实验三 基于神经网络的人脸朝向识别算法

0. 组队信息

姓名	学号	分工
孙成栋	200110810	数据处理，代码实现
陈佳涛	200110817	报告撰写，查阅资料

1. 实验目的

研究人脸图像朝向分类任务，评估不同模型在该任务上的性能，以及研究数据集划分和模型调优对实验结果的影响。这有助于进一步提高图像分类算法的性能，并有助于解决人脸识别和图像分类相关的实际问题。

2. 数据集处理

```
import os
import shutil

dir = r"./dataset"
for part in os.listdir(dir):
    sub_dir = os.path.join(dir, part)
    for file in os.listdir(sub_dir):
        sub_file_2 = os.path.join(sub_dir, file)
        for file_name in os.listdir(sub_file_2):
            file_name, file_extension = os.path.splitext(file_name)  # 分离文件名和
扩展名
            new_file_name = file_name + "_" + file + file_extension  # 添加标签到文
件名末尾
            file_path = os.path.join(sub_file_2, file_name + file_extension)
            new_file_path = os.path.join(sub_file_2, new_file_name)
            os.rename(file_path, new_file_path)

            import os


# 定义最大的文件夹路径
# main_folder = r"./dataset/test"
# main_folder = r"./dataset/train"
# main_folder = r"./dataset/valid"

# 列出最大文件夹下的所有小文件夹
subfolders = [os.path.join(main_folder, folder) for folder in
os.listdir(main_folder) if os.path.isdir(os.path.join(main_folder, folder))]

# 移动每个小文件夹中的文件到最大的文件夹
for folder in subfolders:
    for item in os.listdir(folder):
```

```
        item_path = os.path.join(folder, item)
        if os.path.isfile(item_path):
            shutil.move(item_path, main_folder)

# 删除小文件夹
for folder in subfolders:
    shutil.rmtree(folder)
```

将所有的图片全部重命名并且移至母文件夹中，便于后续操作。

3. 实验步骤

- `dataset.py`

```
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import torch
from torch.utils.data import Dataset
from torchvision import transforms
from PIL import Image


class FaceDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.image_files = os.listdir(data_dir)
        self.label_encoder = LabelEncoder()

        self.labels = [file.split("_")[-1] for file in self.image_files]
        self.labels = self.label_encoder.fit_transform(self.labels)

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        image_name = os.path.join(self.data_dir, self.image_files[idx])
        image = Image.open(image_name)
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label
```

`FaceDataset`类定义:

- `__init__`方法: 类的构造函数接受两参数，`data_dir`和`transform`。`data_dir`是包含人脸图像数据的文件夹路径，`transform`是一个可选的图像转换函数，默认为`None`。
- 在构造函数中，它执行以下操作:
  - 存储`data_dir`和`transform`为类成员变量。
  - 列出`data_dir`中的所有图像文件，并存储在`self.image_files`列表中。
  - 创建一个`LabelEncoder`对象，用于将字符串标签编码成整数。

    - 从图像文件名中提取标签，并使用`LabelEncoder`进行标签编码。这些编码后的标签存储在`self.labels`列表中。
- `__len__`方法: 返回数据集中图像的数量，即`self.image_files`的长度。
- `__getitem__`方法: 用于获取数据集中的图像和对应的标签。接受一个索引`idx`作为参数，并执行以下操作:
    - 构建图像文件的完整路径。
    - 打开图像文件，将其加载为PIL图像对象。
    - 获取对应的标签（已经编码的标签）。
    - 如果指定了`transform`，则应用图像转换操作（例如，缩放、裁剪、标准化等）。
    - 返回处理后的图像和标签作为元组`(image, label)`。

这个`FaceDataset`类可以用于创建PyTorch的数据加载器，以便将图像数据提供给深度学习模型进行训练。你可以使用`torch.utils.data.DataLoader`来加载这个自定义数据集，同时也可以自定义图像预处理操作，以满足特定的深度学习模型需求。

