CNN_CIFAR10_Classification

April 19, 2025

1 Homework2：CNN 实现 CIFAR-10 图像分类

xeCJK

SimSun

导出 PDF 时避免中文乱码

1.1 Step 1：导入必要的库文件

[1]:
```
# PyTorch 核心模块 (CUDA 12.4 版本)
import torch
import torch.nn as nn
import torch.optim as optim

# torchvision 模块：用于加载数据、数据增强
import torchvision
import torchvision.transforms as transforms

# 科学计算与可视化模块
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# 模型评估指标函数
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score,␣
↪classification_report

# 数据拆分工具
from torch.utils.data import random_split, DataLoader
```

Tips：导入用于模型训练、数据加载、可视化和评估所需的全部工具包。

1.2 Step 2：数据预处理、加载、划分验证集

[2]:
```
# 数据预处理：转换为 Tensor 并标准化（均值 0.5，方差 0.5）
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
```

```
])

# 加载原始训练数据集
full_trainset = torchvision.datasets.CIFAR10(root='./data', train=True,␣
 ↪download=True, transform=transform)

# 划分训练集和验证集（按 8:2 划分）
train_size = int(0.8 * len(full_trainset))  # 80% 作为训练集
val_size = len(full_trainset) - train_size  # 20% 作为验证集
trainset, valset = random_split(full_trainset, [train_size, val_size])

# 加载划分后的数据
trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
valloader = DataLoader(valset, batch_size=64, shuffle=False)

# 提取 CIFAR-10 的标签类别
classes = full_trainset.classes
```

Tips：将训练集按 8:2 划分为训练集与验证集，并加载测试集。

1.3 Step 3：构建卷积神经网络结构

[3]:
```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv = nn.Sequential(
            # 第一层：卷积 + 归一化 + 激活 + 池化
            nn.Conv2d(3, 32, 3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2),

            # 第二层：卷积 + 归一化 + 激活 + 池化
            nn.Conv2d(32, 64, 3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(2),

            # 第三层：卷积 + 归一化 + 激活 + 池化
            nn.Conv2d(64, 128, 3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
        #   全连接层设置
        self.fc = nn.Sequential(
            nn.Linear(128 * 4 * 4, 256), # 输出维度根据输入图片大小和卷积层调整
```

```
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, 10) # CIFAR-10 将输出标签分为 10 类
        )
    # 前向传播函数
    def forward(self, x):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        return self.fc(x)
```

Tips:网络包含三层卷积层 + 池化层,最后使用两层全连接层输出十类结果。网络中使用 BatchNorm 和 Dropout 提高鲁棒性。

1.4 Step 4: 训练设置初始化

[4]:
```
# CUDA 可用时进行 GPU 加速训练过程
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = SimpleCNN().to(device)

# 交叉熵损失函数用于分类问题
criterion = nn.CrossEntropyLoss()

# Adam 优化器 + 学习率调度器（每 10 轮衰减）
optimizer = optim.Adam(model.parameters(), lr=0.001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.5)

EPOCHS = 15  # 训练轮次为 15 轮
train_loss_list = []
val_loss_list = []
val_acc_list = []
```

Tips：构建交叉熵损失函数用于分类问题，采用 Adam 优化器和学习率调度器（每 10 轮衰减）进行训练迭代，同时使用 GPU 加速训练过程，一共训练 30 轮次。

1.5 Step 5: 训练神经网络

[5]:
```
val_loss_list = []
val_acc_list = []

for epoch in range(EPOCHS): # 20 轮次训练
    model.train()
    running_loss = 0.0

    for inputs, labels in trainloader:
        inputs, labels = inputs.to(device), labels.to(device)

        # 清空梯度
```

```
        optimizer.zero_grad()
        # 前向传播
        outputs = model(inputs)
        # 计算损失
        loss = criterion(outputs, labels)
        # 反向传播
        loss.backward()

        optimizer.step()
        running_loss += loss.item()

    scheduler.step()
    avg_train_loss = running_loss / len(trainloader)
    train_loss_list.append(avg_train_loss)

    # 验证过程
    model.eval()
    val_loss = 0.0
    correct = 0
    total = 0

    # 遍历测试集获取预测与真实标签
    with torch.no_grad():
        for inputs, labels in valloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    avg_val_loss = val_loss / len(valloader)
    val_accuracy = correct / total
    val_loss_list.append(avg_val_loss)
    val_acc_list.append(val_accuracy)

    # 输出每一轮次的训练损失、验证集损失和验证集准确率
    print(f"Epoch {epoch+1}/{EPOCHS}, Train Loss: {avg_train_loss:.4f}, Val␣
↪Loss: {avg_val_loss:.4f}, Val Acc: {val_accuracy:.4f}")
```

