CVPR 第二次作业　图像分类实验

吴天阳　4124136039　人工智能学院 B2480

1　实验目的

1. 基于两层神经网络的图像分类器;
2. 学习使用 PyTorch 深度学习框架搭建图像分类器;
3. 学习使用常用 CNN 结构和图像增强技术.

2　实验原理

2.1　全连接网络

全连接网络用于图像分类的基本流程如下:

2.1.1　输入图像

给定一幅输入图像,假设大小为 $H \times W \times C$,其中:

- H 为图像高度(像素数);
- W 为图像宽度(像素数);
- C 为通道数(灰度图通道数为 1,RGB 图像通道数为 3).

将输入图像表示为一个张量 $x \in \mathbb{R}^{H\times W\times C}$.

2.1.2　特征展平

为了输入到全连接层,首先将图像展平成一个一维向量:

$$x_{\text{flat}} = \text{flatten}(x) \in \mathbb{R}^{HWC}.$$

此过程保留了图像的所有像素信息,但丢失了空间结构信息.

2.1.3　全连接层计算

全连接层通过一个权重矩阵 W 和一个偏置向量 $\boldsymbol{b}$ 对输入进行线性变换:

$$\boldsymbol{z} = Wx_{\text{flat}} + \boldsymbol{b},$$

其中:

- $W \in \mathbb{R}^{N\times(HWC)}$ 是权重矩阵,N 为神经元的数量.
- $b \in \mathbb{R}^N$ 是偏置向量.
- $z \in \mathbb{R}^N$ 是线性变换的结果.

2.1.4　激活函数

在线性变换之后,通过非线性激活函数(例如 ReLU)引入非线性特性:

$$\boldsymbol{a} = \sigma(\boldsymbol{z}),$$

其中 σ 是激活函数,常用的包括:

- ReLU: $\sigma(x) = \max(0, x)$
- Sigmoid: $\sigma(x) = \dfrac{1}{1+e^{-x}}$
- Mish: $\sigma(x) = x\tanh(\text{softplus}(x)) = x\tanh(\ln(1+e^x))$

2.1.5 输出层和分类

输出层通常是另一个全连接层，其输出的维度等于分类任务的类别数 C_{class}：

$$\boldsymbol{y}_{\text{pred}} = \text{softmax}(W_{\text{out}}\boldsymbol{a} + \boldsymbol{b}_{\text{out}}),$$

其中：

- $W_{\text{out}} \in \mathbb{R}^{C_{\text{class}}\times N}$ 为输出层的权重.
- $\boldsymbol{b}_{\text{out}} \in \mathbb{R}^{C_{\text{class}}}$ 为输出层的偏置.
- softmax 将输出变为概率分布：$\text{softmax}(z_i) = \dfrac{e^{z_i}}{\sum_j e^{z_j}}$.

2.1.6 损失函数

使用交叉熵损失函数（Cross-Entropy Loss）来衡量预测概率分布和真实标签的差异：

$$\mathcal{L} = -\sum_{i=1}^{C_{\text{class}}} y_i \log(\hat{y}_i),$$

其中：

- y_i 是真实标签的 one-hot 编码.
- $\hat{y}_i$ 是模型预测的概率分布.

通过梯度下降或其他优化方法更新网络参数，最小化损失函数.

2.1.7 分类结果

最终的分类结果为输出概率中最大值对应的类别：

$$\text{class} = \arg\max_i \boldsymbol{y}_{\text{pred}}.$$

2.2 卷积网络

一个典型的 CNN 模型包括以下几层：

2.2.1 卷积层

卷积层通过卷积核对输入数据进行操作，提取局部特征. 卷积运算公式如下：

$$z_{i,j}^k = \sum_{m=1}^{M}\sum_{n=1}^{N} x_{i+m-1,j+n-1} w_{m,n}^k + b^k,$$

其中：

- $x_{i,j}$ 是输入数据.
- $w_{m,n}^k$ 是第 k 个卷积核的权重.
- b^k 是偏置项.
- $z_{i,j}^k$ 是卷积结果.