- `model.py`

```
import torch.nn as nn

# picture size (360 * 480)

class CNN(nn.Module):
    def __init__(self, num_classes, dropout=0.8):
        super(CNN, self).__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(16, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2, 2),
        )
        self.fc = nn.Sequential(
            nn.Linear(64 * 45 * 60, 512),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        x = self.cnn(x)
        x = x.view(x.size(0), -1)
        x = self.fc(x)
        return x
```

- `train_eval.py`

```
import torch
```

```python
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import accuracy_score
import numpy as np


# 数据预处理和模型训练的代码
import torch
import torch.optim as optim
import torch.nn as nn
import numpy as np


def train(
    model, train_loader, val_loader, num_epochs=100, patience=10,
learning_rate=0.001
):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    best_val_loss = float("inf")
    best_val_accuracy = 0.0
    best_model_state_dict = model.state_dict()
    patience_count = 0

    for epoch in range(num_epochs):
        model.train()
        for batch_idx, (images, labels) in enumerate(train_loader):
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            # 打印每个batch的训练信息
            if (batch_idx + 1) % 10 == 0:
                print(
                    f"Epoch [{epoch+1}/{num_epochs}] - Batch
[{batch_idx+1}/{len(train_loader)}] - Training Loss: {loss.item():.4f}"
                )

        model.eval()
        val_accuracy = []
        val_labels = []
        val_losses = []

        with torch.no_grad():
            for images, labels in val_loader:
                outputs = model(images)
                _, predicted = torch.max(outputs, 1)
                val_accuracy.extend((predicted == labels).cpu().numpy())
                val_labels.extend(labels.cpu().numpy())
                val_loss = criterion(outputs, labels)
                val_losses.append(val_loss.item())

            mean_val_accuracy = np.mean(val_accuracy)
            mean_val_loss = np.mean(val_losses)
```

```
            print(
                f"Epoch [{epoch+1}/{num_epochs}] - Validation Accuracy:
{mean_val_accuracy:.4f} - Validation Loss: {mean_val_loss:.4f}"
            )

            if mean_val_loss < best_val_loss:
                best_val_loss = mean_val_loss
                best_val_accuracy = mean_val_accuracy
                best_model_state_dict = model.state_dict()
                patience_count = 0
                # 保存模型到磁盘
                torch.save(best_model_state_dict, "best_model.pth")
            else:
                patience_count += 1

            if patience_count >= patience:
                print(f"Early stopping after {patience} epochs of no
improvement.")
                # 打印最好的模型的准确率和损失
                print(
                    f"Best Validation Accuracy: {best_val_accuracy:.4f} - Best
Validation Loss: {best_val_loss:.4f}"
                )
                break

    model.load_state_dict(best_model_state_dict)
    return model


# 模型评估的代码
def eval(model, val_loader):
    # 模型评估
    model.eval()
    val_accuracy = []
    val_labels = []
    for images, labels in val_loader:
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        val_accuracy.extend((predicted == labels).cpu().numpy())
        val_labels.extend(labels.cpu().numpy())

    # accuracy = accuracy_score(val_labels, val_accuracy)
    accuracy = np.mean(val_accuracy)
    print(f"Test Accuracy: {accuracy:.4f}")
```

1. `train` 函数:
    - 参数:
        - `model`: 要训练的PyTorch模型。
        - `train_loader`: 包含训练数据的数据加载器。
        - `val_loader`: 包含验证数据的数据加载器。
        - `num_epochs`: 训练的总轮数（默认为100）。
        - `patience`: 提前停止的耐心（默认为10），即如果验证损失在连续的`patience`轮中没有改善，就提前停止训练。
        - `learning_rate`: 学习率（默认为0.001）。
    - 功能:

        - 创建交叉熵损失（`nn.CrossEntropyLoss`）和优化器（`optim.Adam`）。
        - 初始化一些变量，如最佳验证损失（`best_val_loss`）、最佳验证准确率（`best_val_accuracy`）、最佳模型状态字典（`best_model_state_dict`）和耐心计数器（`patience_count`）。
        - 迭代训练数据集，执行以下操作：
            - 将模型设置为训练模式。
            - 对每个批次进行前向传播、计算损失、反向传播和权重更新。
            - 打印每个批次的训练损失。
        - 将模型设置为评估模式，并在验证数据集上进行评估：
            - 计算验证集上的准确率和损失。
            - 检查验证损失是否优于先前最佳的验证损失，如果是则更新最佳验证损失、最佳验证准确率和最佳模型状态字典，重置耐心计数器，并保存最佳模型到磁盘。
            - 如果验证损失连续`patience`次没有改善，就提前停止训练。
        - 返回训练后的模型。
2. `eval`函数：
    - 参数：
        - `model`：已经训练好的PyTorch模型。
        - `val_loader`：包含验证数据的数据加载器。
    - 功能：
        - 设置模型为评估模式。
        - 在验证数据集上执行以下操作：
            - 计算每个样本的预测。
            - 计算每个样本的准确性（预测是否与真实标签相匹配）。
            - 计算并打印验证集的平均准确率。
        - 返回验证准确率。

- `main.py`

```
from model import CNN
from train_eval import train, eval
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
from dataset import FaceDataset
import os
import torch
import random
import numpy as np


if __name__ == "__main__":
    # 获取当前脚本所在的绝对路径
    current_script_path = os.path.abspath(__file__)
    # 获取当前脚本所在的目录
    current_script_directory = os.path.dirname(current_script_path)
    # 设置当前工作目录为脚本所在的目录
    os.chdir(current_script_directory)

    # 设置随机数种子
    seed = 42
```

```
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed) if torch.cuda.is_available() else None
    random.seed(seed)
    np.random.seed(seed)

    test_dir = r"./dataset/test"
    train_dir = r"./dataset/train"
    val_dir = r"./dataset/valid"
    batch_size = 32

    transform = transforms.Compose([transforms.ToTensor()])

    # 创建数据集和数据加载器
    train_dataset = FaceDataset(train_dir, transform=transform)
    train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
    val_dataset = FaceDataset(val_dir, transform=transform)
    val_loader = DataLoader(val_dataset, batch_size=batch_size)
    test_dataset = FaceDataset(test_dir, transform=transform)
    test_loader = DataLoader(test_dataset, batch_size=batch_size)

    model = CNN(num_classes=3)
    model = train(model, train_loader, val_loader, num_epochs=10, patience=3)
    eval(model, test_loader)
```

1. 导入必要的库和模块：
    - `model.CNN`：从 `model` 模块导入一个名为 `CNN` 的深度学习模型，这个模型是用于图像分类任务的卷积神经网络（CNN）。
    - `train_eval.train` 和 `train_eval.eval`：从 `train_eval` 模块导入 `train` 和 `eval` 函数，用于模型的训练和评估。
    - `torchvision.transforms`：包含图像转换函数，用于对图像进行预处理。
    - `torch.utils.data`：包含数据加载器和数据集类，用于加载和批处理数据。
    - `dataset.FaceDataset`：从 `dataset` 模块导入 `FaceDataset` 数据集类，用于加载人脸图像数据。
    - 其他必要的 Python 库。
2. 设置随机数种子：在模型训练中设置随机数种子，以确保训练的可重复性。
3. 定义数据目录和超参数：
    - `test_dir`、`train_dir` 和 `val_dir`：分别是测试集、训练集和验证集的数据目录路径。
    - `batch_size`：批处理大小，用于定义每个训练和评估批次中包含的样本数量。
    - `transform`：图像转换操作，用于在加载图像时进行预处理。在这里，只是将图像转换为张量格式。
4. 创建数据集和数据加载器：
    - 使用 `FaceDataset` 数据集类创建训练、验证和测试数据集。
    - 使用 `DataLoader` 创建相应的数据加载器，用于批处理数据。
5. 创建模型：
    - 创建一个名为 `model` 的 CNN 模型，该模型用于图像分类任务。模型结构在 `model.CNN` 中定义。
6. 训练模型：
    - 使用 `train` 函数训练模型，传入训练数据加载器 `train_loader` 和验证数据加载器 `val_loader`，并指定训练的总轮数（`num_epochs`）和提前停止的耐心（`patience`）。
    - 训练过程将打印训练损失、验证准确率和验证损失，并在提前停止时保存最佳模型。

