이미지 분류

전진희

프로젝트 개요

전처리된 데이터를 적절한 디렉토리에 위치시키고, 커스텀 데이터 클래스를 정의하여 데이터를 로드하는 과정을 포함합니다. 학습을 위해 사전 훈련된 모델을 불러와 사용하고, 성능을 향상시키기 위해 모델이나 손실 함수를 변경하여 다양한 실험을 수행하여 결과를 비교했습니다. 최종적으로는 인퍼런스 코드를 작성하여 모델을 사용해 예측을 수행했습니다.

데이터 준비

이전 시간 전처리한 4종류 치즈 이미지의 파일 구조를 정리해보겠습니다.

```
import os
import shutil
import glob
from sklearn.model_selection import train_test_split
'''
파일 구조 아래와 같이 만든다
cheese_dataset
 ㄴdata
  ㄴblue/camembert/cheddar/emmental
'''
class ImageMove :
   def __init__(self, org_folder):
      self.org_folder = org_folder
   def move_images(self):
      file_path_list = glob.glob(os.path.join(self.org_folder, "*" , "*.png"))
      for file_path in file_path_list :
         folder_name = file_path.split("\\")[1]
         if folder_name == "blue" :
            shutil.move(file_path, "./cheese_dataset/data/blue")
         elif folder_name == "camembert" :
            shutil.move(file_path, "./cheese_dataset/data/camembert")
         elif folder_name == "cheddar" :
            shutil.move(file_path, "./cheese_dataset/data/cheddar")
         elif folder_name == "emmental" :
            shutil.move(file_path, "./cheese_dataset/data/emmental")

'''
함수 실행
'''
#test = ImageMove("./final_cheese_data/")
#test.move_images()
```

```
'''
이제 파일 구조 만들 일 없이 그냥 돌리면 된다.
'''

class ImageDataMove:
    def __init__(self, org_dir, train_dir, val_dir):
        self.org_dir = org_dir
        self.train_dir = train_dir
        self.val_dir = val_dir

    def move_images(self):
        # 파일 경로 리스트 가져오기
                file_path_list01 = glob.glob(os.path.join(self.org_dir, "*", "blue", "*.png"))
        file_path_list02 = glob.glob(os.path.join(self.org_dir, "*", "camembert", "*.png"))
        file_path_list03 = glob.glob(os.path.join(self.org_dir, "*", "cheddar", "*.png"))
        file_path_list04 = glob.glob(os.path.join(self.org_dir, "*", "emmental", "*.png"))

        # 데이터 분할
                bl_train_data_list, bl_val_data_list = train_test_split(file_path_list01, test_size=0.2)
        cm_train_data_list, cm_val_data_list = train_test_split(file_path_list02, test_size=0.2)
        ch_train_data_list, ch_val_data_list = train_test_split(file_path_list03, test_size=0.2)
        em_train_data_list, em_val_data_list = train_test_split(file_path_list03, test_size=0.2)

        # 파일 이동
                self.copy_files(bl_train_data_list, os.path.join(self.train_dir, "blue"))
        self.copy_files(bl_val_data_list, os.path.join(self.val_dir, "blue"))
        self.copy_files(cm_train_data_list, os.path.join(self.train_dir, "camembert"))
        self.copy_files(cm_val_data_list, os.path.join(self.val_dir, "camembert"))
        self.copy_files(ch_train_data_list, os.path.join(self.train_dir, "cheddar"))
        self.copy_files(ch_val_data_list, os.path.join(self.val_dir, "cheddar"))
        self.copy_files(em_train_data_list, os.path.join(self.train_dir, "emmental"))
        self.copy_files(em_val_data_list, os.path.join(self.val_dir, "emmental"))

    def copy_files(self, file_list, mov_dir):
        os.makedirs(mov_dir, exist_ok=True)
        for file_path in file_list:
            shutil.copy2(file_path, mov_dir)

org_dir = "cheese_dataset"
train_dir = "./cheese_data/train"
val_dir = "./cheese_data/val"

'''
move_temp = ImageDataMove(org_dir, train_dir, val_dir)
move_temp.move_images()
'''
```

