이미지 분류3

전진희

1. Train/val data split

```
import os
import random
import shutil

# 원본 데이터 경로
label_folder_path = "./pneumonia_dataset"

# 새로운 데이터 경로
dataset_folder_path = "./xray_dataset"

# train과 val 폴더 경로
train_folder_path = os.path.join(dataset_folder_path, "train")
val_folder_path = os.path.join(dataset_folder_path, "val")

# train과 val 폴더 생성
os.makedirs(train_folder_path, exist_ok=True)
os.makedirs(val_folder_path, exist_ok=True)

# 라벨 폴더
org_folders = os.listdir(label_folder_path)

for org_folder in org_folders:
    org_folder_full_path = os.path.join(label_folder_path, org_folder)
    images = os.listdir(org_folder_full_path)
    random.shuffle(images)  # 이미지를 무작위로 섞음

    # 라벨 폴더 생성
    train_label_folder_path = os.path.join(train_folder_path, org_folder)
    val_label_folder_path = os.path.join(val_folder_path, org_folder)
    os.makedirs(train_label_folder_path, exist_ok=True)
    os.makedirs(val_label_folder_path, exist_ok=True)

    # 이미지를 train 폴더로 이동
    split_index = int(len(images) * 0.9)
    for image in images[:split_index]:
        src_path = os.path.join(org_folder_full_path, image)  # 원본 이미지 경로
        dst_path = os.path.join(train_label_folder_path, image)
# ./dataset/train/라벨_폴더/이미지.jpg
        shutil.copyfile(src_path, dst_path)

    # 이미지를 val 폴더로 이동
    for image in images[split_index:]:
        src_path = os.path.join(org_folder_full_path, image)  # 원본 이미지 경로
        dst_path = os.path.join(val_label_folder_path, image)
# ./dataset/val/라벨_폴더/이미지.jpg
```

```
        shutil.copyfile(src_path, dst_path)
```

2. CustomDataSet 함수 정의

```
import os
import cv2
import glob
from torch.utils.data import Dataset

class MyPneumoniaDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        # 데이터 디렉토리 경로 설정
        self.data_dir = glob.glob(os.path.join(data_dir, "*", "*.jpeg"))
        self.transform = transform
        self.label_dict = self.create_label_dict()

    def create_label_dict(self):
        # 라벨 딕셔너리 생성
        label_dict = {}
        for filepath in self.data_dir:
            label = os.path.basename(os.path.dirname(filepath))
            if label not in label_dict:
                label_dict[label] = len(label_dict)
        return label_dict

    def __getitem__(self, item):
        # 데이터셋에서 아이템 가져오기
        image_filepath = self.data_dir[item]
        img = cv2.imread(image_filepath)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        label = os.path.basename(os.path.dirname(image_filepath))
        label_idx = self.label_dict[label]

        if self.transform is not None:
            # 이미지 변환 적용
            img = self.transform(image=img)['image']

        return img, label_idx

    def __len__(self):
        # 데이터셋의 총 길이 반환
        return len(self.data_dir)
```

