浙江大学
ZHEJIANG UNIVERSITY

计算机视觉（本科）实验报告

作业名称 Learning CNN

姓　　名 杨逍宇

学　　号 3220105453

电子邮箱 1877114891@qq.com

联系电话 13518290755

导　　师 潘纲

2024 年 12 月 15 日

1 已实现的功能简述及运行简要说明

1.1 已实现的功能简述：

程序运行之后，需要手动调整是需要训练新的模型还是加载已经训练好的模型。

- 训练新的模型：训练过程中会给出训练轮数和当前的 `Loss`，运行结束后会将训练的模型保存在指定路径之下，并且显示出模型在测试集上的效果，同时显示 10 张图片的 `label` 及其 `pred` 预测值。
- 加载已经训练好的模型：程序会从指定路径加载模型，并且显示出模型在测试集上的效果，同时显示 10 张图片的 `label` 及其 `pred` 预测值。

1.2 运行简要说明：

(1). 其中 python 文件 `hw_4` 在目录 `src` 下, 运行所依赖的数据集保存在 `dataset` 目录下，运行过程图片保存在 `assets` 目录下，训练结果模型保存在 `model` 目录下，相关文档保存在 `docs` 目录下。

(2). 其中 `LeNet_test.py` 是针对 MNIST 数据集进行的 `LeNet5` 网络模型训练的文件。

(3). 其中 `calssifier.py` 是针对 CIFAR-10 数据集进行的 `MobileNetV1` 网络模型训练的文件。

(4). 其中 `calssifier_resnet.py` 是针对 CIFAR-10 数据集进行的 `ResNet` 网络模型训练的文件

2 开发与运行环境

本实验使用的软件和工具如下：

- 开发环境：Win11,CUDA 12.6.0
- 编程语言：python
- 库：pytorch 2.4.1+cu124;torchvision 0.19.1+cu124

3 手写数字识别

3.1 算法基本思路

这个实验采用 LeNet-5 模型进行图像分类任务。算法的基本思路如下：

(1). 数据加载：使用 DataLoader 加载训练集和测试集数据，设置合适的 batch size，并对训练集数据进行 shuffle。

(2). 模型训练：定义 train_model 函数，输入参数包括模型、训练数据加载器、损失函数、优化器、训练轮数和设备。

(a) 将模型加载到指定设备（这里我配置 CUDA 环境，使用 GPU 进行训练）。

(b) 设置模型为训练模式。

(c) 初始化损失列表，用于记录每个 epoch 的损失。

(d) 迭代训练，每个 epoch 中：

i. 初始化 running_loss 为 0。

ii. 遍历训练数据加载器，获取批量图像和标签，并将其加载到指定设备。

iii. 前向传播：将图像输入模型，获取输出。

iv. 计算损失：使用损失函数计算模型输出与真实标签之间的损失。

v. 反向传播：清零优化器梯度，进行反向传播，更新模型参数。

(3). 模型评估：在测试集上评估模型性能，计算准确率等指标。

通过上述步骤，算法实现了图像分类任务。具体来说，算法通过以下步骤实现了这一目标：

(1). **数据加载**：将训练集和测试集数据加载到内存中，并进行批量处理，以提高训练效率。

(2). **模型训练**：通过前向传播和反向传播，使用训练数据对 LeNet 模型进行训练，优化模型参数，使其能够更好地拟合训练数据。

(3). **模型评估**：在测试集上评估训练好的模型，计算模型的准确率等性能指标，以验证模型的泛化能力。

3.2 算法实现要点

算法实现的主要步骤如下：

(1). **数据加载**：使用 DataLoader 加载训练集和测试集数据，设置合适的 batch size，并对训练集数据进行 shuffle。

```
    train_loader = DataLoader(dataset=train_dataset,
                              batch_size=batch_size,
                              shuffle=True)

    test_loader = DataLoader(dataset=test_dataset,
                             batch_size=batch_size,
                             shuffle=False)
    return train_loader, test_loader
```

(2). **模型训练**：定义 train_model 函数，输入参数包括模型、训练数据加载器、损失函数、优化器、训练轮数和设备。

```
    def train_model(model, train_loader, criterion, optimizer,
        num_epochs, device):
        model.to(device)
        model.train()
        loss_list = []
        for epoch in range(num_epochs):
            running_loss = 0.0
            for i, (images, labels) in enumerate(train_loader):
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)

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

                running_loss += loss.item()
            loss_list.append(running_loss / len(train_loader))
```

