

---

# 计算机视觉实践——练习 2\_LeNet5 实验报告

## 目录

|                  |   |
|------------------|---|
| 1 实验目的 .....     | 2 |
| 2 实验原理 .....     | 2 |
| 2.1 模型结构 .....   | 2 |
| 2.1.1 整体框架 ..... | 2 |
| 2.1.2 卷积层 .....  | 3 |
| 2.1.3 池化层 .....  | 3 |
| 2.1.4 激活函数 ..... | 3 |
| 2.1.5 全连接层 ..... | 4 |
| 2.2 损失函数 .....   | 5 |
| 2.3 优化器 .....    | 5 |
| 3 实验 .....       | 5 |
| 3.1 数据集 .....    | 5 |
| 3.2 实验细节 .....   | 6 |
| 3.3 实验结果 .....   | 6 |
| 3.4 实验分析 .....   | 6 |
| 附录——代码展示 .....   | 7 |

# 1 实验目的

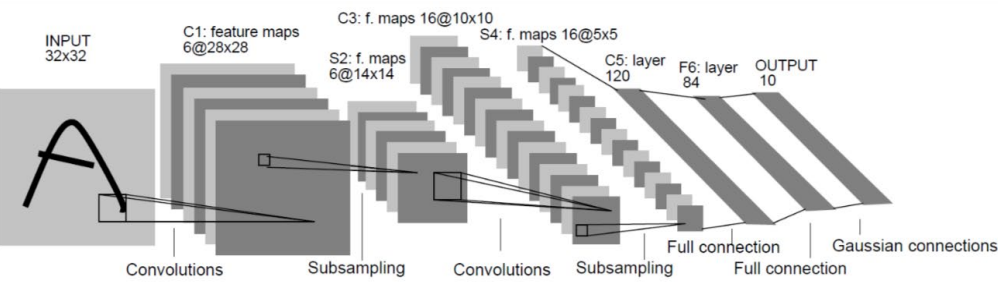
- 熟悉早期卷积神经网络的基本结构，包括卷积层、池化层、激活函数、全连接层等。
- 学习经典手写数字识别网络 LeNet。
- 在 MNIST 数据集上完成图像分类任务，包括训练与测试。

# 2 实验原理

主要包括模型结构、损失函数和优化器 3 个部分。

## 2.1 模型结构

### 2.1.1 整体框架



(a)

|        | Layer           | Feature Map | Size  | Kernel Size | Stride | Activation |
|--------|-----------------|-------------|-------|-------------|--------|------------|
| Input  | Image           | 1           | 32x32 | -           | -      | -          |
| 1      | Convolution     | 6           | 28x28 | 5x5         | 1      | tanh       |
| 2      | Average Pooling | 6           | 14x14 | 2x2         | 2      | tanh       |
| 3      | Convolution     | 16          | 10x10 | 5x5         | 1      | tanh       |
| 4      | Average Pooling | 16          | 5x5   | 2x2         | 2      | tanh       |
| 5      | Convolution     | 120         | 1x1   | 5x5         | 1      | tanh       |
| 6      | FC              | -           | 84    | -           | -      | tanh       |
| Output | FC              | -           | 10    | -           | -      | softmax    |

(b)

图 1 LeNet5 框架

本次实验中，对 LeNet5 做了大致的复现，取消了最后一个全连接层后的激活函数（即图 1(b)中最后一个 FC 中的 softmax），取消 softmax 激活函数是因为本次实验使用的损失函数为 Pytorch 自带的交叉熵损失函数，其中自带 softmax。

### 2.1.2 卷积层

卷积层（Convolutional Layer）通过将输入特征图与一组可学习的卷积核（也称为滤波器）进行卷积操作（严格来说为互相关运算）来产生输出特征图。卷积操作可以看作是将卷积核在输入特征图上滑动，并计算在每个位置上卷积核与输入特征图的乘积之和。