데이터 셋 클래스 준비

커스텀 데이터셋을 정의하는 클래스인 CustomDataset을 만들어 봅시다

```
import os

from torch.utils.data import Dataset
from PIL import Image
import glob

class CustomDataset(Dataset) :
    def __init__(self, data_dir, transform=None):
        # data_dir = ./data/train/
        self.data_dir = glob.glob(os.path.join(data_dir, "*", "*.png"))
        self.transform = transform
        self.label_dict = {"blue" : 0 , "camembert" : 1, "cheddar" : 2, "emmental" : 3}

    def __getitem__(self, item):
        image_path = self.data_dir[item]
        #image_path >>> ./data/train\waveshow\rock.00006_augmented_noise.png
        image = Image.open(image_path)
        image = image.convert("RGB")
        label_name = image_path.split("\\")[1]
        label = self.label_dict[label_name]

        if self.transform is not None :
            image = self.transform(image)

        return image ,label

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

모델 학습 시키기

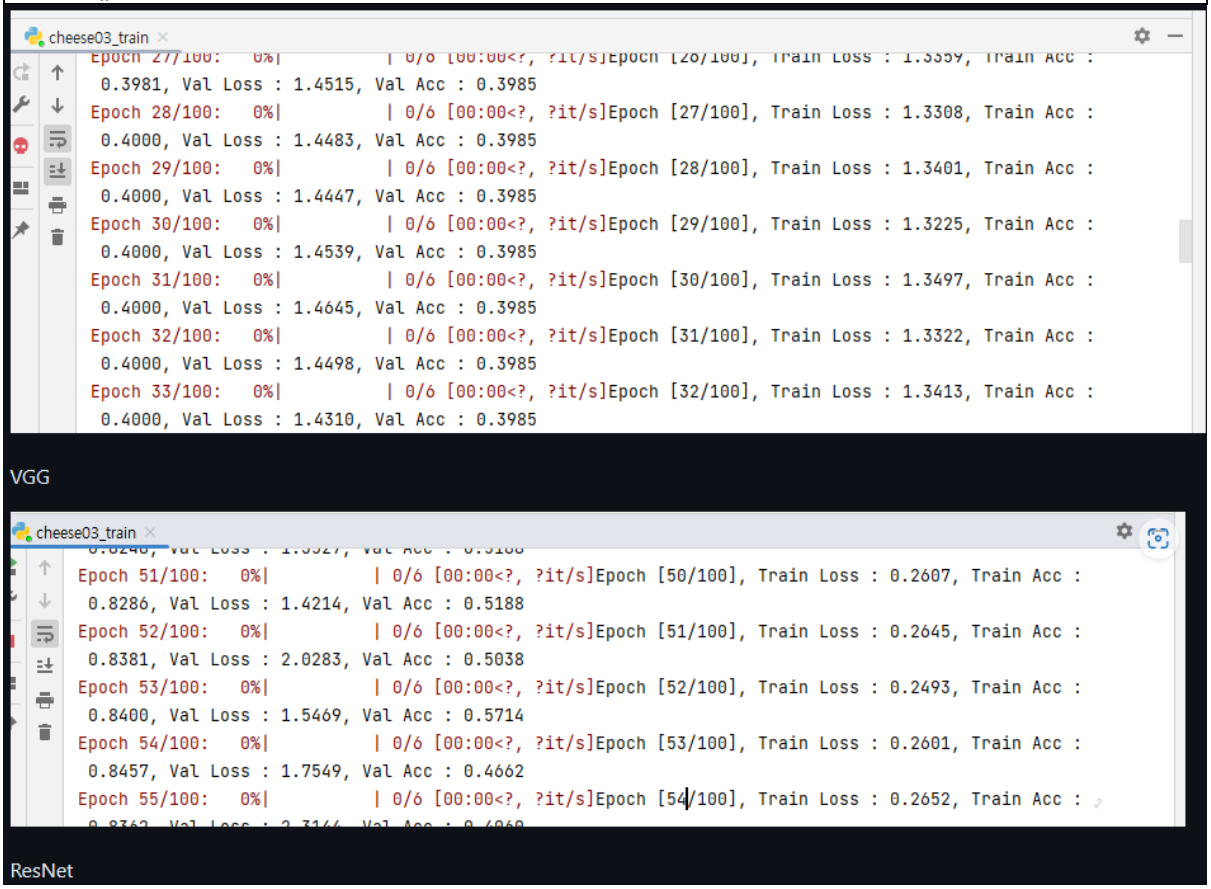
VGG, ResNet 등 여러 모델을 사용해보고 좋은 모델의 best.pt를 추출합니다.

```
import torch
import torchvision
import torchvision.transforms as transforms
from torchvision.models import vgg11, VGG11_Weights
from torchvision.models import ResNet18_Weights
import torchvision.models as models
from torch.optim import AdamW
from torch.nn import CrossEntropyLoss
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt
import torch.nn as nn
from cheese02_customdataset import CustomDataset
from tqdm import tqdm

def train(model, train_loader, val_loader, epochs, DEVICE, optimizer, criterion) :
    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)
        target = target.to(DEVICE)

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

        train_loss += loss.item()

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

        # print the loss
        if i % 10 == 9 :
            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)
            target = target.to(DEVICE)

            outputs = model(data)
            pred = outputs.argmax(dim=1, keepdim=True)
            val_acc += pred.eq(target.view_as(pred)).sum().item()
            val_loss += criterion(outputs, 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 the model with the best val acc
        if val_acc > best_val_acc :
            torch.save(model.state_dict(), 'best_cheese_model.pt')
            best_val_acc = val_acc

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

    return model, train_losses, val_losses, train_accs, val_accs

def main() :
    DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    # DEVICE_MPS = torch.device("mps") # mac m1 or m2