(3). **模型评估**：在测试集上评估模型性能，计算准确率等指标。

```
    def evaluate_model(model, test_loader, device):
        model.to(device)
        model.eval()
        correct = 0
```

```
        total = 0
        with torch.no_grad():
            for images, labels in test_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        accuracy = correct / total
        return accuracy
```

3.3 实验结果及分析

我们设置相关参数如下:

```
batch_size = 64
learning_rate = 0.001
num_epochs = 10
```

训练过程中的 Loss 曲线如下:

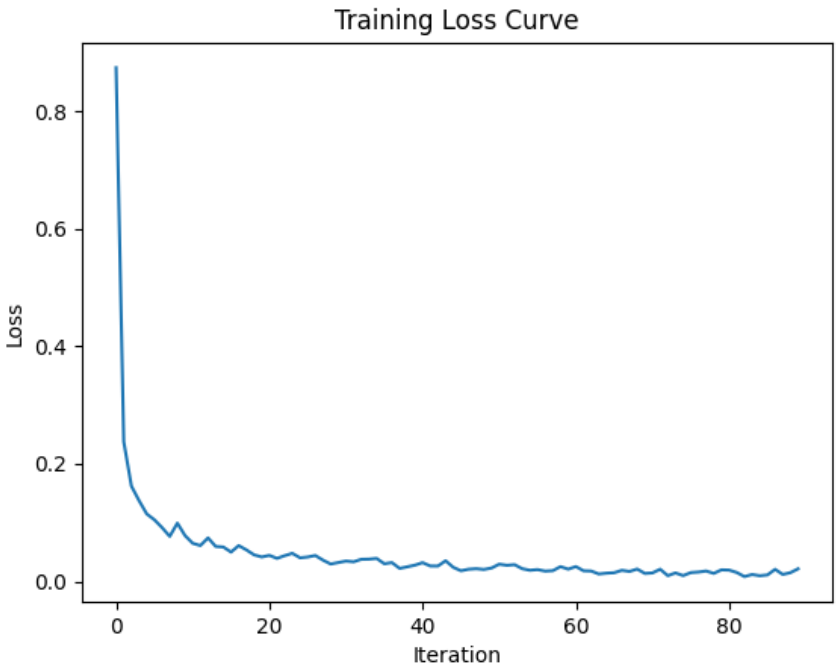

Figure 1: LeNet-5 Loss

得到模型的实验结果如下图所示:

```
model.load_state_dict(torch.load('./model/lenet5_model.pth'))
Model loaded from lenet5_model.pth
Accuracy of the model on the 10000 test images: 98.88 %
```