3-1. train 코드

efficientnet_b0모델을 이용하여 모델을 학습했습니다.

```
import argparse
import os
import torch.nn as nn
import torch
import torchvision
import albumentations as A
import pandas as pd
import matplotlib.pyplot as plt
from albumentations.pytorch import ToTensorV2
from torchvision.models.efficientnet import efficientnet_b0
from torch.utils.data import DataLoader
from torch.optim import AdamW
from torch.nn import CrossEntropyLoss
from tqdm import tqdm

from ex02_0718_01_customdataset import MyPneumoniaDataset

class Classifier_XRay:
    def __init__(self):
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        self.model = None
        self.train_losses = []
        self.valid_losses = []
        self.train_accs = []
        self.valid_accs = []

    def train(self, train_loader, val_loader, epochs, optimizer, criterion, start_epoch=0):
        best_val_acc = 0.0
        print("Training....")

        for epoch in range(start_epoch, epochs):
            train_loss = 0.0
            val_loss = 0.0
            train_acc = 0.0
            val_acc = 0.0

            self.model.train()
            train_loader_iter = tqdm(train_loader, desc=(f"Epoch : {epoch + 1}/{epochs}"),
leave=False)

            for index, (data, target) in enumerate(train_loader_iter):
                data, target = data.float().to(self.device), target.to(self.device)

                optimizer.zero_grad()
                outputs = self.model(data)
                loss = criterion(outputs, target)
                loss.backward()
                optimizer.step()

                train_loss += loss.item()

                _, pred = torch.max(outputs, 1)
                train_acc += (pred == target).sum().item()
```

```python
                train_loader_iter.set_postfix({"Loss": loss.item()})

            train_loss /= len(train_loader)
            train_acc = train_acc / len(train_loader.dataset)

            # eval()
            self.model.eval()
            with torch.no_grad():
                for data, target in val_loader:
                    data, target = data.float().to(self.device), target.to(self.device)
                    output = self.model(data)
                    pred = output.argmax(dim=1, keepdim=True)
                    val_acc += pred.eq(target.view_as(pred)).sum().item()
                    val_loss += criterion(output, target).item()

            val_loss /= len(val_loader)
            val_acc = val_acc / len(val_loader.dataset)

            self.train_losses.append(train_loss)
            self.train_accs.append(train_acc)
            self.valid_losses.append(val_loss)
            self.valid_accs.append(val_acc)

            print(f"Epoch [{epoch + 1}/{epochs}], Train loss: {train_loss:.4f}, "
                  f"Val loss: {val_loss:.4f}, Train ACC: {train_acc:.4f}, Val ACC: {val_acc:.4f}")

            if val_acc > best_val_acc:
                torch.save(self.model.state_dict(), "./ex02_0718_efficientnet_b0_best.pt")
                best_val_acc = val_acc

            # save the model state and optimizer state after each epoch
            torch.save({
                'epoch': epoch + 1,
                'model_state_dict': self.model.state_dict(),
                'optimizer_state_dict': optimizer.state_dict(),
                'train_losses': self.train_losses,
                'train_accs': self.train_accs,
                'val_losses': self.valid_losses,
                'val_accs': self.valid_accs,
            }, "./weight/0718/ex02_0718_efficientnet_b0_checkpoint.pt")

        torch.save(self.model.state_dict(), "./ex02_0718_efficientnet_b0_last.pt")

        self.save_result_to_csv()
        self.plot_loss()
        self.plot_accuracy()

    def save_result_to_csv(self):
        df = pd.DataFrame({
            'Train Loss': self.train_losses,
            'Train ACC': self.train_accs,
            'Validation Loss': self.valid_losses,
            'Validation ACC': self.valid_accs
        })
        df.to_csv('./train_val_result_ex02.csv', index=False)

    def plot_loss(self):
        plt.figure()
        plt.plot(self.train_losses, label="Train loss")
```

```
        plt.plot(self.valid_losses, label="val loss")
        plt.xlabel("Epoch")
        plt.ylabel('Loss')
        plt.legend()
        plt.savefig("./ex02_loss_plot.jpg")

    def plot_accuracy(self):
        plt.figure()
        plt.plot(self.train_accs, label="Train Accuracy")
        plt.plot(self.valid_accs, label="Valid Accuracy")
        plt.xlabel("Epoch")
        plt.ylabel('Loss')
        plt.legend()
        plt.savefig("./ex02_accuracy_plot.jpg")

    def run(self, args):
        self.model = efficientnet_b0(pretrained=True)
        self.model.classifier[0] = nn.Dropout(p=0.5, inplace=True)
        self.model.classifier[1] = nn.Linear(1280, out_features=3)
        self.model.to(self.device)

        train_transforms = A.Compose([
            A.Resize(width=224, height=224),
            A.HorizontalFlip(),
            A.VerticalFlip(),
            A.Normalize(),
            ToTensorV2()
        ])

        val_transforms = A.Compose([
            A.Resize(width=224, height=224),
            A.Normalize(),
            ToTensorV2()
        ])

        # dataset and dataloader
        train_dataset = MyPneumoniaDataset(args.train_dir, transform=train_transforms)
        val_dataset = MyPneumoniaDataset(args.val_dir, transform=val_transforms)

        train_loader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True)
        val_loader = DataLoader(val_dataset, batch_size=args.batch_size, shuffle=True)

        epochs = args.epochs
        criterion = CrossEntropyLoss().to(self.device)
        optimizer = AdamW(self.model.parameters(), lr=args.learning_rate,
weight_decay=args.weight_decay)
        start_epoch = 0

        if args.resume_training:
            checkpoint = torch.load(args.checkpoint_path)
            self.model.load_state_dict(checkpoint['model_state_dict'])
            optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
            self.train_losses = checkpoint['train_losses']
            self.train_accs = checkpoint['train_accs']
            self.valid_losses = checkpoint['val_losses']
            self.valid_accs = checkpoint['val_accs']
            start_epoch = checkpoint['epoch']

        self.train(train_loader, val_loader, epochs, optimizer, criterion, start_epoch=start_epoch)
```

```python
if __name__ == "__main__":
    '''
    파서는 사용자가 입력한 값들을 쉽게 읽고 사용하기 위한 도구정도로 이해하자
    '''
    parser = argparse.ArgumentParser()
    parser.add_argument("--train_dir", type=str, default="./xray_dataset/train/",
                        help='directory path to the training dataset')
    parser.add_argument("--val_dir", type=str, default="./xray_dataset/val/",
                        help="directory path to the validation dataset")
    parser.add_argument("--epochs", type=int, default=20,
                        help="number of epochs for training")
    parser.add_argument("--batch_size", type=int, default=32,
                        help='batch size for training and validation')
    parser.add_argument("--learning_rate", type=float, default=0.001,
                        help="learning rate for optimizer")
    parser.add_argument("--weight_decay", type=float, default=1e-4,
                        help='weight decay for optimizer')
    parser.add_argument("--resume_training", action='store_true',
                        help='resume training from the last checkpoint')
    parser.add_argument("--checkpoint_path", type=str,
                        default="./weight/0718/ex02_0718_efficientnet_b0_checkpoint.pt",
                        help="path to the checkpoint file")
    parser.add_argument("--checkpoint_folder_path", type=str,
                        default="./weight/0718")
    args = parser.parse_args()

    weight_folder_path = args.checkpoint_folder_path
    os.makedirs(weight_folder_path, exist_ok=True)

    classifier = Classifier_XRay()
    classifier.run(args)
```

