机器学习 实验二

【学号】22336259 【姓名】谢宇桐 【专业】计算机科学与技术

实验问题：

探索神经网络在图像分类任务上的应用。在给定数据集 CIFAR-10 的训练集上训练模型， 并在测试集上验证其性能。

采用的模型结构和训练方法：

本次实验使用PyTorch进行深度学习

```
# 导入库
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import numpy as np
from torch.utils.data import TensorDataset, DataLoader
import pickle

# 数据加载
def load_data(dir):
    X_train = []
    Y_train = []
    for i in range(1, 6):
        with open(dir + r'/data_batch_' + str(i), 'rb') as fo:
            dict = pickle.load(fo, encoding='bytes')
        X_train.append(dict[b'data'])
        Y_train += dict[b'labels']
    X_train = np.concatenate(X_train, axis=0)

    with open(dir + r'/test_batch', 'rb') as fo:
        dict = pickle.load(fo, encoding='bytes')
    X_test = dict[b'data']
    Y_test = dict[b'labels']

    return X_train, Y_train, X_test, Y_test

# 读取数据
X_train, Y_train, X_test, Y_test = load_data('./data')

# 数据预处理：将图像数据转换为适合神经网络处理的格式，并归一化到 [0, 1] 范围内。标签被转换为独热编码。
X_train = X_train.reshape(-1, 3, 32, 32).astype(np.float32) / 255.0
X_test = X_test.reshape(-1, 3, 32, 32).astype(np.float32) / 255.0
Y_train = np.eye(10)[Y_train]
Y_test = np.eye(10)[Y_test]
```

```python
# 转换为 PyTorch 张量
X_train_tensor = torch.tensor(X_train)
Y_train_tensor = torch.tensor(Y_train)
X_test_tensor = torch.tensor(X_test)
Y_test_tensor = torch.tensor(Y_test)

# 创建数据集和 DataLoader，使用 DataLoader 以批量方式加载数据。
train_dataset = TensorDataset(X_train_tensor, Y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, Y_test_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

```python
# 训练模型
def train_model(model, optimizer, criterion, num_epochs=10):
    for epoch in range(num_epochs):
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            if batch_idx % 100 == 0:
                print(f"Epoch {epoch}, Batch {batch_idx}, Loss: {loss.item()}")

# 测试函数
def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = model(data)
            _, predicted = torch.max(output, 1)  # 得到预测结果
            total += target.size(0)
            correct += (predicted == target.argmax(dim=1)).sum().item()  # 修正比
较逻辑

    accuracy = 100 * correct / total
    print(f"Accuracy of the network on the 10000 test images: {accuracy} %")
    return accuracy
```

实验内容：

1. **在给定的训练数据集上，分别训练一个线性分类器（Softmax 分类器），多层感知机（MLP）和卷积神经网络（CNN）**

   线性分类器（Softmax 分类器）：通过一个线性层和 softmax 层进行分类。我们的目标是通过一个线性模型，将输入特征映射到对应的类别标签。代码实现为一个简单的全连接神经网络，将输入图像展平后连接到一个输出层，输出层有 10 个神经元，对应 10 个类别。

```python
class SoftmaxClassifier(nn.Module):
    def __init__(self):
        super(SoftmaxClassifier, self).__init__()
        self.fc = nn.Linear(3 * 32 * 32, 10)

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32)
        x = self.fc(x)
        return x
```

```python
# 训练和评估 Softmax 分类器
model = SoftmaxClassifier()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
train_model(model, optimizer, criterion)
print("Accuracy of the trained model is:")
evaluate_model(model, train_loader)
print("Accuracy of the test model is:")
evaluate_model(model, test_loader)
```

MLP：多层感知机（MLP），包含多个全连接层，每层后跟一个 ReLU 激活函数。MLP 模型堆叠多层非线性变换来增强网络的表达能力，能捕获输入数据的复杂特征关系。

```python
# 修改后的MLP类
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dims, num_classes):
        super(MLP, self).__init__()
        layers = []
        in_dim = input_dim
        for hidden_dim in hidden_dims:
            layers.append(nn.Linear(in_dim, hidden_dim))
            layers.append(nn.BatchNorm1d(hidden_dim))  # 添加批量归一化
            layers.append(nn.ReLU())
            layers.append(nn.Dropout(0.5))  # 添加Dropout
            in_dim = hidden_dim
        layers.append(nn.Linear(in_dim, num_classes))
        self.network = nn.Sequential(*layers)

    def forward(self, x):
        x = x.view(-1, 3 * 32 * 32)  # 确保输入特征维度正确
        return self.network(x)
```

```python
# 训练和评估 MLP
input_size = 3 * 32 * 32  # 输入特征维度
hidden_sizes = [512, 512, 512, 512]  # 隐藏层尺寸
output_size = 10  # 输出类别数量

model = MLP(input_size, hidden_sizes, output_size)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)  # 尝试使用Adam优化器
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.1)  #
添加学习率调度器
```

