，池化层与原模型的实现稍有不同，同时增加BN层以加快神经网络收敛速度。

模型的基本结构如下图所示：




我们可以将模型分为前后两个部分

- 前半部分包括输入层，卷积层核池化层。用于提取图像特征，并降低参数维度。
- 后半部分包括最后的三层全连接层。用于利用抽象特征实现图像分类。

## 输入层

输入为一张28*28的图像，通过padding将其填充为32*32的图像，作为模型的输入

## 卷积层

模型共有两个卷积层，每个卷积层都由以下几个顺序操作构成：首先对输入的图像用5*5的卷积核进行卷积，然后经过一个BatchNorm层对卷积的结果进行归一化，最后经过激活函数的激活得到图像特征的隐藏表示

- 第一个卷积层使用6通道的5*5的卷积核，故输出维度为**(batch_size, 6, 28, 28)**
- 第二个卷积层使用16通道的5*5的卷积核，故输出维度为**(batch_size, 16, 10, 10)**

## 池化层

模型共有两个池化层，池化层都是跟在卷积层之后。它可以用于数据降维，减少模型的参数

- 第一个池化层使用最大池化，池化核为2*2大小，故输出维度为**(batch_size, 6, 14, 14)**
- 第二个池化层使用最大池化，池化核为2*2大小，故输出维度为**(batch_size, 16, 5, 5)**

### 全连接层

第二个池化层之后紧接着是三个全连接层，用于将模型前半部分提取到的抽象特征用于图像分类任务

- 第一个全连接层接收展平后的池化层输出作为输入，设定输出为120个神经元，故输出维度为**(batch_size, 120)**
- 第二个全连接层设定输出为84个神经元，故输出维度为**(batch_size, 84)**
- 第三个全连接层设定输出为类别数10，故输出维度为**(batch_size, 10)**

最后一层的输出表示图像对应某一个类别的分数

# 模型代码

如上`模型结构`节所述，模型的代码实现如下：

```python
class LeNet5(nn.Module):
    def __init__(self) -> None:
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(1, 6, 5, padding=2),
            nn.BatchNorm2d(6),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(6, 16, 5),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2, 2)
        )
        self.flatten = nn.Flatten()
        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, NUM_CLASSES)

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.flatten(out)
        out = self.fc1(out)
        out = self.fc2(out)
        out = self.fc3(out)
        return out
```

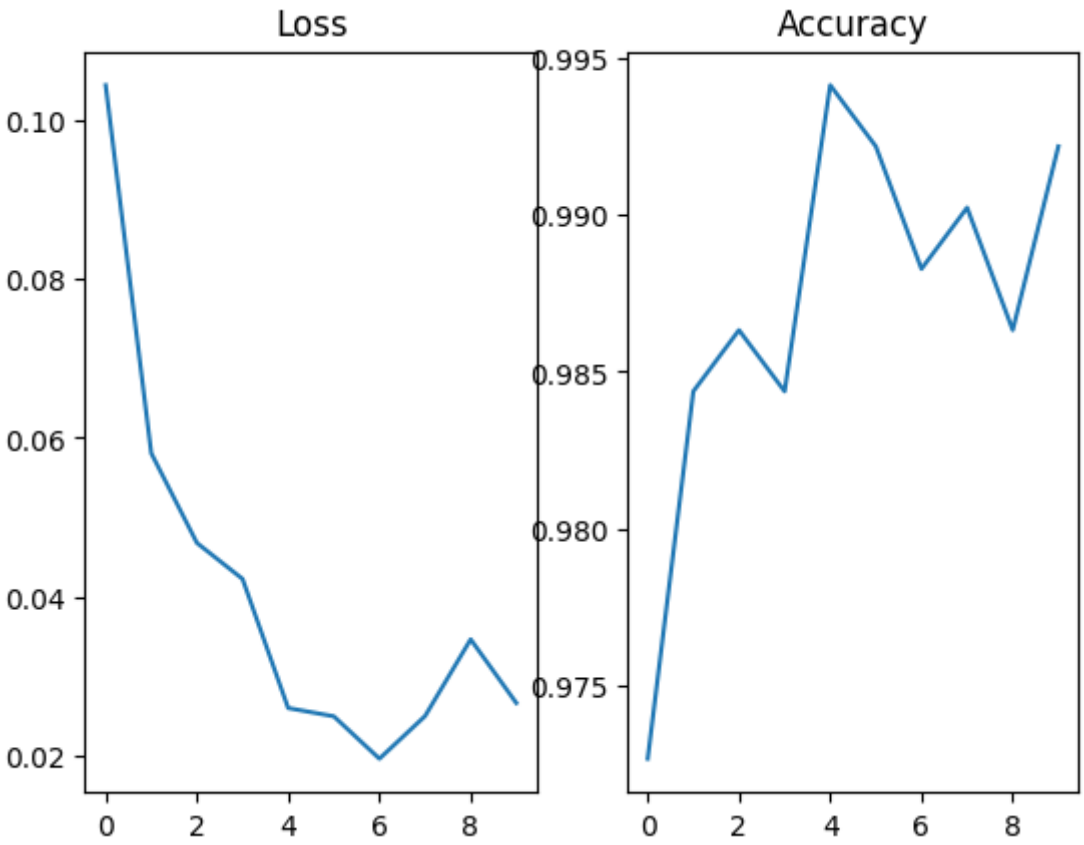
> 其他训练和测试代码请见代码文件

# 训练和测试效果

## 训练

训练时间：采用GPU训练的情况下训练10个epoch共花费2min

训练效果：前五个eopch训练时，loss迅速下降，后五个loss在小范围内震荡；准确率同理




## 测试

测试结果如下所示：整体准确率接近99%

```
batch 0 acc: 0.9921875
batch 1 acc: 0.982421875
batch 2 acc: 0.98046875
batch 3 acc: 0.978515625
batch 4 acc: 0.982421875
batch 5 acc: 0.98046875
batch 6 acc: 0.986328125
batch 7 acc: 0.986328125
batch 8 acc: 0.98828125
batch 9 acc: 0.986328125
batch 10 acc: 0.998046875
batch 11 acc: 0.990234375
batch 12 acc: 0.990234375
batch 13 acc: 0.9921875
batch 14 acc: 1.0
batch 15 acc: 0.99609375
batch 16 acc: 0.994140625
batch 17 acc: 0.998046875
batch 18 acc: 0.99609375
batch 19 acc: 0.9926470518112183
total acc 0.9895737767219543
```

# 亮点

- 数据处理方面：对输入的图像进行归一化

```
train_data = datasets.MNIST(root='data', train=True, download=True, transform=transforms.Compose([
                                transforms.ToTensor(),
                                transforms.Normalize((0.1307,), (0.3081,))
                            ]))
test_data = datasets.MNIST(root='data', train=False, download=True, transform=transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize((0.1307,), (0.3081,))
                       ]))
```

- 模型实现方面，有几个实现与原模型的实现不一样：
    1. 加入了BatchNorm层，加速模型收敛速度
    2. 激活函数采用ReLu函数，而非原模型的TanH函数
    3. 池化层采用MaxPooling，而非原模型的AvgPooling