ex02_0718_02_train × ex_01_0718_train ×

```
Epoch [11/20], Train loss: 0.2979, Val loss: 0.2265, Train ACC: 0.8780, Val ACC: 0.9092
Epoch [12/20], Train loss: 0.2595, Val loss: 0.2042, Train ACC: 0.8942, Val ACC: 0.9162
Epoch [13/20], Train loss: 0.2341, Val loss: 0.1650, Train ACC: 0.9010, Val ACC: 0.9331
Epoch [14/20], Train loss: 0.2081, Val loss: 0.1866, Train ACC: 0.9143, Val ACC: 0.9331
Epoch [15/20], Train loss: 0.2086, Val loss: 0.1952, Train ACC: 0.9134, Val ACC: 0.9242
Epoch [16/20], Train loss: 0.1902, Val loss: 0.1578, Train ACC: 0.9244, Val ACC: 0.9251
Epoch [17/20], Train loss: 0.1548, Val loss: 0.1062, Train ACC: 0.9404, Val ACC: 0.9621
Epoch [18/20], Train loss: 0.1627, Val loss: 0.1686, Train ACC: 0.9329, Val ACC: 0.9331
Epoch [19/20], Train loss: 0.1579, Val loss: 0.1305, Train ACC: 0.9354, Val ACC: 0.9521
Epoch [20/20], Train loss: 0.1429, Val loss: 0.1074, Train ACC: 0.9472, Val ACC: 0.9601
```

3-2. train 코드

ImageNet 데이터셋에서 이미지 분류를 위한 모델을 학습해봤습니다.