7. 模型评估：
    - 使用 `eval` 函数评估训练完成的模型，传入测试数据加载器 `test_loader`，计算并打印测试准确率。

4. 实验结果

```
Epoch [1/10] - Batch [10/57] - Training Loss: 1.0460
Epoch [1/10] - Batch [20/57] - Training Loss: 0.6098
Epoch [1/10] - Batch [30/57] - Training Loss: 0.2091
Epoch [1/10] - Batch [40/57] - Training Loss: 0.1973
Epoch [1/10] - Batch [50/57] - Training Loss: 0.2880
Epoch [1/10] - Validation Accuracy: 0.9433 - Validation Loss: 0.1536
Epoch [2/10] - Batch [10/57] - Training Loss: 0.0810
Epoch [2/10] - Batch [20/57] - Training Loss: 0.1706
Epoch [2/10] - Batch [30/57] - Training Loss: 0.1809
Epoch [2/10] - Batch [40/57] - Training Loss: 0.1863
Epoch [2/10] - Batch [50/57] - Training Loss: 0.3104
Epoch [2/10] - Validation Accuracy: 0.9550 - Validation Loss: 0.1273
Epoch [3/10] - Batch [10/57] - Training Loss: 0.1116
Epoch [3/10] - Batch [20/57] - Training Loss: 0.0035
Epoch [3/10] - Batch [30/57] - Training Loss: 0.0133
Epoch [3/10] - Batch [40/57] - Training Loss: 0.2143
Epoch [3/10] - Batch [50/57] - Training Loss: 0.0428
Epoch [3/10] - Validation Accuracy: 0.9800 - Validation Loss: 0.0605
Epoch [4/10] - Batch [10/57] - Training Loss: 0.0263
Epoch [4/10] - Batch [20/57] - Training Loss: 0.0287
Epoch [4/10] - Batch [30/57] - Training Loss: 0.0135
Epoch [4/10] - Batch [40/57] - Training Loss: 0.0951
Epoch [4/10] - Batch [50/57] - Training Loss: 0.1054
Epoch [4/10] - Validation Accuracy: 0.9717 - Validation Loss: 0.0836
Epoch [5/10] - Batch [10/57] - Training Loss: 0.0056
Epoch [5/10] - Batch [20/57] - Training Loss: 0.0618
Epoch [5/10] - Batch [30/57] - Training Loss: 0.0086
Epoch [5/10] - Batch [40/57] - Training Loss: 0.0030
Epoch [5/10] - Batch [50/57] - Training Loss: 0.0100
Epoch [5/10] - Validation Accuracy: 0.9717 - Validation Loss: 0.0928
Epoch [6/10] - Batch [10/57] - Training Loss: 0.0009
Epoch [6/10] - Batch [20/57] - Training Loss: 0.0057
Epoch [6/10] - Batch [30/57] - Training Loss: 0.0034
Epoch [6/10] - Batch [40/57] - Training Loss: 0.0416
Epoch [6/10] - Batch [50/57] - Training Loss: 0.0024
Epoch [6/10] - Validation Accuracy: 0.9650 - Validation Loss: 0.0972
Early stopping after 3 epochs of no improvement.
Best Validation Accuracy: 0.9800 - Best Validation Loss: 0.0605
Test Accuracy: 0.9900
```