2018312856 KoSungwon

I have made 8 notebook files, corresponding to followings:

```
1. Low Quality data - face2face dataset / ResNet18
2. Low Quality data - face2face dataset / EfficientNetB0
3. Low Quality data - NeuralTexture dataset / ResNet18
4. Low Quality data - NeuralTexture dataset / EfficientNetB0

5. High Quality data - face2face dataset / ResNet18
6. High Quality data - face2face dataset / EfficientNetB0
7. High Quality data - NeuralTexture dataset / ResNet18
8. High Quality data - NeuralTexture dataset / EfficientNetB0
```

due to importation problem with timm, I used ResNet18 and EfficientNetB0. All eight files are conducted in same way, but they just differ from model selection, and data. I will explain my code for first Low Quality data – face2face dataset / ResNet18

1. import modules.

```python
# 필요한 라이브러리 호출 / 디바이스 확인, 할당
import torch
import torchvision
from torch.utils.data import DataLoader, Dataset
from torchvision import transforms # 이미지 변환 기능을 제공하는 라이브러리
from torch.autograd import Variable
from torch import optim
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models # 다양한 파이토치 네트워크를 사용할 수 있게 해주는 패키지
from torchsummary import summary

import os
import cv2
import time
import glob
from PIL import Image
from tqdm import tqdm_notebook as tqdm
import random
import copy
from matplotlib import pyplot as plt
import numpy as np
```

```python
USE_CUDA = torch.cuda.is_available()
device = torch.device('cuda:0' if USE_CUDA else 'cpu')

print('CUDA 사용 가능 여부: ', USE_CUDA)
print('현재 사용 device: ', device)
print('CUDA Index', torch.cuda.current_device())
print('GPU 이름: ', torch.cuda.get_device_name())
print('GPU 개수: ', torch.cuda.device_count())
```

```
CUDA 사용 가능 여부:  True
현재 사용 device:  cuda:0
CUDA Index 0
GPU 이름:  GeForce GTX 1650 Ti
GPU 개수:  1
```

also, I checked for GPU resources, since I did it on my local.

2. dataset file

original dataset file doesn't fit to PyTorch's DataLoader. To use dataloader, I have to make class folders.

```python
# 이미지 데이터셋 나누기

import shutil

def make_class_folders(target_path):
    real_dir = os.path.join(target_path, 'real')
    fake_dir = os.path.join(target_path, 'fake')
    os.mkdir(real_dir)
    os.mkdir(fake_dir)


    file_name_list = os.listdir(target_path)
    for file_name in file_name_list:
        if file_name == 'fake' or file_name == 'real':
            pass
        else:
            src = os.path.join(target_path, file_name)

            if 'real' in file_name:
                dst = os.path.join(real_dir, file_name)
            else:
                dst = os.path.join(fake_dir, file_name)

            shutil.move(src, dst)
```

« DFdetection_HW3 › DF_data › Low_Quality › f2f_data › test ›

이름	수정한 날짜	유형
fake	2023-05-18 오전 9:19	파일
real	2023-05-18 오전 9:19	파일

so I made it into two class folders.

3. DataLoader

```python
mean = (0.485, 0.456, 0.406)
std = (0.229, 0.224, 0.225)

transform = transforms.Compose(
    [transforms.Resize([256, 256]),  # 이미지의 크기를 256x256으로 조정
     transforms.RandomResizedCrop(224), # 이미지 데이터셋 확장. 랜덤한 비율로 자른 후 224x224으로 맞춘다.
     transforms.RandomHorizontalFlip(), # 이미지를 랜덤하게 수평으로 뒤집는다.
     transforms.ToTensor(), # 이미지를 텐서로 변환
     transforms.Normalize(mean, std)
    ]
)

train_dataset = torchvision.datasets.ImageFolder(
    data_path,
    transform=transform
)

train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=32,
    num_workers=8,
    shuffle=True
)
```

I defined dataloader. For training, I used data augmentation. It resizes original image, and filp image for 50% chance. Also, Normalize image data with given values.

```python
print(len(train_dataset))
```

```
1200
```

I checked train dataset, and it correctly found dataset. (Real 600 + Fake 600)

Then, I checked train data.

```python
# 학습에 사용될 이미지 출력
import warnings
warnings.filterwarnings('ignore')

samples, labels = iter(train_loader).next()
classes = {0: 'fake', 1: 'real'}
fig = plt.figure(figsize=(16,24))
for i in range(24):
    a = fig.add_subplot(4, 6, i+1)
    a.set_title(classes[labels[i].item()])
    a.axis('off')
    a.imshow(np.transpose(samples[i].numpy(), (1, 2, 0)))

plt.subplots_adjust(bottom=0.2, top=0.6, hspace=0)
```

result:

As you can see, images are distorted as I intended.