Figure 2: LeNet-5 result

从图中可以看到，我们模型在 MNIST 的 1 万测试样本获得的识别率达到 98.88%，可见取得了较好的效果。效果如下：

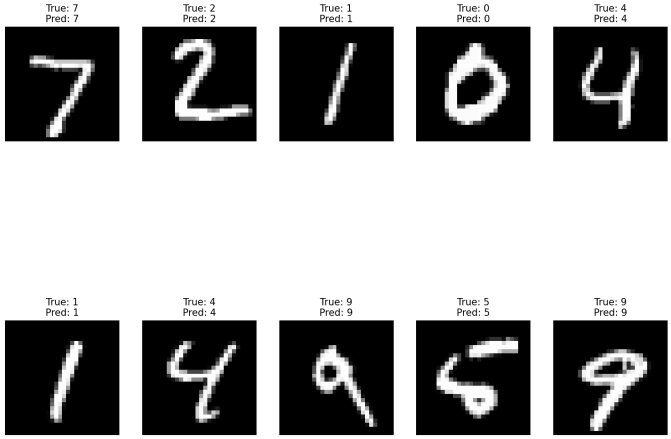

Figure 3: LeNet-5 show

4 物体分类

4.1 算法基本思路

这个实验采用 ResNet34 模型进行图像分类任务。算法的基本思路如下：

(1). 数据预处理: 对 CIFAR-10 数据集进行随机裁剪、水平翻转、随机旋转和颜色抖动等操作，并进行归一化处理。对训练集进一步划分验证集和训练集。

(2). 使用 ResNet34 模型进行图像分类任务。ResNet34 模型由多个残差块（Residual Block）组成，每个残差块包含多个卷积层。

(3). 在模型的构建过程中，定义了四个卷积层组（conv2_x, conv3_x, conv4_x, conv5_x），每个卷积层组包含多个残差块。

(4). 损失函数和优化器：定义交叉熵损失函数，并使用带动量的随机梯度下降（SGD）优化器。

(5). 学习率调度器：使用 StepLR 调度器，每隔一定的 epoch 数降低学习率。

(6). 模型训练：定义 train_model 函数，输入参数包括模型、训练数据加载器、损失函数、优化器、学习率调度器、训练轮数和设备。

(a) 将模型加载到指定设备（这里我配置 CUDA 环境，使用 GPU 进行训练）。

(b) 设置模型为训练模式。

(c) 初始化损失列表，用于记录每个 epoch 的损失。

(d) 迭代训练，每个 epoch 中：

i. 初始化 running_loss 为 0。

ii. 遍历训练数据加载器，获取批量图像和标签，并将其加载到指定设备。

iii. 前向传播：将图像输入模型，获取输出。

iv. 计算损失：使用损失函数计算模型输出与真实标签之间的损失。

v. 反向传播：清零优化器梯度，进行反向传播，更新模型参数。

vi. 更新学习率：使用学习率调度器更新学习率。

(7). 模型保存：训练完成后，将模型参数保存到文件中。

(8). 模型评估：在测试集上评估模型性能，计算准确率等指标。

4.2 算法实现要点

算法实现的主要步骤如下：

(1). **基本残差块的定义**：

- 残差块由两个卷积层和一个捷径连接（shortcut）组成。
- 每个卷积层包含卷积操作、批量归一化（Batch Normalization）和 ReLU 激活函数。

```
1     # 定义基本的残差块
2  class BasicBlock(nn.Module):
3      expansion = 1
4
5      def __init__(self, in_channels, out_channels, stride=1):
6          super(BasicBlock, self).__init__()
7          self.residual_function = nn.Sequential(
8              nn.Conv2d(in_channels, out_channels, kernel_size=3, stride
                   =stride, padding=1, bias=False),
9              nn.BatchNorm2d(out_channels),
10             nn.ReLU(inplace=True),
```

```
            nn.Conv2d(out_channels, out_channels * BasicBlock.
                expansion, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels * BasicBlock.expansion)
        )

        self.shortcut = nn.Sequential()

        if stride != 1 or in_channels != BasicBlock.expansion *
            out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * BasicBlock.
                    expansion, kernel_size=1, stride=stride, bias=False
                    ),
                nn.BatchNorm2d(out_channels * BasicBlock.expansion)
            )

    def forward(self, x):
        return nn.ReLU(inplace=True)(self.residual_function(x) + self
            .shortcut(x))
```

(2). **定义 ResNet 模型**:

- 首先定义卷积层函数 _make_layer: 每个卷积层组包含多个残差块（Residual Block），每个残差块由多个卷积层组成。
- 使用 AdaptiveAvgPool2d 进行全局平均池化，并使用全连接层（fc）输出分类结果。

```
        # 定义ResNet模型
class ResNet(nn.Module):
    def __init__(self, block, num_block, num_classes=10):
        super(ResNet, self).__init__()

        self.in_channels = 64

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2_x = self._make_layer(block, 64, num_block[0], 1)
```

```
        self.conv3_x = self._make_layer(block, 128, num_block[1], 2)
        self.conv4_x = self._make_layer(block, 256, num_block[2], 2)
        self.conv5_x = self._make_layer(block, 512, num_block[3], 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, out_channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride
                ))
            self.in_channels = out_channels * block.expansion

        return nn.Sequential(*layers)

    def forward(self, x):
        output = self.conv1(x)
        output = self.conv2_x(output)
        output = self.conv3_x(output)
        output = self.conv4_x(output)
        output = self.conv5_x(output)
        output = self.avg_pool(output)
        output = output.view(output.size(0), -1)
        output = self.fc(output)

        return output
```

(3). **数据预处理**:

