

REPORT

Term_Project



과목명	딥러닝
담당교수	정우환 교수님
학생이름	박준우
학과	인공지능학과
학번	2021006253
제출일	2023.11.30

HANYANG UNIVERSITY

사용한 모델

`CLASS/vision.models.EfficientNet_B3_Weights(value)` [\[SOURCE\]](#)

The model builder above accepts the following values as the `weights` parameter.

`EfficientNet_B3_Weights.DEFAULT` is equivalent to `EfficientNet_B3_Weights.IMAGENET1K_V1`. You can also use strings, e.g. `weights='DEFAULT'` or `weights='IMAGENET1K_V1'`.

EfficientNet_B3_Weights.IMAGENET1K_V1:

These weights are ported from the original paper. Also available as `EfficientNet_B3_Weights.DEFAULT`.

acc@1 (on ImageNet-1K)	82.008
acc@5 (on ImageNet-1K)	96.054
categories	tench, goldfish, great white shark, ... (997 omitted)
min_size	height=1, width=1
recipe	link
num_params	12233232
GFLOPS	1.83
File size	47.2 MB

모델의 크기가 55MB 를 초과하면 안 되므로 47MB 정도의 크기를 가지면서 어느 정도 정확도가 높은 EffiecientNet_B3 모델을 채택하였다.

TTA (Test Time Augmentation)이란

- Model 을 테스트 할 때에도, Data Augmentation 을 함.
- TTA 는 일종의 Ensemble 기법
- Ensemble 이란 일반적으로 어떤 데이터에 대해 여러 모델의 예측결과를 평균내어 편향된 데이터를 억제하는 역할을 함으로써 정확도를 높이는 데에 사용
- 원본 이미지를 flip 및 rotation, zoom 등을 하여, 원본으로부터 변형된 여러가지 Image Augmentation 에 평가를 실시하여, 최종 분류값이 무엇인지 예측하는 기법.
- 모델에 한 가지의 이미지를 주는 것보다는 여러가지 변형된 이미지를 주어, 평가를 하게 되면, 발생하는 오차는 작아짐
- TTA 를 쓰게 되면, 모델이 편향된 학습결과를 가지고 있을 때, 그러한 편향에서 벗어나 좀 더 좋은 예측을 할 수 있게 된다.

Source code for train.py

```
import argparse

import numpy as np
from tqdm import tqdm
from utils._utils import make_data_loader
from model import BaseModel

import torch
# from torchvision.models import resnet18, ResNet18_Weights
# 다른 모델
from torchvision.models import efficientnet_b3, EfficientNet_B3_Weights#
from torchvision import transforms#

def acc(pred,label):
    pred = pred.argmax(dim=-1)
    return torch.sum(pred == label).item()

# Define TTA transforms#
tta_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(10),
])#

def validate(model, data_loader, device):#
    model.eval()
    total = 0
    correct = 0
    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total#
```

```

def train(args, data_loader, mode):
    """
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=args.learning_rate)

    for epoch in range(args.epochs):
        train_losses = []
        train_acc = 0.0
        total=0
        print(f"[Epoch {epoch+1} / {args.epochs}]")

        model.train()
        pbar = tqdm(data_loader)
        for i, (x, y) in enumerate(pbar):
            image = x.to(args.device)
            label = y.to(args.device)
            optimizer.zero_grad()

            output = model(image)

            label = label.squeeze()
            loss = criterion(output, label)
            loss.backward()
            optimizer.step()

            train_losses.append(loss.item())
            total += label.size(0)

            train_acc += acc(output, label)

        epoch_train_loss = np.mean(train_losses)
        epoch_train_acc = train_acc/total

        print(f'Epoch {epoch+1}')
        print(f'train_loss : {epoch_train_loss}')
        print(f'train_accuracy : {:.3f}'.format(epoch_train_acc*100))

        # Save the model and validate with TTA every 4 epochs#
        if (epoch + 1) % 1 == 0:
            general_accuracy = validate(model, val_loader, args.device)
            # tta_accuracy = validate_with_tta(model, val_loader, args.device)

            print('Validation accuracy after epoch {}: {:.3f}%'.format(epoch + 1,
general_accuracy * 100))
            # print('Validation accuracy with TTA after epoch {}: {:.3f}%'.format(epoch +
1, tta_accuracy * 100))

        # Save the model's state_dict
        torch.save(model.state_dict(), f'{args.save_path}/model_epoch_{epoch+1}.pth')#