```
num_epochs = 20

train_model(model, optimizer, criterion)
print("Accuracy of the trained model is:")
evaluate_model(model, train_loader)
print("Accuracy of the test model is:")
evaluate_model(model, test_loader)
```

CNN：卷积神经网络（CNN），包含两个卷积层和三个全连接层。卷积层用于提取图像特征，全连接层用于分类。

```
class ModifiedLeNet(nn.Module):
    def __init__(self, num_conv_layers=2):
        super(ModifiedLeNet, self).__init__()
        self.num_conv_layers = num_conv_layers
        self.layers = nn.ModuleList()
        in_channels = 3
        out_channels = 32

        for i in range(num_conv_layers):
            self.layers.append(nn.Conv2d(in_channels, out_channels,
kernel_size=5, padding=2))
            self.layers.append(nn.BatchNorm2d(out_channels))
            self.layers.append(nn.ReLU())
            self.layers.append(nn.MaxPool2d(kernel_size=2, stride=2))
            in_channels = out_channels
            out_channels *= 2

        self.fc1 = nn.Linear(256 * 4 * 4, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        x = x.view(-1, 256 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
# 训练和评估 CNN
model = ModifiedLeNet()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)  # 使用Adam优化器
num_epochs = 20  # 增加训练轮数

train_model(model, optimizer, criterion, num_epochs)
print("Accuracy of the trained model is:")
evaluate_model(model, train_loader)
print("Accuracy of the test model is:")
evaluate_model(model, test_loader)
```

结果在后续实验中会体现。

2. **在 MLP 实验中，研究使用不同网络层数和不同神经元数量对模型性能的影响**

修改`hidden_sizes`，迭代次数定为10次

层数：

2层512神经元数量：

```
Epoch 9, Batch 600, Loss: 1.3239529147394933
Epoch 9, Batch 700, Loss: 1.2186004121904261
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 57.756 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 51.76 %

进程已结束，退出代码为 0
```

4层512神经元数量：

```
Epoch 9, Batch 600, Loss: 1.4297556452511344
Epoch 9, Batch 700, Loss: 1.34220058612118
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 57.834 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 53.65 %

进程已结束，退出代码为 0
```

8层512神经元数量：

```
Epoch 9, Batch 600, Loss: 1.3855667714087758
Epoch 9, Batch 700, Loss: 1.3428281741216779
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 49.734 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 47.1 %

进程已结束，退出代码为 0
```

由上图可以看到，写出的MLP训练模型平均准确率在55%左右。而由上可得，随着 MLP 的层数增加，模型可以更充分捕获到特征信息，测试集上的效果更好。但若增加过多，则会出现过拟合现象，此时准确率反而下降。

神经元数量：

我们将层数定为4层：

神经元数量为256：

```
Epoch 9, Batch 600, Loss: 1.7277542774099857
Epoch 9, Batch 700, Loss: 1.5465347809367813
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 54.304 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 51.31 %

进程已结束，退出代码为 0
```

神经元数量为512：

```
Epoch 9, Batch 600, Loss: 1.4297556452511344
Epoch 9, Batch 700, Loss: 1.34220058612118
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 57.834 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 53.65 %

进程已结束，退出代码为 0
```

神经元数量为1024：

```
Epoch 9, Batch 600, Loss: 1.5533132173295598
Epoch 9, Batch 700, Loss: 1.481706439575646
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 59.972 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 53.53 %

进程已结束，退出代码为 0
```

由上我们可得，随着神经元数量的增加，MLP 的性能也会变得更好。

3. **在 CNN 实验中，以 LeNet 模型为基础，探索不同模型结构因素（如：卷积层数、滤波器数量、Pooling 的使用等）对模型性能的影响**

卷积层数：

```
class ModifiedLeNet(nn.Module):
    def __init__(self, num_conv_layers=3):
        super(ModifiedLeNet, self).__init__()
        self.num_conv_layers = num_conv_layers
        self.layers = nn.ModuleList()
        in_channels = 3
        out_channels = 32

        # 添加卷积层
        for i in range(num_conv_layers):
            self.layers.append(nn.Conv2d(in_channels, out_channels,
kernel_size=5, padding=2))
            self.layers.append(nn.BatchNorm2d(out_channels))
            self.layers.append(nn.ReLU())
            self.layers.append(nn.MaxPool2d(kernel_size=2, stride=2))
            in_channels = out_channels
            out_channels *= 2

        # 计算全连接层的输入尺寸
        def conv2d_size_out(size, kernel_size=5, stride=2, padding=2,
dilation=1):
            return (size + 2 * padding - dilation * (kernel_size - 1) - 1) //
stride + 1

        # 初始输入尺寸为32x32
        conv2d_size = 32
        for _ in range(num_conv_layers):
            conv2d_size = conv2d_size_out(conv2d_size)