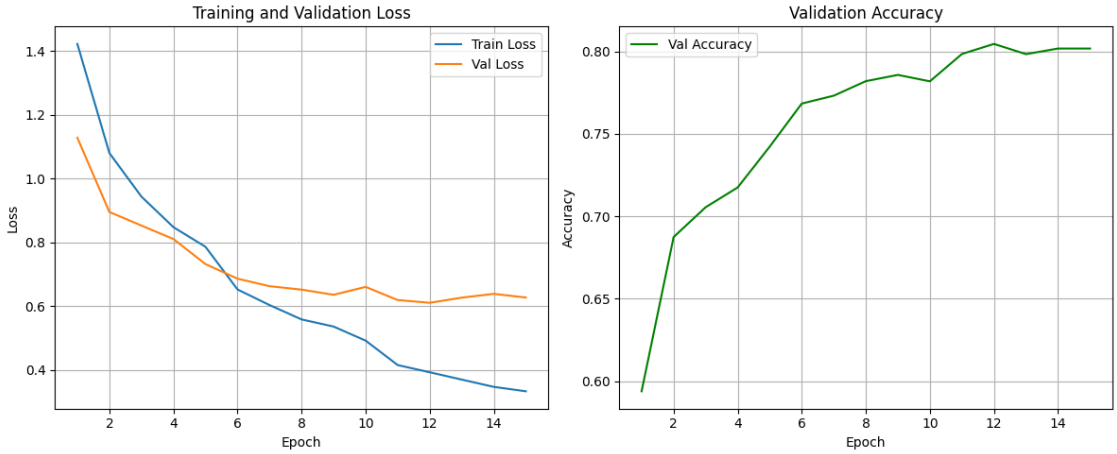
```
Epoch 1/15, Train Loss: 1.4219, Val Loss: 1.1277, Val Acc: 0.5938
Epoch 2/15, Train Loss: 1.0798, Val Loss: 0.8951, Val Acc: 0.6874
Epoch 3/15, Train Loss: 0.9433, Val Loss: 0.8524, Val Acc: 0.7055
Epoch 4/15, Train Loss: 0.8474, Val Loss: 0.8101, Val Acc: 0.7175
Epoch 5/15, Train Loss: 0.7858, Val Loss: 0.7316, Val Acc: 0.7423
Epoch 6/15, Train Loss: 0.6521, Val Loss: 0.6859, Val Acc: 0.7684
Epoch 7/15, Train Loss: 0.6031, Val Loss: 0.6624, Val Acc: 0.7732
```

```
Epoch 8/15, Train Loss: 0.5581, Val Loss: 0.6515, Val Acc: 0.7820
Epoch 9/15, Train Loss: 0.5358, Val Loss: 0.6354, Val Acc: 0.7858
Epoch 10/15, Train Loss: 0.4917, Val Loss: 0.6601, Val Acc: 0.7819
Epoch 11/15, Train Loss: 0.4151, Val Loss: 0.6189, Val Acc: 0.7985
Epoch 12/15, Train Loss: 0.3925, Val Loss: 0.6103, Val Acc: 0.8046
Epoch 13/15, Train Loss: 0.3695, Val Loss: 0.6264, Val Acc: 0.7984
Epoch 14/15, Train Loss: 0.3467, Val Loss: 0.6383, Val Acc: 0.8018
Epoch 15/15, Train Loss: 0.3328, Val Loss: 0.6268, Val Acc: 0.8018
```

Tips：反向传播算法进行学习率迭代，同时每轮训练后进行验证集评估，并保存损失和准确率结果用于可视化。

1.6 Step 6：结果可视化（损失曲线绘制）+ 训练模型保存

[6]:
```python
epochs = range(1, EPOCHS + 1)

plt.figure(figsize=(12,5))

# 绘制损失曲线（Loss 曲线）
plt.subplot(1, 2, 1)
plt.plot(epochs, train_loss_list, label='Train Loss')
plt.plot(epochs, val_loss_list, label='Val Loss')
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()
plt.grid(True)

# 绘制验证准确率曲线（Accuracy 曲线）
plt.subplot(1, 2, 2)
plt.plot(epochs, val_acc_list, label='Val Accuracy', color='green')
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.title("Validation Accuracy")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

torch.save(model.state_dict(), "cnn_model.pth")
```



Tips：通过绘制图线，便于直观地观察训练过程中是否存在过拟合或欠拟合的情况。保存模型 (cnn_model.pth) 以便后续使用或部署。

1.7 Step 7：模型评估与指标输出

[7]:
```
model.eval()
y_true = []
y_pred = []

# 加载测试集
testset = torchvision.datasets.CIFAR10(root='./data', train=False,␣
 ↪download=True, transform=transform)
testloader = DataLoader(testset, batch_size=64, shuffle=False)

with torch.no_grad():
    for inputs, labels in testloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        y_true += labels.cpu().tolist()
        y_pred += predicted.cpu().tolist()

# 输出总体准确率
acc = accuracy_score(y_true, y_pred)
print(f"\n Test Accuracy: {acc:.4f}")

# 输出每类的 precision
for i, c in enumerate(classes):
    p = precision_score(y_true, y_pred, labels=[i], average='macro')
    print(f"Class {c} Precision: {p:.4f}")
```

```
# 输出完整分类报告
print("\n Classification Report:")
print(classification_report(y_true, y_pred, target_names=classes))
```