```

```

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description='2023 DL Term Project')
    parser.add_argument('--save-path',
default='/content/drive/MyDrive/Term_Project/checkpoints/', help="Model's state_dict")
    parser.add_argument('--data', default='/content/drive/MyDrive/Term_Project/data/',
type=str, help='data folder')
    args = parser.parse_args()

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    args.device = device

    # hyperparameters
    args.epochs = 40#
    args.learning_rate = 0.001#
    args.batch_size = 16#

    # check settings
    print("=====")
    print("Save path:", args.save_path)
    print('Using Device:', device)
    print('Number of usable GPUs:', torch.cuda.device_count())

    # Print Hyperparameter
    print("Batch_size:", args.batch_size)
    print("learning_rate:", args.learning_rate)
    print("Epochs:", args.epochs)
    print("=====")

    # Make Data loader and Model
    train_loader, val_loader = make_data_loader(args)
    # model = BaseModel()

    # torchvision model
    # Initialize the EfficientNet_B3 model with pre-trained weights
    weights = EfficientNet_B3_Weights.DEFAULT#
    model = efficientnet_b3(weights=weights)#

    # Adjust the number of output features to the number of classes in your dataset
    num_classes = 10
    #num_features = model.fc.in_features # edited
    #model.fc = nn.Linear(num_features, num_classes) # edited

    model.classifier[1] = torch.nn.Linear(model.classifier[1].in_features, num_classes)#

    # 체크포인트 로드

    model.to(device)
    print(model)

    # Training The Model
    train(args, train_loader, model)

```

-기존에 있던 ResNet18 대신 파일 크기가 55MB 이하인 모델 중 Efficient_b3 모델을 사용

-기존 skeleton_code 는 ResNet 을 사용하여 마지막에 fully connected 를 사용하여 데이터셋의 클래스 수에 맞게 조정하지만, Efficient_B3 모델은 FC 를 사용하지 않고 모델의 분류기 레이어를 가지고 클래스 수에 맞게 조정함.

```
def validate(model, data_loader, device):#
    model.eval()
    total = 0
    correct = 0
    with torch.no_grad():
        for images, labels in data_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total#
```

-validate 함수를 통해 데이터 로더에서 배치를 순차적으로 로드하여 각 배치에 대해 모델은 이미지를 입력으로 받아 예측을 수행하고, 예측된 레이블과 실제 레이블을 비교하여 정확도를 계산.

- model.eval() 을 통해 모델을 평가 모드로 설정 -> 훈련 중에만 사용되는 기능들을 비활성화함.

- with torch.no_grad()을 통해 해당 블록 안에서 진행되는 연산들은 그라디언트 계산을 수행하지 않음. 이를 통하여 메모리 사용량을 줄이고 연산 속도를 향상할 수 있음

- 검증 데이터셋의 이미지와 레이블을 배치 단위로 순회하며 이미지와 레이블을 디바이스(GPU)로 옮김.

- 이후 모델에 이미지를 입력하여 output 을 얻음.

- _, predicted = torch.max(outputs, 1) 에서 각 예측 결과에서 가장 높은 확률을 가진 레이블을 선택함 torch.max 는 최대값과 인덱스를 반환하는데 여기서는 인덱스만 사용하기에 앞에는 _ 사용.

- total += labels.size(0)에서는 total 에 현재 배치의 이미지 수를 더함

- correct += (predicted == labels).sum().item() 에서는 모델이 얼마나 많은 이미지를 정확하게 분류했는지를 계산.

- predicted == labels 는 모델의 예측이 실제 레이블과 일치하는지를 확인하는 Boolean 연산.

- 이 연산의 결과는 각 이미지에 대해 True 또는 False 값을 가지는 텐서.

- .sum() 메소드는 이 텐서에서 True 값의 개수, 즉 정확한 예측의 수를 계산.

- 마지막으로, .item() 메소드는 이 숫자를 파이썬 정수로 변환하며 correct 변수에 정확한 예측의 총 수를 누적함.