4. Model

```
resnet18 = models.resnet18(pretrained=True) # 사전학습된 가중치를 사용하겠음!

#ResNet18: 50개의 계층으로 구성된 합성곱 신경망. 입력 제약이 크고, RAM이 충분하지 않으면 훈련 속도
def set_parameter_requires_grad(model, feature_extracting=True):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False    # conv 층, pooling 층 -> 파라미터에 대해서 gradient

set_parameter_requires_grad(resnet18)
#17. ResNet18에 완전연결층 추가
resnet18.fc = nn.Linear(512, 2)

#18. 모델의 파라미터 값 확인
for name, param in resnet18.named_parameters():
    if param.requires_grad:
        print(name, param.data)

#19. 모델 객체 생성 및 손실 함수 정의
model = models.resnet18(pretrained=True)

for param in model.parameters():
    param.requires_grad = False
```

```
model.fc = torch.nn.Linear(512,2)
for param in model.fc.parameters():
    param.requires_grad = True

optimizer = torch.optim.Adam(model.fc.parameters())
cost = torch.nn.CrossEntropyLoss()
model.to(device)
summary(model, input_size = (3, 224, 224))
```

For ResNet, I used pretrained model in Torchvision. I froze weights in Feature extraction parts with convolutional layers, and added new classifier layer at the top, with 2 output for binary classification.

For EfficientNet, I used pretrained model from github.

```python
efficientnet = torch.hub.load('NVIDIA/DeepLearningExamples:torchhub', 'nvidia_efficientnet_b0', pretrained=True)

efficientnet.to(device)
def set_parameter_requires_grad(model, feature_extracting=True):
    if feature_extracting:
        for param in model.parameters():
            param.requires_grad = False    # conv 층, pooling 층 -> 파라미터에 대해서 gradient 계산하지 않고, update도 없다.

set_parameter_requires_grad(efficientnet)
```

```python
new_classifier = nn.Sequential(
    nn.AdaptiveAvgPool2d(output_size=1),
    nn.Flatten(),
    nn.Dropout(p=0.2, inplace=False),
    nn.Linear(in_features=1280, out_features=2, bias=True)
)

efficientnet.classifier = new_classifier.to(device)
for param in efficientnet.classifier.parameters():
    param.requires_grad = True
summary(efficientnet, input_size = (3, 224, 224))
```

5. Train model

```python
# 손실함수, 옵티마이저 정의 및 전달
optimizer = torch.optim.Adam(efficientnet.classifier.parameters())
cost = torch.nn.CrossEntropyLoss()
```

✓ 0.0s

```python
# 모델 학습을 위한 함수 생성
def train_model(model, dataloaders, criterion, optimizer, device, 
    since = time.time()
    acc_history = []
    loss_history = []
    best_acc = 0.0
```

I used Adam optimizer, and CrossEntropyLoss function. when training, I track model's accuracy and record the best one. After 100 epochs, I saved the best model, and store them in models folder.

이름	수정한 날짜	유형
HQ_f2f_EfficientNet_best.pth	2023-05-18 오후 5:48	PTH 파일
HQ_f2f_ResNet_best.pth	2023-05-18 오후 5:20	PTH 파일
HQ_nt_EfficientNet_best.pth	2023-05-18 오후 9:13	PTH 파일
HQ_nt_ResNet_best.pth	2023-05-18 오후 8:51	PTH 파일
LQ_f2f_EfficientNet_best.pth	2023-05-18 오후 1:08	PTH 파일
LQ_f2f_ResNet_best.pth	2023-05-18 오후 2:11	PTH 파일
LQ_nt_EfficientNet_best.pth	2023-05-18 오후 4:56	PTH 파일
LQ_nt_ResNet_best.pth	2023-05-18 오후 3:56	PTH 파일

Training process looks like:

```
Epoch 97/100
----------
Loss: 0.237294 Acc: 0.900833

Epoch 98/100
----------
Loss: 0.254547 Acc: 0.897500

Epoch 99/100
----------
Loss: 0.278823 Acc: 0.885833

Epoch 100/100
----------
Loss: 0.242955 Acc: 0.893333

Training complete in 20m 43s
Best Acc: 0.910000
```

6. test model.