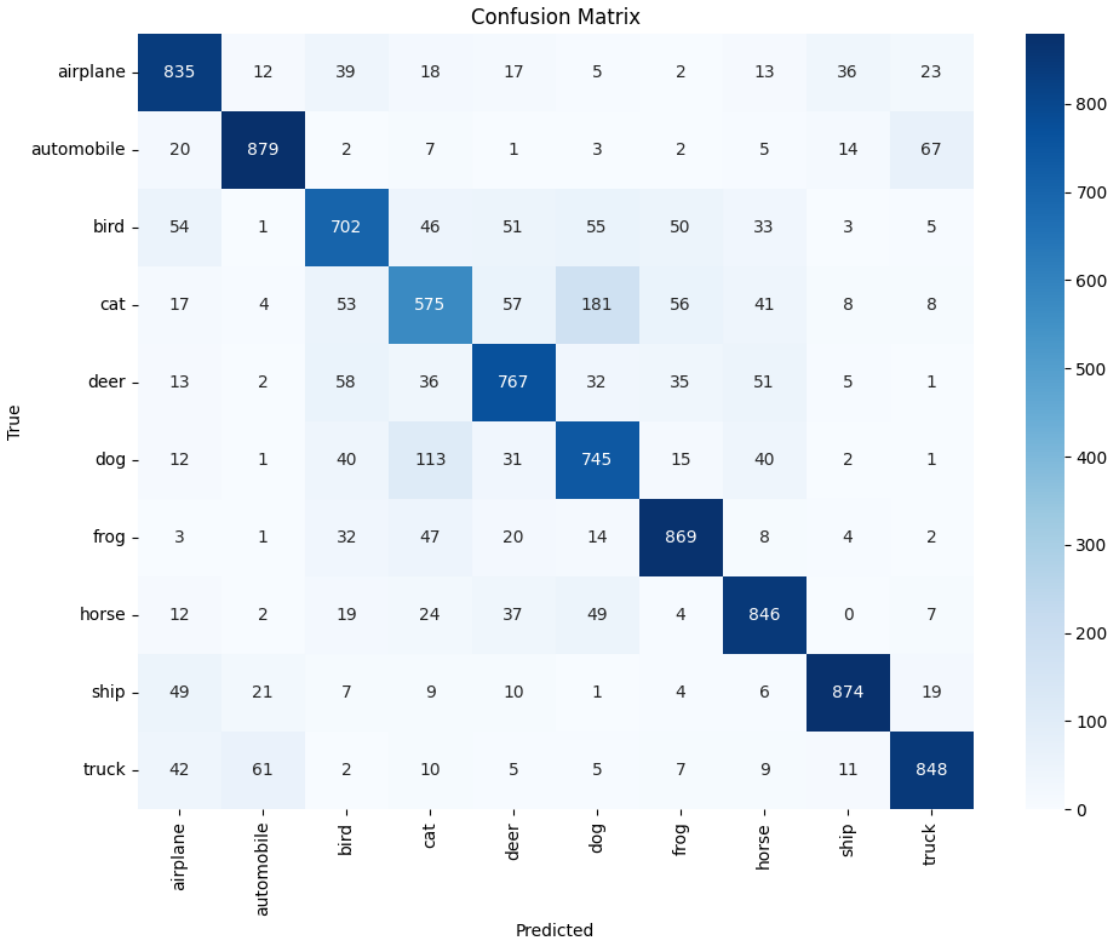
```
 Test Accuracy: 0.7940
Class airplane Precision: 0.7900
Class automobile Precision: 0.8933
Class bird Precision: 0.7358
Class cat Precision: 0.6497
Class deer Precision: 0.7701
Class dog Precision: 0.6835
Class frog Precision: 0.8324
Class horse Precision: 0.8042
Class ship Precision: 0.9133
Class truck Precision: 0.8644

 Classification Report:
              precision    recall  f1-score   support

    airplane       0.79      0.83      0.81      1000
  automobile       0.89      0.88      0.89      1000
        bird       0.74      0.70      0.72      1000
         cat       0.65      0.57      0.61      1000
        deer       0.77      0.77      0.77      1000
         dog       0.68      0.74      0.71      1000
        frog       0.83      0.87      0.85      1000
       horse       0.80      0.85      0.82      1000
        ship       0.91      0.87      0.89      1000
       truck       0.86      0.85      0.86      1000

    accuracy                           0.79     10000
   macro avg       0.79      0.79      0.79     10000
weighted avg       0.79      0.79      0.79     10000
```

Tips：评估分类性能，使用 sklearn 输出准确率、每类精度和整体报告。

1.8 Step 8：混淆矩阵可视化

[8]:
```
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10,8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=classes, yticklabels=classes,␣
 ↪cmap='Blues')
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
```

```
plt.tight_layout()
plt.show()
```



Tips：可视化混淆矩阵以展示 CNN 网络在哪些类别上分类混淆，表明该网络仍具有优化空间。