        # 计算全连接层输入特征数
        self.fc1 = nn.Linear(out_channels * conv2d_size * conv2d_size, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        x = x.view(x.size(0), -1)  # 展平特征图
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

2层：

```
Epoch 9, Batch 600, Loss: 0.35902273998222256
Epoch 9, Batch 700, Loss: 0.6374367167615107
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 80.82 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 68.65 %

进程已结束，退出代码为 0
```

4层：

```
Epoch 9, Batch 600, Loss: 0.4835333594249096
Epoch 9, Batch 700, Loss: 0.29404408732631904
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 85.182 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 72.03 %

进程已结束，退出代码为 0
```

随着 CNN 层数的增加，CNN可以捕获到更加复杂的图像特征，准确率更高。

滤波器数量：

```
Epoch 9, Batch 700, Loss: 0.40834627816115976
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 89.6 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 74.35 %

进程已结束，退出代码为 0
```

```
Epoch 9, Batch 600, Loss: 0.3413094235479619
Epoch 9, Batch 700, Loss: 0.30535054037726717
Accuracy of the trained model is:
Accuracy of the network on the 10000 test images: 90.062 %
Accuracy of the test model is:
Accuracy of the network on the 10000 test images: 74.69 %

进程已结束，退出代码为 0
```

我们可以看到随着滤波器数量的增加，模型可以捕获到更加丰富的特征，从而提升了准确率。

4. **分别使用 SGD 算法、SGD Momentum 算法和 Adam 算法训练模型，观察并讨论他们对模型训练速度和性能的影响**

我们直接通过线性分类器进行比较：

```
# 训练和评估 Softmax 分类器
model = SoftmaxClassifier()
criterion = nn.CrossEntropyLoss()
# optimizer = optim.SGD(model.parameters(), lr=0.01) SGD
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9) # SGD Momentum
# optimizer = optim.Adam(model.parameters(), lr=0.001) Adam
train_model(model, optimizer, criterion)
evaluate_model(model, train_loader)
evaluate_model(model, test_loader)
```

SGD:

```
Epoch 9, Batch 400, Loss: 1.4315868287812918
Epoch 9, Batch 500, Loss: 1.7352082091711054
Epoch 9, Batch 600, Loss: 1.6802128977142274
Epoch 9, Batch 700, Loss: 1.8445314969867468
Accuracy of the network on the 10000 test images: 39.492 %
Accuracy of the network on the 10000 test images: 36.27 %

进程已结束，退出代码为 0
```

SGD Momentum:

```
Epoch 9, Batch 400, Loss: 1.7982580112293363
Epoch 9, Batch 500, Loss: 1.7394819082692266
Epoch 9, Batch 600, Loss: 1.6569924912182614
Epoch 9, Batch 700, Loss: 1.743945091497153
Accuracy of the network on the 10000 test images: 30.738 %
Accuracy of the network on the 10000 test images: 30.04 %

进程已结束，退出代码为 0
```

Adam:

```
Epoch 9, Batch 400, Loss: 2.4958901741520094
Epoch 9, Batch 500, Loss: 3.4382625045545865
Epoch 9, Batch 600, Loss: 2.841442233097041
Epoch 9, Batch 700, Loss: 3.020319999428466
Accuracy of the network on the 10000 test images: 27.374 %
Accuracy of the network on the 10000 test images: 25.86 %

进程已结束，退出代码为 0
```

通过对比来看，Adam准确率较低，而SGD最好，SGD Momentum较好，且这两种算法的训练过程较为稳定，

5. 比较并讨论线性分类器、MLP 和 CNN 模型在 CIFAR-10 图像分类任务上的性能区别

我们通过上述横向比较可得知：

在CIFAR-10图像分类任务中，线性分类器、MLP（多层感知器）和CNN（卷积神经网络）的性能存在显著差异：

线性分类器由于缺乏对图像特征的提取能力和空间信息的利用，性能不如MLP和CNN。

MLP通过增加隐藏层来学习数据的非线性特征，相比线性分类器，MLP能够捕捉更复杂的特征表示。MLP能够通过堆叠多个全连接层来提高模型的学习能力，但仍然不如CNN，因为MLP没有利用到图像的空间结构信息，受限于其对图像数据的平铺处理，没有考虑像素之间的空间关系。

CNN通过卷积操作能够学习到图像的局部模式，并通过池化层降低特征的空间维度，能够捕捉到图像中的空间层次结构和局部特征，能够达到更高的准确率。

综上，对于CIFAR-10图像分类任务，CNN因其能够捕捉图像的空间特征而表现最佳。MLP虽然比线性分类器表现更好，但由于缺乏对空间结构的利用，其性能仍然不如CNN。线性分类器在这类任务中通常表现最差，因为它无法有效利用图像数据的复杂结构。