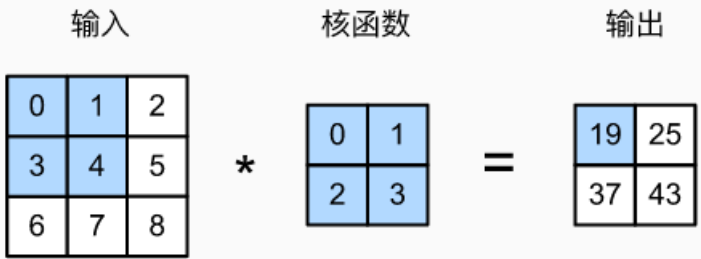


图 2 卷积层的互相关运算

### 2.1.3 池化层

本次实验使用的池化层为平均池化层。

平均池化层的作用是提取输入特征图的平均特征，同时降低特征图的空间分辨率。由于每个池化区域的像素平均值只是输入特征图的一个近似值，因此平均池化可能会损失一些信息。然而，在某些情况下，这种信息的损失可以被认为是有益的，因为它可以防止过拟合和提高模型的泛化能力。

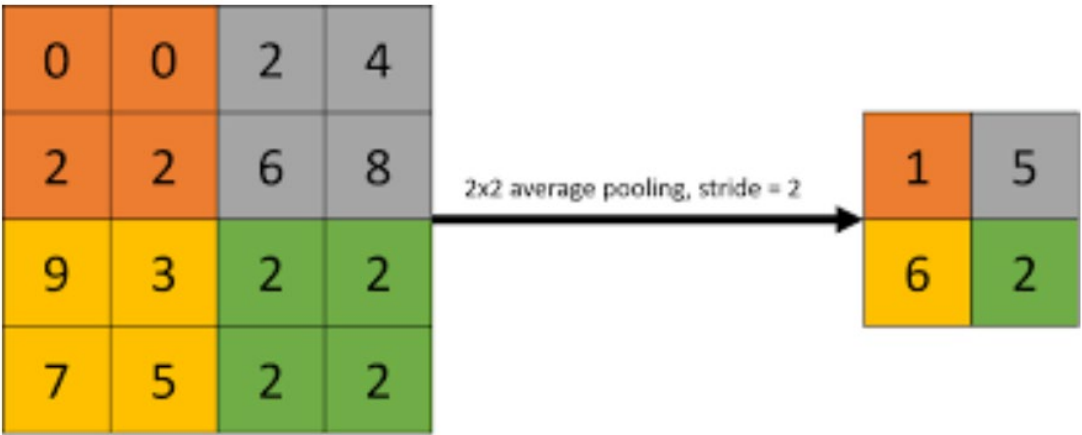


图 3 平均池化层的运算

### 2.1.4 激活函数

激活函数是深度神经网络中的一个非线性函数，它的作用是给神经网络增加非线性变换的能力。激活函数通常被放置在每个神经元的输出上，以便通过激活函数将神经元的输出转换为非线性形式。常见的激活函数有 sigmoid、ReLU、tanh 等。

激活函数的作用是引入非线性变换，使神经网络可以更好地逼近复杂函数。例如，在没有激活函数的情况下，多个线性变换的组合仍然是线性的，这样神经网络就无法处理非线性模式。激活函数的引入可以将线性模型转换为非线性模型，

从而使神经网络可以更好地逼近非线性函数。

本次实验中，使用的激活函数为 Tanh。Tanh 函数可以看作是 sigmoid 激活函数的变形，它的输出值也在 0 到 1 之间，但是当输入值为负时，它的输出值为负，可以产生更大的负梯度，因此在一些场景下具有优势。与 sigmoid 函数相比，Tanh 函数的输出值范围更广，因此它可以对输入数据的分布进行更多的变换。但是，当输入值很大或很小时，Tanh 函数的梯度会接近于 0，这也会导致梯度消失的问题。因此，在深度神经网络中，通常使用 ReLU 或其变体作为激活函数，但本次实验中为复现 LeNet5，故采用 Tanh 激活函数。