    #model = vgg11(weights=VGG11_Weights.DEFAULT)
    # ResNet 모델 불러오기
        model = models.resnet18(weights=ResNet18_Weights.DEFAULT)
    num_features = model.fc.in_features # 모델의 Fully Connected 층의 입력 특징 개수 가져오기
        model.fc = nn.Linear(num_features, 4) # 클래스 개수에 맞게 출력 층 변경
        model.to(DEVICE)

    """
    # transforms
    1. aug
    2. ToTensor
    3. Normalize
    """
    train_transform = transforms.Compose([

        transforms.Resize((224,224)),
        transforms.ToTensor()
    ])

    val_transform = transforms.Compose([

        transforms.Resize((224,224)),
        transforms.ToTensor()
    ])

    # dataset
    train_dataset = CustomDataset("./cheese_data/train/", transform=train_transform)
    val_dataset = CustomDataset("./cheese_data/val/", transform=val_transform)

    # dataloader
    train_loader = DataLoader(train_dataset, batch_size=100, num_workers=4,
                              pin_memory=True, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=100, num_workers=4,
                            pin_memory=True, shuffle=False)
    # import time
```

```python
    # import math
    # test = time.time()
    # math.factorial(100000)
    # test01 = time.time()
    # print(f"{test01 - test :.5f} sec")

    epochs = 100
    criterion = CrossEntropyLoss().to(DEVICE)
    optimizer = AdamW(model.parameters(), lr=0.001, weight_decay=1e-2)

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

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



VGG

ResNet

데이터 수가 부족하여 일정 epoch 이후 성능 향상이 되지 않음을 확인했습니다.

VGG와 ResNet 중에선 ResNet의 성능이 더 잘 나오는 것을 볼 수 있습니다.

```
Epoch [50/50], Train Loss : 0.3634, Train Acc : 0.8114, Val Loss : 2.1990, Val Acc : 0.4511
```

최종적으로 사용한 ResNet 훈련 모델 결과입니다. 조금 더 성능을 향상 시키기 위해 Loss function을 일반화 성능을 향상 시켜준다는 Label smoothing으로 바꿔주고 다시 돌려봤습니다.

```
Epoch 19/20:   0%|          | 0/3 [00:00<?, ?it/s]Epoch [18/20], Train Loss : 0.1819, Train Acc : 0.8305, Val Loss : 0.9724, Val Acc : 0.6241
Epoch 20/20:   0%|          | 0/3 [00:00<?, ?it/s]Epoch [19/20], Train Loss : 0.1775, Train Acc : 0.8305, Val Loss : 0.8948, Val Acc : 0.6165
Epoch [20/20], Train Loss : 0.1752, Train Acc : 0.8438, Val Loss : 0.9376, Val Acc : 0.6391
```

조금 더 나아졌습니다. 그러나 성능이 낮은 가장 큰 요인은 데이터 부족이라는 생각이 듭니다(예상)

인퍼런스 코드 만들고 돌려보기

학습된 모델을 사용하여 새로운 입력 데이터에 대한 예측을 수행하는 코드를 짜봅시다.

path 변수를 로드하여 이미지를 표시하는 과정에서 path 변수 수정에 어려움이 많아 직접 가져오는 방식으로 chat GPT를 통해 수정하였습니다.

```python
import torch
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

from torch.utils.data import DataLoader
from torchvision.models import resnet18
from cheese02_customdataset import CustomDataset
import cv2

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

    # model setting
    model = resnet18()
    num_features = model.fc.in_features
    model.fc = nn.Linear(num_features, 4)

    # .pt load
    model.load_state_dict(torch.load(f="./best_cheese_model.pt"))

    val_transforms = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
    ])

    test_dataset = CustomDataset("./cheese_data/val/", transform=val_transforms)
    test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)

    model.to(device)
    model.eval()

    correct = 0
    label_dict = {0: "blue", 1: "camembert", 2: "cheddar", 3: "emmental"}

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
```

```
        output = model(data)
        pred = output.argmax(dim=1, keepdim=True)

        target_label = label_dict[target.item()]
        true_label_text = f"true : {target_label}"

        pred_label = label_dict[pred.item()]
        pred_text = f"pred : {pred_label}"

        print(true_label_text, pred_text)

        img = transforms.functional.to_pil_image(data.squeeze(0).cpu())
        img = cv2.cvtColor(np.array(img), cv2.COLOR_RGB2BGR)
        img = cv2.resize(img, (500, 500))
        img = cv2.rectangle(img, (0, 0), (500, 100), (255, 255, 255), -1)
        img = cv2.putText(img, true_label_text, (0, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
        img = cv2.putText(img, pred_text, (0, 70),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

        cv2.imshow("test", img)
        if cv2.waitKey() == ord('q'):
            break

        correct += pred.eq(target.view_as(pred)).sum().item()

    cv2.destroyAllWindows()

    print("test set : Acc {}/{} [{:.0f}%]\n".format(
        correct, len(test_loader.dataset),
        100 * correct / len(test_loader.dataset)
    ))


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


```
true : emmental pred : emmental
true : emmental pred : emmental
true : emmental pred : emmental
true : emmental pred : cheddar
true : emmental pred : emmental
test set : Acc 86/133 [65%]
```

결과 이미지