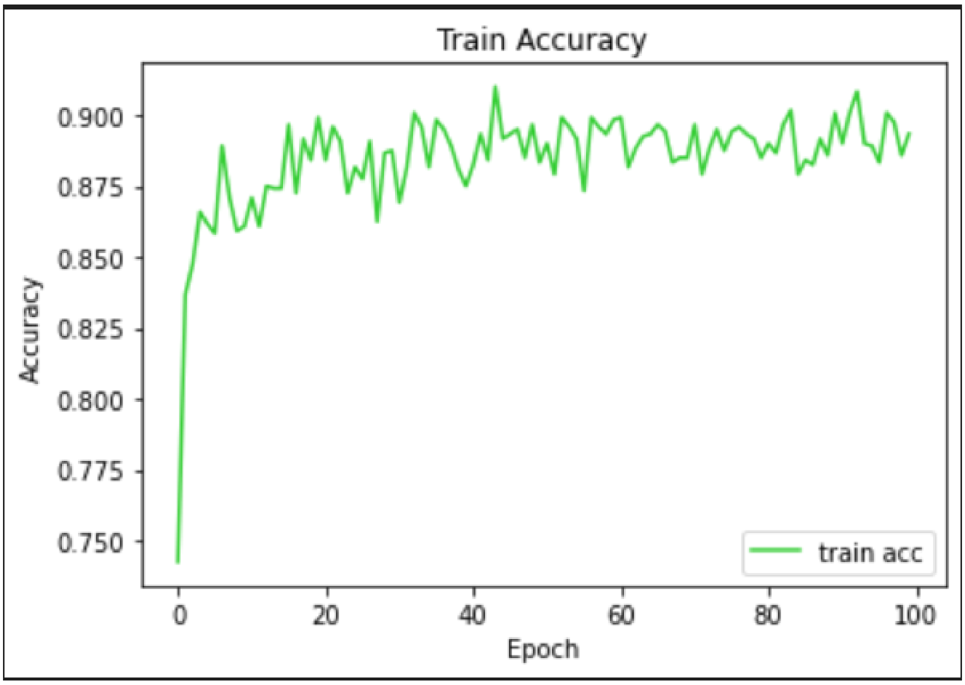
first, train loader. when training, It doesn't distort images.

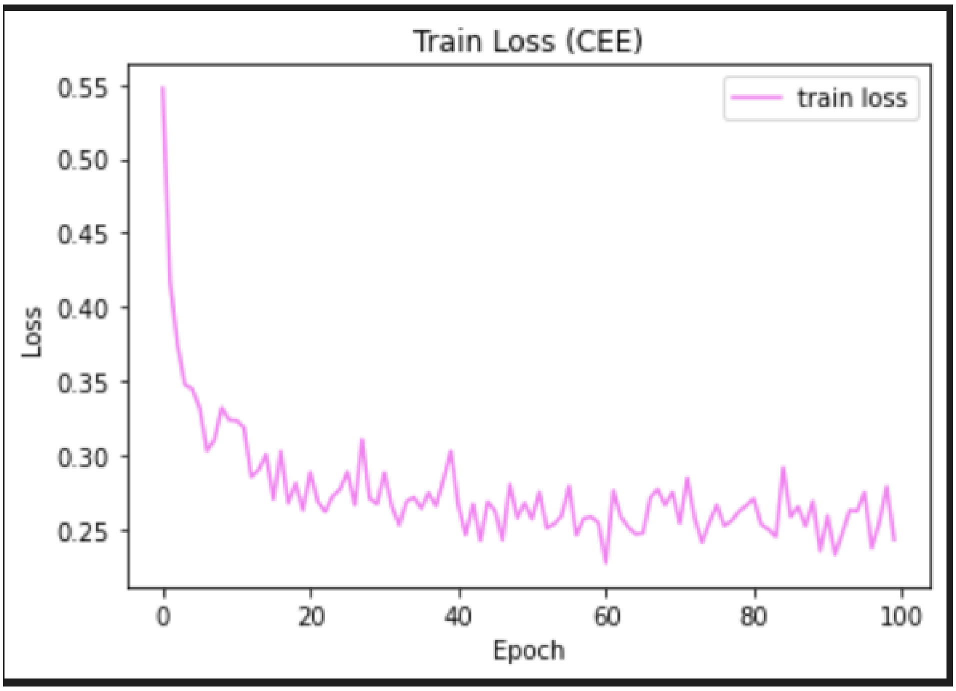
```
# 테스트 데이터 호출 및 전처리
transform = transforms.Compose(
    [transforms.Resize(224),
     transforms.CenterCrop(224),
     transforms.ToTensor(),
     transforms.Normalize(mean, std)]
)
test_dataset = torchvision.datasets.ImageFolder(
    root = test_path,
    transform = transform
)
test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=32,
    num_workers=1,
    shuffle=True
)
```

```
print(len(test_dataset))
✓ 0.0s
398
```

```
Acc: 0.9121

Test complete in 0m 3s
Best Acc: 0.912060
              precision    recall  f1-score   support

           0       0.94      0.88      0.91       199
           1       0.89      0.94      0.91       199

    accuracy                           0.91       398
   macro avg       0.91      0.91      0.91       398
weighted avg       0.91      0.91      0.91       398
```

plot loss, and accuracy of model training process.