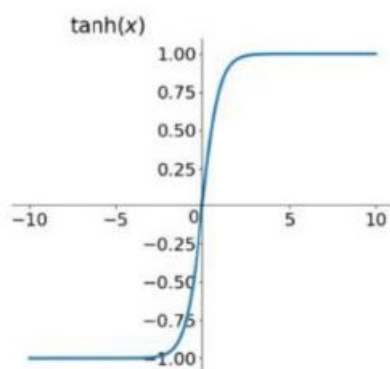


图 4 Tanh 激活函数图像

### 2.1.5 全连接层

全连接层（Full Connected Layer, FC）是神经网络中的一种基本层，也被称为密集连接层（Dense Layer），它的每个输入神经元都连接到输出层的每个神经元，因此全连接层的权重矩阵是一个二维矩阵。卷积神经网络中全连接层的作用主要是作为分类器，将学到的特征映射到对应的样本类别。

在 Pytorch 中，全连接层由 `nn.Linear()` 即线性层实现。

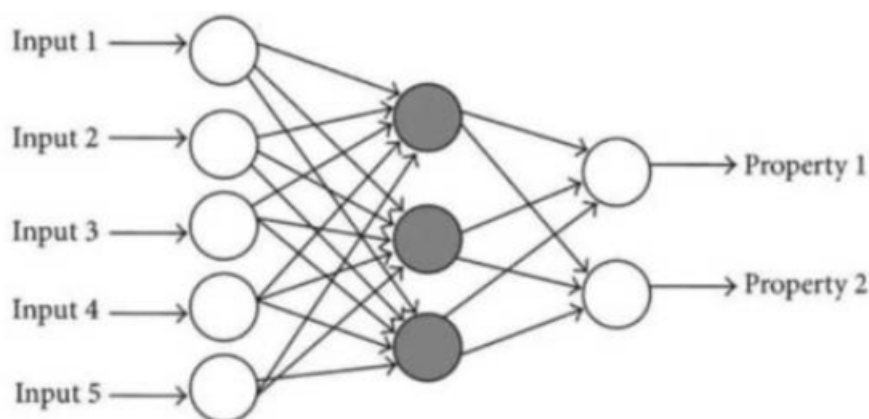


图 5 全连接层

## 2.2 损失函数

本次实验选取的损失函数为交叉熵损失函数 (Cross Entropy Loss)，由 Pytorch 中的 `nn.CrossEntropyLoss` 实现。

Cross Entropy 公式如下：

$$L = - \sum_{c=1}^C y_c \log(p_c)$$

此处的  $p_c$  也即网络输出层的输出经 softmax 后的结果， $y_c$  为 one-hot 编码，用于单标签多分类， $C$  为标签总共的类别数。

在网络训练过程中，对损失函数进行梯度下降进行优化，训练网络中的参数。

## 2.3 优化器

在深度学习中，优化器 (Optimizer) 是一种用于优化神经网络参数的算法，它的主要目标是通过调整神经网络的权重和偏置，使神经网络的输出尽可能接近真实标签，从而最小化损失函数。

优化器的工作原理是通过计算损失函数对参数的梯度，并根据梯度的方向和大小来更新参数。常见的优化算法包括：随机梯度下降 (Stochastic Gradient Descent, SGD)、动量 (Momentum)、AdaGrad、Adam 等。

本次实验中，使用了 Adam 优化器，Adam 是一种自适应学习率算法，它结合了动量和自适应学习率的优点，具有收敛速度快、精度高等优点。

同时，本次实验中，学习率采取等间隔调整，每 20 个 Epoch 降低到前一学习率的一半，初始学习率为  $10^{-5}$ 。

## 3 实验

### 3.1 数据集

本次实验的数据集为 MNIST，实验中采用 Pytorch 内置的 MNIST 数据集函数读取，不过实验外也附带了解码 MNIST 源数据集文件的代码。



0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3  
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4  
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5  
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6  
7 7 7 7 7 7 7 7 7 7 7 7 7 7 7  
8 8 8 8 8 8 8 8 8 8 8 8 8 8 8  
9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

