이미지 분류02

전진희

1.Image preprocessing

```
import json
import os
import glob
import numpy as np
from PIL import Image
from tqdm import tqdm
from torchvision.transforms import functional as F

# 라벨 정보
labels = ['crease', 'crescent_gap', 'inclusion', 'oil_spot',
          'punching_hole', 'rolled_plt', 'silk_spot',
          'waist_folding', 'water_spot', 'welding_line']

def crop_and_save_image(json_path, output_dir, train_ratio=0.9) :
    with open(json_path ,'r',   encoding='utf-8') as f :
        json_data = json.load(f)

    # train, val folder create
    train_dir = os.path.join(output_dir, 'train')
    val_dir = os.path.join(output_dir, 'val')
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(val_dir, exist_ok=True)
    """
    output_dir = "ex02_dataset"
    ex02_dataset
        train
        val
    """
    # labels folder create
    for label in labels :
        train_label_dir = os.path.join(train_dir, label)
        os.makedirs(train_label_dir, exist_ok=True)
        val_label_dir = os.path.join(val_dir, label)
        os.makedirs(val_label_dir, exist_ok=True)

    for filename in tqdm(json_data.keys()) :
        json_image = json_data[filename]
        width = json_image['width']
        height = json_image['height']
        file_name = json_image['filename']
        bboxes = json_image['anno']

        # image loader
        image_path = os.path.join('./metal_data/images', file_name)
        image = Image.open(image_path)
        image = image.convert("RGB")

        for bbox_idx, bbox in enumerate(bboxes) :
            label_name = bbox['label']
```

```
                bbox_xyxy = bbox['bbox']
                x1, y1, x2, y2 = bbox_xyxy

                # bounding box crop
                cropped_image = image.crop((x1, y1, x2, y2))

                # padding
                width_, height_ = cropped_image.size
                if width_ > height_ :
                    padded_image = Image.new(cropped_image.mode, (width_,width_), (0,))
                    padding = (0, int((width_ - height_) /2))
                else :
                    padded_image = Image.new(cropped_image.mode, (height_, height_), (0,))
                    padding = (int((height_ - width_)/2) ,0)

                padded_image.paste(cropped_image, padding)

                # image resize
                size=(255,255)
                resize_image = F.resize(cropped_image, size)

                # train val label folder image save
                if np.random.rand() < train_ratio :
                    save_dir = os.path.join(train_dir, label_name)
                else :
                    save_dir = os.path.join(val_dir, label_name)
                os.makedirs(save_dir, exist_ok=True)
                save_path = os.path.join(save_dir, f"{filename}_{label_name}_{bbox_idx}.png")
                padded_image.save(save_path)



if __name__ == "__main__" :
    json_path = "./metal_data/anno/annotation.json"
    output_dir = "./metal_dataset"

    crop_and_save_image(json_path,output_dir)
```

2. CustumDataset 클래스 만들기

```
import os
import cv2
import glob

from torch.utils.data import Dataset
from PIL import Image, ImageFile

class MyDataset(Dataset):
    def __init__(self, data_dir, transforms=None):
        self.data_dir = glob.glob(os.path.join(data_dir, "*", "*.png"))
        self.transforms = transforms
        self.label_dict = self.create_label_dict()

    def create_label_dict(self):
        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):
            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.transforms is not None:
                img = self.transforms(image=img)['image']

            return img, label_idx

        def __len__(self):
            return len(self.data_dir)
```

3. Train

```
import pandas as pd
import torch.nn as nn
import torch
import torchvision
import albumentations as A
from albumentations.pytorch import ToTensorV2
from torchvision.models.efficientnet import efficientnet_b0, EfficientNet
from torch.optim import AdamW
from torch.nn import CrossEntropyLoss
from tqdm import tqdm
from torch.utils.data import DataLoader
from metal_0717_02_Customdata import MyDataset
import cv2


def train(model, train_loader, val_loader, epochs, optimizer, criterion, device):
    best_val_acc = 0.0
    train_losses = []
    val_losses = []
    train_accs = []
    val_accs = []
    print("Train ....")
    for epoch in range(epochs):
        train_loss = 0.0
        val_loss = 0.0
        val_acc = 0.0
        train_acc = 0.0

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

```
leave=False)

            for i, (data, target) in enumerate(train_loader_iter):
                data = data.to(device, dtype=torch.float)
                target = target.to(device)

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

                train_loss += loss.item()
                # acc
                _, pred = torch.max(outputs, 1)
                train_acc += (pred == target).sum().item()

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

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

            # eval
            model.eval()
            with torch.no_grad():
                for data, target in val_loader:
                    data = data.to(device, dtype=torch.float)
                    target = target.to(device)

                    output = 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)

            train_losses.append(train_loss)
            train_accs.append(train_acc)
            val_losses.append(val_loss)
            val_accs.append(val_acc)

            # save model
            if val_acc > best_val_acc:
                torch.save(model.state_dict(), "./ex01_0714.pt")
                best_val_acc = val_acc
            print(f"Epoch [{epoch + 1}/{epochs}], Train loss [{train_loss:.4f}], "
                  f"Val loss [{val_loss:.4f}], Train ACC [{train_acc:.4f}], "
                  f"Val ACC [{val_acc:.4f}]")

    torch.save(model.state_dict(), "./metal_0717_last.pt")
    return model, train_losses, val_losses, train_accs, val_accs


def main():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model = efficientnet_b0(pretrained=True)
    model._fc = nn.Linear(1280, 10)
```

```
    model = model.to(device)

    # aug
    train_transforms = A.Compose([
        A.Resize(width=225, height=225),
        A.RandomShadow(),
        A.RandomBrightnessContrast(),
        A.HorizontalFlip(),
        A.VerticalFlip(),
        ToTensorV2(),
    ])

    val_transforms = A.Compose([
        A.Resize(width=255, height=255),
        ToTensorV2()
    ])

    # dataset dataloader
    train_dataset = MyDataset("./metal_dataset/train/", transforms=train_transforms)
    val_dataset = MyDataset("./metal_dataset/val/", transforms=val_transforms)

    train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=64, shuffle=False)

    # loss function optimizer, epochs
    epochs = 40
    criterion = CrossEntropyLoss().to(device)
    optimizer = AdamW(model.parameters(), lr=0.001, weight_decay=1e-2)

    train(model, train_loader, val_loader, epochs, optimizer, criterion, device)


if __name__ == "__main__":
    main()
```

```
Epoch : 1/40:   0%|          | 0/51 [00:00<?, ?it/s]Train ....
Epoch : 2/40:   0%|          | 0/51 [00:00<?, ?it/s]Epoch [1/40], Train loss [1.0581], Val loss [0.5727], Train ACC [0.7702], Val ACC [0.8347]
Epoch : 3/40:   0%|          | 0/51 [00:00<?, ?it/s]Epoch [2/40], Train loss [0.3563], Val loss [0.3645], Train ACC [0.8974], Val ACC [0.8943]
Epoch : 3/40:  94%|█████████| 48/51 [01:14<00:04,  1.57s/it, Loss=0.309]
```

최종적으로 모델이 훈련 중에 있습니다.