2.2.2 池化层

池化层用于降维和减少计算量，常用的操作有最大池化和平均池化．例如，对于最大池化：

$$z_{i,j} = \max_{p,q} x_{i+p,j+q},$$

其中 p, q 是池化窗口的范围．

2.2.3 全连接层

全连接层将前面提取的特征映射到最终的输出空间．其计算公式为：

$$z = Wx + b,$$

其中 W 是权重矩阵，b 是偏置项．

3 实验步骤与结果分析

3.1 在 cifar10 上用 PyTorch 训练两层神经网络分类器

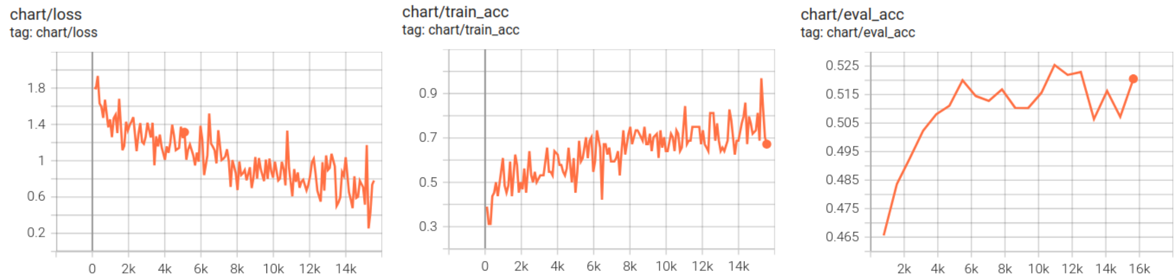
训练流程为，定义超参数、神经网络，读取数据集，划分数据集为训练集与验证集，实例化模型、优化器、损失函数，开始训练，在验证集上验证模型性能，保存模型，具体代码如下：

```python
import time
from pathlib import Path

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torch.utils.tensorboard.writer import SummaryWriter

# Tensorbaord 日志
path_log = Path(f"./logs/{time.strftime('%Y%m%d-%H%M%S')}")
writer = SummaryWriter(path_log)

# 超参数设置
batch_size = 64
learning_rate = 0.001
num_epochs = 20
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# 数据加载和预处理
transform = transforms.Compose([
  transforms.ToTensor(),         # 转换为 Tensor
  transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # 标准化到 [-1, 1]
])

train_dataset = datasets.CIFAR10(root='./data', train=True,
↪  transform=transform, download=True)
```

```python
test_dataset = datasets.CIFAR10(root='./data', train=False,
    transform=transform, download=True)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
    shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
    shuffle=False)

# 定义全连接神经网络
class FullyConnectedNN(nn.Module):
  def __init__(self, input_size, hidden_size, num_classes):
    super(FullyConnectedNN, self).__init__()
    self.fc1 = nn.Linear(input_size, hidden_size)  # 输入到隐藏层
    self.relu = nn.ReLU()              # 激活函数
    self.fc2 = nn.Linear(hidden_size, num_classes)  # 隐藏层到输出层

  def forward(self, x):
    x = x.view(x.size(0), -1)  # 展平
    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    return x

# 模型实例化
input_size = 32 * 32 * 3  # CIFAR-10 图像大小 (32x32x3)
hidden_size = 256     # 隐藏层神经元数
num_classes = 10      # CIFAR-10 分类数
model = FullyConnectedNN(input_size, hidden_size, num_classes).to(device)

# 定义损失函数和优化器
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
global_step = 0
best_eval_acc = 0

# 训练模型
for epoch in range(num_epochs):
  model.train()
  for batch_idx, (data, target) in enumerate(train_loader):
    data, target = data.to(device), target.to(device)
    # 前向传播
    outputs = model(data)
    loss = criterion(outputs, target)
    _, predicted = torch.max(outputs, 1)
    acc = (target == predicted).sum().item() / batch_size

    # 反向传播
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    global_step += 1
```

```python

    if (batch_idx + 1) % 100 == 0:
      writer.add_scalar("chart/loss", loss.item(), global_step)
      writer.add_scalar("chart/train_acc", acc, global_step)
      print(f"Epoch [{epoch + 1}/{num_epochs}], Step [{batch_idx + 1}/{len(train_loader)}], Loss: {loss.item():.4f}, Acc: {acc:.4f}")

  # 测试模型
  model.eval()
  correct = 0
  total = 0
  with torch.no_grad():
    for data, target in test_loader:
      data, target = data.to(device), target.to(device)
      outputs = model(data)
      _, predicted = torch.max(outputs.data, 1)
      total += target.size(0)
      correct += (predicted == target).sum().item()

  eval_acc = correct / total
  print(f"Test Accuracy: {100 * eval_acc:.2f}%")
  writer.add_scalar("chart/eval_acc", eval_acc, global_step)

  if eval_acc > best_eval_acc:
    best_eval_acc = eval_acc
    # 保存最优 eval 模型
    path_save_model = f"cifar10_fc_model_best_eval.pth"
    torch.save(model.state_dict(), path_log / path_save_model)
    print(f"Best eval model ({100*eval_acc:.2f}%) saved as {path_log / path_save_model}")


# 保存模型
path_save_model = f"cifar10_fc_model_{global_step}.pth"
torch.save(model.state_dict(), path_log / path_save_model)
print(f"Last model saved as {path_log / path_save_model}")
print(f"Best eval accuracy {100 * best_eval_acc:.2f}%")
```