2. Accuracy, F1 score, precision, recall of each cases.

2-1. Low Quality data – face2face dataset / ResNet18

```
Acc: 0.9322

Test complete in 0m 3s
Best Acc: 0.932161
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       199
           1       0.95      0.91      0.93       199

    accuracy                           0.93       398
   macro avg       0.93      0.93      0.93       398
weighted avg       0.93      0.93      0.93       398
```

2-2. Low Quality data - face2face dataset / EfficientNetB0

```
Acc: 0.9246

Test complete in 0m 8s
Best Acc: 0.924623
              precision    recall  f1-score   support

           0       0.95      0.90      0.92       199
           1       0.90      0.95      0.93       199

    accuracy                           0.92       398
   macro avg       0.93      0.92      0.92       398
weighted avg       0.93      0.92      0.92       398
```

2-3. Low Quality data – NeuralTexture dataset / ResNet18

```
Acc: 0.8920

Test complete in 0m 3s
Best Acc: 0.891960
              precision    recall  f1-score   support

           0       0.91      0.86      0.89       199
           1       0.87      0.92      0.89       199

    accuracy                           0.89       398
   macro avg       0.89      0.89      0.89       398
weighted avg       0.89      0.89      0.89       398
```

2-4. Low Quality data – NeuralTexture dataset / EfficientNetB0

```
Acc: 0.9196

Test complete in 0m 3s
Best Acc: 0.919598
              precision    recall  f1-score   support

           0       0.95      0.89      0.92       199
           1       0.90      0.95      0.92       199

    accuracy                           0.92       398
   macro avg       0.92      0.92      0.92       398
weighted avg       0.92      0.92      0.92       398
```

2-5. High Quality data – face2face dataset / ResNet18

```
Acc: 0.7312

Test complete in 0m 3s
Best Acc: 0.731156
              precision    recall  f1-score   support

           0       0.78      0.65      0.71       199
           1       0.70      0.81      0.75       199

    accuracy                           0.73       398
   macro avg       0.74      0.73      0.73       398
weighted avg       0.74      0.73      0.73       398
```

2-6. High Quality data - face2face dataset / EfficientNetB0

```
Acc: 0.7487

Test complete in 0m 7s
Best Acc: 0.748744
              precision    recall  f1-score   support

           0       0.79      0.67      0.73       199
           1       0.72      0.82      0.77       199

    accuracy                           0.75       398
   macro avg       0.75      0.75      0.75       398
weighted avg       0.75      0.75      0.75       398
```

2-7. High Quality data – NeuralTexture dataset / ResNet18

```
Acc: 0.7111

Test complete in 0m 4s
Best Acc: 0.711055
              precision    recall  f1-score   support

           0       0.70      0.73      0.72       199
           1       0.72      0.69      0.71       199

    accuracy                           0.71       398
   macro avg       0.71      0.71      0.71       398
weighted avg       0.71      0.71      0.71       398
```

2-8. High Quality data – NeuralTexture dataset / EfficientNetB0

```
Acc: 0.9121

Test complete in 0m 3s
Best Acc: 0.912060
              precision    recall  f1-score   support

           0       0.94      0.88      0.91       199
           1       0.89      0.94      0.91       199

    accuracy                           0.91       398
   macro avg       0.91      0.91      0.91       398
weighted avg       0.91      0.91      0.91       398
```

Summary. Each score is written in perspective of (0) – fake detection.

	Accuracy	F1 score	Precision	Recall
1. LQ f2f ResNet	0.93	0.93	0.91	0.95
2. LQ f2f Eff_Net	0.92	0.92	0.90	0.95
3. LQ nt ResNet	0.89	0.89	0.91	0.92
4. LQ nt Eff_Net	0.92	0.92	0.95	0.89
5. HQ f2f ResNet	0.73	0.71	0.78	0.65
6. HQ f2f Eff_Net	0.75	0.75	0.79	0.67
7. HQ nt ResNet	0.71	0.72	0.70	0.73
8. HQ nt Eff_Net	0.91	0.91	0.94	0.88

Overall, models show better result in Low-Quality dataset. In perspective of models, ResNet performed better in Low Quality dataset. But Efficient net with High quality dataset showed very good results with accuracy of 91% .

I think too much resolution cause overfitting to models, referring from results above. But it is surprising that trained models can actually tell which is fake and real.