上机实验三

实验设计

1. 任务描述

我们设计一个简单的卷积神经网络来完成图像分类任务，并对最后一层卷积层进行剪枝，观察剪枝对模型性能的影响。

2. 简单模型设计

我们设计一个CNN，包含以下层：

- 2个卷积层（Conv2D） + 激活函数（ReLU） + 最大池化层（MaxPool2D）
- 1个全连接层（Linear） + 输出层（Softmax）

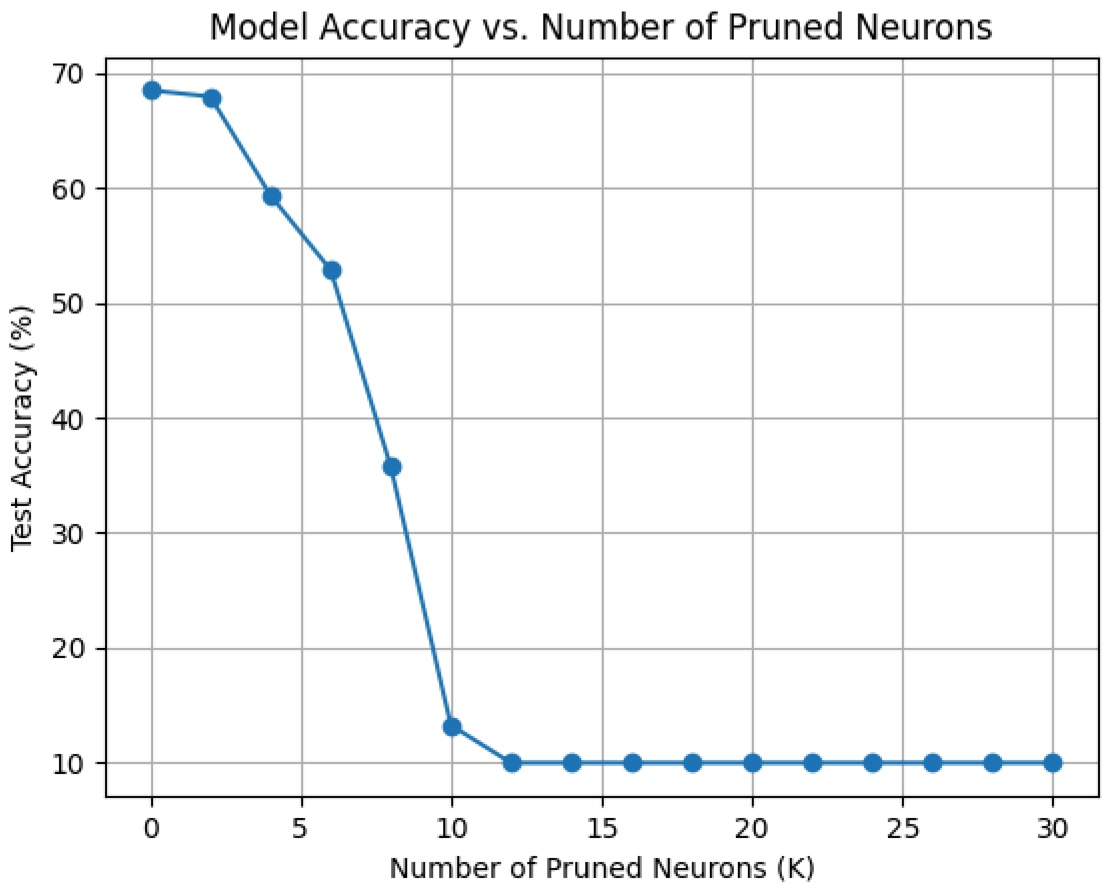
3. 剪枝方法

- 对最后一层卷积层的输出特征图进行剪枝。
- 根据神经元激活的平均值排序，剪掉激活值最低的 K 个神经元。
- 剪枝方法：将对应神经元的权重设为0。

代码实现

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torchvision import datasets, transforms
import matplotlib.pyplot as plt

batch_size = 64
learning_rate = 0.001
epochs = 10
num_classes = 10  # CIFAR-10数据集有10个类别

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
transform=transform)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
batch_size=batch_size, shuffle=False)

# CNN模型
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(32 * 8 * 8, 256)
        self.fc2 = nn.Linear(256, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

model = SimpleCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

for epoch in range(epochs):
    model.train()
    for i, (images, labels) in enumerate(train_loader):
        outputs = model(images)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

def test_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    return accuracy

initial_accuracy = test_model(model, test_loader)
print(f'Initial Test Accuracy: {initial_accuracy:.2f}%')

#剪枝
def prune_model(model, K):
    # 最后一层卷积层的权重
    last_conv_layer = model.conv2
    weights = last_conv_layer.weight.data

    activations = []
    with torch.no_grad():
        for images, _ in test_loader:
            output = model.conv2(model.pool(F.relu(model.conv1(images))))
            activations.append(output.mean(dim=(0, 2, 3)))  # 每个特征图的平均
激活值
    average_activations = torch.stack(activations).mean(dim=0)  # 所有测试样
本的平均激活值
    sorted_indices = torch.argsort(average_activations)
    for i in range(K):
        idx = sorted_indices[i]
        weights[idx] = 0  # 将前K个神经元的权重设为0
    return model

accuracies = []
K_values = range(0, 32, 2)
for K in K_values:
    pruned_model = prune_model(model, K)
    accuracy = test_model(pruned_model, test_loader)
    accuracies.append(accuracy)
    print(f'K = {K}, Test Accuracy: {accuracy:.2f}%')

plt.plot(K_values, accuracies, marker='o')
plt.xlabel('Number of Pruned Neurons (K)')
plt.ylabel('Test Accuracy (%)')
plt.title('Model Accuracy vs. Number of Pruned Neurons')
plt.grid()
plt.show()
```

实验结果



可以看到

- 随着剪枝比例 K 的增加，模型的准确率逐渐下降。
- 当 K 较小时，剪枝对模型性能影响较小，说明部分神经元对模型贡献较低。
- 当 K 较大时，模型性能显著下降，说明剪枝过多会导致模型丢失重要特征