TensorBoard 日志图片如下：



(a) 损失函数图像 (b) 训练中准确率图像 (c) 训练中准确率图像

图 1: 训练 20 个 epochs 的 TensorBoard 日志图像，在验证集上的最优准确率为第 11 个 epoch 时的 52.54%

3.2 在 cifar10 上用 PyTorch 训练卷积网络分类器

与全连接神经网络不同之处在于：

1. 使用了图像增强，包括随机裁剪，随机水平翻转，色彩抖动；
2. 三个 CNN 卷积块（2D 卷积，批归一化，Mish 激活函数），每个卷积块后经过一个最大池化将图像缩小一倍，最后展平，用全连接做输出头预测类别.

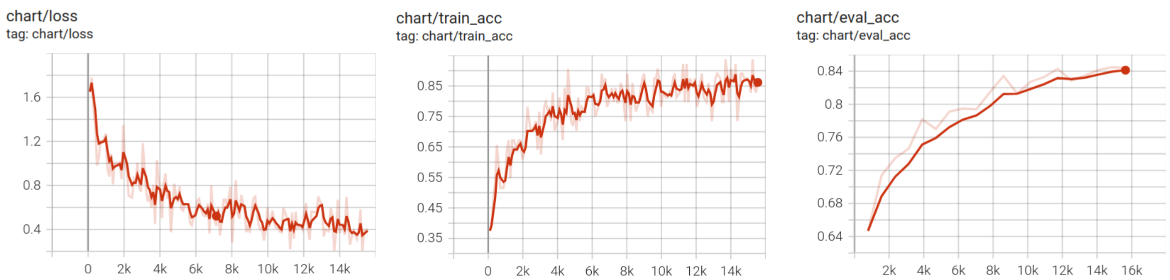
```
import time
from pathlib import Path

import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
from torch.utils.tensorboard.writer import SummaryWriter

# Tensorbaord 日志
path_log = Path(f"./logs/{time.strftime('%Y%m%d-%H%M%S')}")
writer = SummaryWriter(path_log)

# 检查是否有可用 GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# 超参数设置
batch_size = 64
learning_rate = 0.001
num_epochs = 20

# 数据增强和预处理
transform_train = transforms.Compose([
  transforms.RandomCrop(32, padding=4),       # 随机裁剪
  transforms.RandomHorizontalFlip(),       # 随机水平翻转
  transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2,
    hue=0.1),  # 色彩抖动
  transforms.ToTensor(),
```

```python
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # 标准化
])

transform_test = 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, transform=transform_train, download=True)
test_dataset = datasets.CIFAR10(root='./data', train=False, transform=transform_test, download=True)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

class CNN(nn.Module):
    def __init__(self, in_ch, out_ch, kernel, stride, padding):
        super().__init__()
        self.conv = nn.Conv2d(in_ch, out_ch, kernel_size=kernel, stride=stride, padding=padding)
        self.bn = nn.BatchNorm2d(out_ch)
        self.mish = nn.Mish()

    def forward(self, x):
        return self.mish(self.bn(self.conv(x)))

# 定义 CNN 模型
class Model(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        self.backbone = nn.Sequential(
            CNN(3, 64, 3, 1, 1),
            nn.MaxPool2d(kernel_size=2, stride=2),
            CNN(64, 128, 3, 1, 1),
            nn.MaxPool2d(kernel_size=2, stride=2),
            CNN(128, 256, 3, 1, 1),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )
        self.head = nn.Sequential(
            nn.Linear(256 * 4 * 4, 512),