图 6 MNIST 数据集

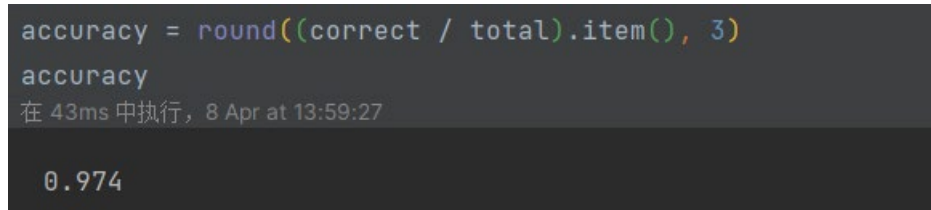
---

## 3.2 实验细节

对训练集进行了数据增强——随机翻转，并对输入的图片调整到大小为  $32 \times 32$ 。训练的轮次为 300，每次训练的 mini-batch 为 128，对训练的数据进行打乱。学习率采取等间隔调整，每 20 个 Epoch 降低到前一学习率的一半，初始学习率为  $10^{-5}$ 。

## 3.3 实验结果

本次实验中，训练的模型对 MNIST 的测试结果的准确为 97.4%，如图 7 所示。



```
accuracy = round((correct / total).item(), 3)
accuracy
在 43ms 中执行, 8 Apr at 13:59:27
0.974
```

图 7 测试结果准确率

## 3.4 实验分析

本次实验的模型为复现模型，故没有使用现在较好的激活函数如 ReLU 或 SiLU 等；同时并未在网络中加入批归一化层（Batch Normalization Layer），此方法为用于深度神经网络的正则化方法，旨在解决神经网络训练过程中的梯度消失和梯度爆炸问题，并加速神经网络的收敛速度。

另外，如果在网络中使用 resnet 结构也许会有更好的效果。

---

## 附录——代码展示

仅展示主要代码。

模型代码：

```
class LeNet5(nn.Module):
    """ iAffected """
    def __init__(self):
        super().__init__()
        self.backbone = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Tanh(),
            nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5),
            nn.Tanh(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Tanh(),
            nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5),
            nn.Tanh(),
            nn.Flatten(start_dim=1),
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, 10),
        )

    """ iAffected """
    def forward(self, x):
        return self.backbone(x)
```

训练代码

```
epochs = 300
for epoch in range(epochs):
    train_model.train()
    mloss = torch.zeros(1, device=device) # mean_loss
    pbar = tqdm(enumerate(train_loader), total=len(train_loader), desc=f'Epoch {epoch}/{epochs}', unit='batches')

    for i, (imgs, labels) in pbar:
        imgs, labels = imgs.to(device), labels.to(device)
        preds = train_model(imgs)
        loss = loss_fn(preds, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        mloss = (mloss * i + loss) / (i + 1)
        mem = f'{torch.cuda.memory_reserved() / 1e9 if torch.cuda.is_available() else 0:.3g}G' # GPU_mem
        pbar.set_postfix(loss=mloss.item(), GPU_mem=mem)

    ckpt = { # checkpoint
        'epoch': epoch,
        'model': deepcopy(train_model).half(),
        'optimizer': optimizer.state_dict(),
    }
    torch.save(ckpt, 'LeNet5.pt')
```

## 测试代码及结果：

```
correct = torch.zeros(1, device=device)
total = torch.zeros(1, device=device)

with torch.no_grad():
    pbar = tqdm(enumerate(test_loader), total=len(test_loader), desc='Test', unit='batches')

    for i, (imgs, labels) in pbar:
        imgs, labels = imgs.to(device), labels.to(device)
        preds = test_model(imgs)
        preds = torch.argmax(nn.Softmax(dim=1)(preds), dim=1) # 将预测结果经softmax后取最大值的序号为预测标签

        total += torch.tensor(labels.size(0))
        correct += (preds == labels).sum()

在 2s 中执行, 8 Apr at 13:59:27

Test: 100%|██████████| 79/79 [00:02<00:00, 28.94batches/s]

accuracy = round((correct / total).item(), 3)
accuracy
在 43ms 中执行, 8 Apr at 13:59:27

0.974
```