https://github.com/pytorch/examples/blob/main/imagenet/main.py 에서 코드 가져와서 Customdataset import 해주고 필요한 부분만 수정하고(모델, epochs, batch_size 등등)나서, 터미널에서 아래 명령어로 실행

python (파일명).py ./(daraset이름)/ --pretrained

```
Administrator: Anaconda Prompt - python .₩ex02_main_with_teacher.py .₩xray_dataset₩ --pretrained
poch: [0][31/42]       Time  2.657 ( 3.179)    Data  0.000 ( 0.655)    Loss 6.9085e-01 (1.4262e+00)    Acc@1  70.16 ( 62.20)   Acc@5 100.00 ( 9
.25)
poch: [0][41/42]       Time  2.659 ( 3.055)    Data  0.000 ( 0.539)    Loss 7.5209e-01 (1.2377e+00)    Acc@1  70.97 ( 64.50)   Acc@5 100.00 ( 9
.14)
est: [1/9]     Time 17.344 (17.344)    Loss 4.6082e-01 (4.6082e-01)    Acc@1  83.06 ( 83.06)   Acc@5 100.00 (100.00)
*   Acc@1 68.263 Acc@5 100.000
poch: [1][ 1/42]       Time 15.636 (15.636)    Data 14.795 (14.795)    Loss 6.4658e-01 (6.4658e-01)    Acc@1  72.58 ( 72.58)   Acc@5 100.00 (10
.00)
poch: [1][11/42]       Time  2.668 ( 3.844)    Data  0.000 ( 1.511)    Loss 6.3467e-01 (6.1317e-01)    Acc@1  70.97 ( 74.34)   Acc@5 100.00 (10
.00)
poch: [1][21/42]       Time  2.659 ( 3.324)    Data  0.000 ( 0.921)    Loss 6.3447e-01 (6.1653e-01)    Acc@1  74.19 ( 74.54)   Acc@5 100.00 (10
.00)
poch: [1][31/42]       Time  2.674 ( 3.113)    Data  0.000 ( 0.683)    Loss 6.1476e-01 (6.0427e-01)    Acc@1  73.39 ( 74.79)   Acc@5 100.00 (10
.00)
poch: [1][41/42]       Time  2.642 ( 3.004)    Data  0.000 ( 0.560)    Loss 5.6572e-01 (6.1071e-01)    Acc@1  75.00 ( 74.11)   Acc@5 100.00 (10
.00)
est: [1/9]     Time 17.069 (17.069)    Loss 7.7603e-01 (7.7603e-01)    Acc@1  70.16 ( 70.16)   Acc@5 100.00 (100.00)
*   Acc@1 67.764 Acc@5 100.000
poch: [2][ 1/42]       Time 15.295 (15.295)    Data 14.441 (14.441)    Loss 5.7661e-01 (5.7661e-01)    Acc@1  75.81 ( 75.81)   Acc@5 100.00 (10
.00)
poch: [2][11/42]       Time  2.661 ( 3.823)    Data  0.000 ( 1.481)    Loss 5.8050e-01 (5.7392e-01)    Acc@1  72.58 ( 75.73)   Acc@5 100.00 (10
.00)
poch: [2][21/42]       Time  2.680 ( 3.278)    Data  0.000 ( 0.863)    Loss 4.3577e-01 (5.4888e-01)    Acc@1  83.06 ( 76.50)   Acc@5 100.00 (10
.00)
poch: [2][31/42]       Time  2.688 ( 3.086)    Data  0.008 ( 0.643)    Loss 5.4270e-01 (5.4392e-01)    Acc@1  75.81 ( 77.00)   Acc@5 100.00 (10
.00)
poch: [2][41/42]       Time  2.687 ( 2.989)    Data  0.000 ( 0.531)    Loss 4.5086e-01 (5.4763e-01)    Acc@1  79.84 ( 76.91)   Acc@5 100.00 (10
.00)
est: [1/9]     Time 17.481 (17.481)    Loss 3.2404e-02 (3.2404e-02)    Acc@1  99.19 ( 99.19)   Acc@5 100.00 (100.00)
*   Acc@1 77.944 Acc@5 100.000
```