            nn.Mish(),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        x = self.backbone(x)
        x = nn.Flatten()(x)
        x = self.head(x)
```

```python
    return x

# 初始化模型、损失函数和优化器
model = Model().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
global_step = 0
best_eval_acc = 0

# 训练模型
for epoch in range(num_epochs):
  model.train()
  for batch_idx, (data, target) in enumerate(train_loader):
    data, target = data.to(device), target.to(device)

    # 前向传播
    outputs = model(data)
    loss = criterion(outputs, target)
    _, predicted = torch.max(outputs, 1)
    acc = (target == predicted).sum().item() / batch_size

    # 反向传播
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    global_step += 1

    if (batch_idx + 1) % 100 == 0:
      writer.add_scalar("chart/loss", loss.item(), global_step)
      writer.add_scalar("chart/train_acc", acc, global_step)
      print(f"Epoch [{epoch + 1}/{num_epochs}], Step [{batch_idx + 1}/{len(train_loader)}], Loss: {loss.item():.4f}, Acc: {acc:.4f}")

  # 测试模型
  model.eval()
  correct = 0
  total = 0
  with torch.no_grad():
    for data, target in test_loader:
      data, target = data.to(device), target.to(device)
      outputs = model(data)
      _, predicted = torch.max(outputs.data, 1)
      total += target.size(0)
      correct += (predicted == target).sum().item()

  eval_acc = correct / total
  print(f"Test Accuracy: {100 * eval_acc:.2f}%")
  writer.add_scalar("chart/eval_acc", eval_acc, global_step)

  if eval_acc > best_eval_acc:
```

```
        best_eval_acc = eval_acc
        # 保存最优 eval 模型
        path_save_model = f"cifar10_fc_model_best_eval.pth"
        torch.save(model.state_dict(), path_log / path_save_model)
        print(f"Best eval model ({100*eval_acc:.2f}%) saved as {path_log / path_save_model}")

# 保存模型
path_save_model = f"cifar10_fc_model_{global_step}.pth"
torch.save(model.state_dict(), path_log / path_save_model)
print(f"Last model saved as {path_log / path_save_model}")
print(f"Best eval accuracy {100 * best_eval_acc:.2f}%")
```

TensorBoard 日志图片如下：



(a) 损失函数图像 (b) 训练中准确率图像 (c) 训练中准确率图像

图 2: 训练 20 个 epochs 的 TensorBoard 日志图像，在验证集上的最优准确率为第 19 个 epoch 时的 84.48%