- 对 CIFAR-10 数据集进行随机裁剪、水平翻转、随机旋转和颜色抖动等操作。
- 对数据进行归一化处理，使其均值为 0，标准差为 1。

```
transform = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15), # 添加随机旋转
    transforms.ColorJitter(brightness=0.2, contrast=0.2,
        saturation=0.2, hue=0.2), # 添加颜色抖动
```

```
6         transforms.ToTensor(),
7         transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023,
              0.1994, 0.2010)),
8     ])
```

(4). **数据集划分**:

- 加载 CIFAR-10 数据集，并将其划分为训练集和验证集。
- 将训练集划分为训练集和验证集，比例为 8:2。
- 加载测试集用于模型评估。

4.3 实验结果及分析

4.3.1 实验结果

实验中使用了两个模型，进行了多次训练，其中训练模型效果最好的相关参数如下：我们设置相关参数如下:

```
1   batch_size = 128
2   learning_rate = 0.01
3   num_epochs = 50
```

训练过程中的 Loss 曲线如下:

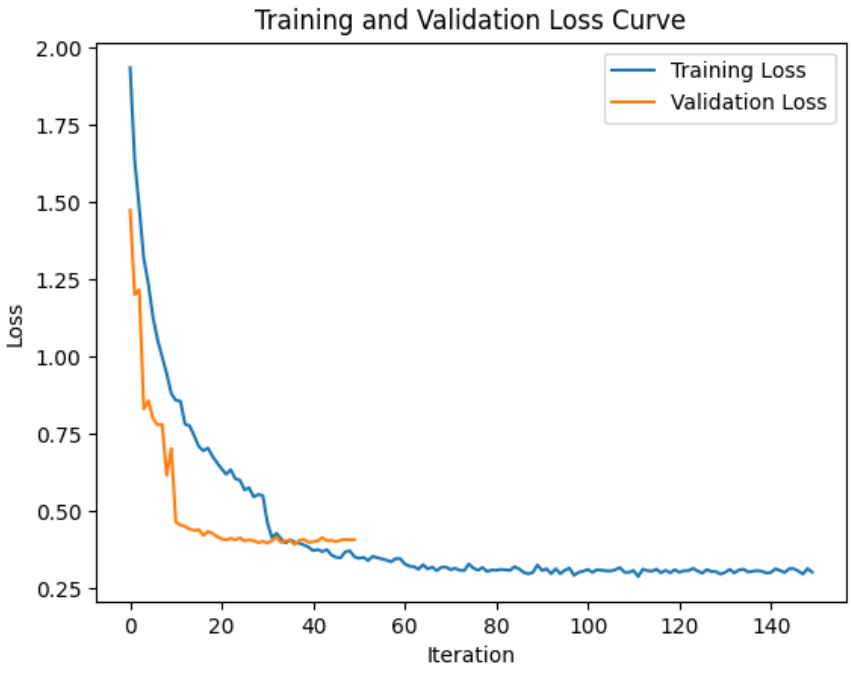

Figure 4: ResNet Loss

得到模型的实验结果如下图所示:

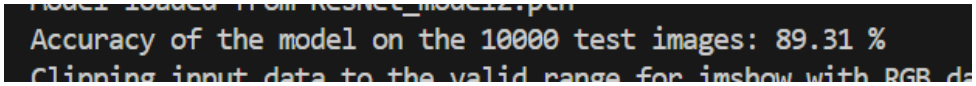

Figure 5: ResNet result

从图中可以看到，我们模型在 CIFAR-10 的测试样本获得的识别率达到 89.31%，可见取得了不错的效果。效果如下:

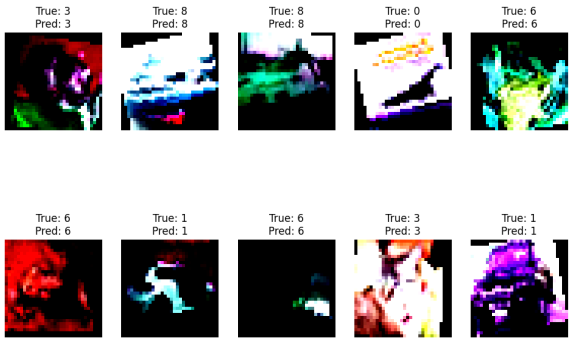

Figure 6: ResNet show

4.3.2 实验分析

这个实验中我一共使用了 MobileV1,ResNet18,ResNet34 三个模型，想要尝试使用 ResNet50 这类的深层网络但是电脑运行内存不够，可惜。不过三个模型取得的训练效果均不够理想，我在模型定义好的基础上做了较多的实验:

- 数据增强: 首先进行了数据增强，对数据添加了随机旋转，水平翻转以及颜色抖动等操作。然后根据 CIFAR-10 数据集的特征做了归一化处理，使其均值为 0，标准差为 1。这样我能提升模型对数据变化的鲁棒性，并且加快模型的收敛速度。这样使得训练结果能够从百分之七十多到达百分之八十左右。
- 优化器的选择: 对于 Adam 优化器和 SGD 优化器均进行过实验，但是效果相差不打，为了模型能够加快收敛且能够自适应学习率因此最后的训练模型选择了 Adam 优化器。
- 学习率调度器的选择: 在实验中选择最简单的 StepLR 和 CosineAnnealingLR，虽然前者较为简单，且参数需要手动经验设置，但是在调整后能取得较好的结果。而 CosineAnnealingLR 使得学习率变化平滑，但在训练后面几轮我们发现 loss 的值几乎不变模型已经收敛，因此对于此数据集仍然选用 StepLR 学习率调度器。

- 学习率的选择：为了训练开始能使得 loss 能够较快下降，因此选择了较大的学习率，但是实验后期发现 loss 几乎不变可能是学习率过大，不过经过实验设置学习率为 0.1 or 0.01 or 0.001 对实验最终结果影响不大。

5 结论与心得体会

在本次实验中，我们实现了三个卷积神经网络（CNN）：LeNet-5、MobileNetV1 和 ResNet。实验过程中，我们使用了 PyTorch 等常用的深度学习开发库，并通过这些库提供的构建函数和训练接口完成了网络的搭建和训练。以下是实验的具体结论与心得体会：

5.1 实验结论

1. LeNet-5 的实现与训练：- 我们使用 MNIST 手写数字数据集对 LeNet-5 进行了训练。该数据集包含 6 万个训练样本和 1 万个测试样本。- 在训练过程中，我们对网络进行了多次迭代，并使用交叉熵损失函数和 Adam 优化器进行优化。- 最终在测试集上的识别率达到了 98% 以上，证明了 LeNet-5 在手写数字识别任务中的有效性。

2. ResNet34 的实现与训练：- 我们使用 CIFAR-10 数据库实现了 ResNet34 的训练和测试。该数据库包含 10 类物体，每类 6000 张图片，总计 6 万张图片。- 在训练过程中，我们同样使用了交叉熵损失函数和 Adam 优化器，并对网络结构进行了适当的调整以提高分类性能。- 最终在测试集上的分类准确率达到了 90% 左右，表明 ResNet34 在物体分类任务中具有较好的表现。

5.2 心得体会

- 深度学习开发库的使用：- TensorFlow 和 PyTorch 等深度学习开发库提供了丰富的构建函数和训练接口，使得我们能够快速搭建和训练复杂的神经网络。- 通过这些库，我们不仅能够方便地实现各种网络结构，还能利用其强大的优化功能，提高了实验效率。

- 数据预处理的重要性：- 在实验过程中，我们发现数据预处理对模型的训练效果有着重要影响。适当的数据归一化和增强技术能够显著提高模型的泛化能力。- 对于 MNIST 数据集，我们进行了灰度归一化处理；对于 CIFAR-10 数据集，我们进行了数据增强处理，如随机裁剪和水平翻转。

- 模型调参与优化：- 在训练过程中，我们尝试了不同的超参数设置，如学习率、批量大小等，并观察其对模型性能的影响。- 通过不断调整和优化，我们最终找到了较为合适的参数组合，使得模型在测试集上的表现达到了预期效果。

- 实验中的挑战与解决方案：- 在实验初期，我们遇到了一些训练不稳定和过拟合的问题。通过增加正则化项和使用早停策略，我们有效地缓解了这些问题。- 此外，我们还尝试了不同的网络结构和激活函数，以进一步提高模型的性能。

总的来说，本次实验不仅加深了我们对卷积神经网络的理解，也提高了我们在深度学习开发库中的实践能力。通过不断的尝试和优化，我们成功地实现了对手写数字和物体分类任务的高效识别。