from \_\_future\_\_ import print\_function, division

import torch

import torch.utils.data

import torch.nn as nn

import torch.optim as optim

from torch.optim import lr\_scheduler

from torchvision import datasets, models, transforms

import os

import copy

import time

import datetime

def main():

start = time.time()

# 이미지 데이터 전처리

data\_transforms = {

'train': transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # mean(r,b,g), stdv(r,b,g)

]),

'val': transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # mean(r,b,g), stdv(r,b,g)

]),

}

# Split된 train, test data load.

data\_dir = 'color'

image\_datasets = {x: datasets.ImageFolder(os.path.join(data\_dir, x), data\_transforms[x]) for x in ['train', 'val']}

# 학습시 batch\_size = 5로 설정.

dataloaders = {x: torch.utils.data.DataLoader(image\_datasets[x], batch\_size=5,

shuffle=True, num\_workers=2)

for x in ['train', 'val']}

dataset\_sizes = {x: len(image\_datasets[x]) for x in ['train', 'val']}

# GPU에 할당.

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

# 모델 Training

def train\_model(model, criterion, optimizer, scheduler, num\_epochs=25):

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_acc = 0.0

for epoch in range(num\_epochs):

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

print('-' \* 10)

for phase in ['train', 'val']:

if phase == 'train':

model.train()

else:

model.eval()

running\_loss = 0.0

running\_corrects = 0

# Iterate

for inputs, labels in dataloaders[phase]:

inputs = inputs.to(device)

labels = labels.to(device)

optimizer.zero\_grad()

with torch.set\_grad\_enabled(phase == 'train'):

outputs = model(inputs)

\_, preds = torch.max(outputs, 1)

loss = criterion(outputs, labels)

if phase == 'train':

loss.backward()

optimizer.step()

running\_loss += loss.item() \* inputs.size(0)

running\_corrects += torch.sum(preds == labels.data)

if phase == 'train':

scheduler.step()

epoch\_loss = running\_loss / dataset\_sizes[phase]

epoch\_acc = running\_corrects.double() / dataset\_sizes[phase]

print('{} Loss: {:.4f} Acc: {:.4f}'.format(

phase, epoch\_loss, epoch\_acc))

# deep copy the model

if phase == 'val' and epoch\_acc > best\_acc:

best\_acc = epoch\_acc

best\_model\_wts = copy.deepcopy(model.state\_dict())

print('Best val Acc: {:4f}'.format(best\_acc))

# load best model weights

model.load\_state\_dict(best\_model\_wts)

return model

model\_ft = models.densenet201(pretrained=True) # Pretrained with imagenet & densenet201

num\_ftrs = model\_ft.classifier.in\_features

model\_ft.fc = nn.Linear(num\_ftrs, 8) # 직접 8개로 분류하도록 수정

model\_ft = model\_ft.to(device) # GPU에 할당

criterion = nn.CrossEntropyLoss()

optimizer\_ft = optim.SGD(model\_ft.parameters(), lr=0.001, momentum=0.9) # SGD with lr=0.001, momentum=0.9

exp\_lr\_scheduler = lr\_scheduler.StepLR(optimizer\_ft, step\_size=7, gamma=0.1)

# Train\_model

model\_ft = train\_model(model\_ft, criterion, optimizer\_ft, exp\_lr\_scheduler,

num\_epochs=20)

sec = time.time() - start

times = str(datetime.timedelta(seconds=sec)).split(".")

times = times[0]

print(times)

# save model

torch.save(model\_ft, "top.pt")

if \_\_name\_\_ == '\_\_main\_\_':

torch.multiprocessing.freeze\_support()

main()