Source code for run(without TTA)

```
import argparse
import os
import torch
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
# from torchvision.models import resnet18, ResNet18_Weights
from torchvision.models import efficientnet_b3, EfficientNet_B3_Weights
from model import BaseModel
from tqdm import tqdm
from PIL import Image
import torch.nn as nn # edited

class ImageDataset(Dataset):

    def __init__(self, root_dir, transform=None, fmt=':04d', extension='.jpg'):
        self.root_dir = root_dir
        self.fmtstr = '{' + fmt + '}' + extension
        self.transform = transform

    def __len__(self):
        return len(os.listdir(self.root_dir))

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_name = self.fmtstr.format(idx)
        img_path = os.path.join(self.root_dir, img_name)
        img = Image.open(img_path).convert('RGB')
        data = self.transform(img)
        return data

def inference(args, data_loader, model):
    """ model inference """
    model.eval()
    preds = []
    with torch.no_grad():
        pbar = tqdm(data_loader)
        for i, x in enumerate(pbar):

            image = x.to(args.device)

            y_hat = model(image)

            y_hat.argmax()

            _, predicted = torch.max(y_hat, 1)
            preds.extend(map(lambda t: t.item(), predicted))

    return preds
```

```

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='2023 DL Term Project')
    parser.add_argument('--load-model', default='checkpoints/model_epoch_40.pth',
help="Model's state_dict")
    parser.add_argument('--batch-size', default=16, help='test loader batch size')
    parser.add_argument('--dataset', default='test_images/', help='image dataset
directory')

    args = parser.parse_args()

    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    args.device = device

    # torchvision model
    num_classes = 10
    model = efficientnet_b3()#
    #num_features = model.fc.in_features # edited
    #model.fc = nn.Linear(num_features, num_classes) # edited
    model.classifier[1] = torch.nn.Linear(model.classifier[1].in_features, num_classes)#

    model.load_state_dict(torch.load(args.load_model, map_location=device))#
    model.to(device)

    # load dataset in test image folder
    # you may need to edit transform
    test_data = ImageDataset(args.dataset,
transform=transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor()]))
    test_loader = torch.utils.data.DataLoader(test_data, batch_size=args.batch_size)

    # write model inference
    preds = inference(args, test_loader, model)

    with open('result.txt', 'w') as f:
        f.writelines('\n'.join(map(str, preds)))

```


Source code for run(with TTA)

```
# Define TTA transforms#
tta_transforms = transforms.Compose([
    transforms.RandomHorizontalFlip(),          # 이미지를 무작위로 수평으로 뒤집는 변환
    transforms.RandomRotation(10),             # 이미지를 최대 10 도까지 무작위로 회전시키는 변환
])#
```

```
def inference_with_tta(args, data_loader, model, tta_steps=10):
    """ model inference with TTA """
    model.eval()
    final_preds = []

    with torch.no_grad():###
        pbar = tqdm(data_loader)
        for images in pbar:
            images = images.to(args.device)
            tta_preds = []
            for _ in range(tta_steps):
                # Apply TTA transforms and make predictions
                augmented_images = tta_transforms(images)
                outputs = model(augmented_images).softmax(dim=-1)
                tta_preds.append(outputs.cpu())

            # Average the predictions across the TTA steps
            tta_preds = torch.mean(torch.stack(tta_preds), dim=0)
            final_preds.append(tta_preds.argmax(dim=-1).numpy())

    # Concatenate all batches
    final_preds = np.concatenate(final_preds)
    return final_preds###
```

```
test_data = ImageDataset(args.dataset, transform=transforms.Compose([
    transforms.Resize((224, 224)),    #resize 적용
    transforms.ToTensor(),
    # Include TTA transforms during dataset creation if you want to visualize the augmented images
]))
test_loader = torch.utils.data.DataLoader(test_data, batch_size=args.batch_size)
```

- 테스트셋을 보니까 scale 이 다른 이미지(32x32, 256x256 등등)가 많아서 transform 에 RandomResizeTransform 을 추가 사용.

```
test_data = ImageDataset(args.dataset, transform=transforms.Compose([transforms.Resize((224,
224)), transforms.ToTensor()])))
```

-기존 run.py) 데이터 로더에서 배치를 순차적으로 로드합니다. 각 배치에 대해 모델은 이미지를 입력으로 받아 예측을 수행하고, 예측된 레이블을 결과 리스트에 추가.

-이미지의 원본 형태를 사용하여 모델의 성능을 평가하려는 경우에 적합.

- TTA 버전) 각 이미지에 대해 TTA 단계(tta_steps)만큼 여러 번의 예측을 수행, 각 예측은 원본 이미지의 증강된 버전에 대해 수행하며 평균화되어 최종 예측을 형성.

- run.py(기존)은 각 이미지에 대해 단일 예측을 수행.

- run.py(tta)은 각 이미지에 대해 여러 증강을 적용하고 여러 예측을 수행한 후, 이를 평균화하여 최종 예측을 결정.

- transforms.RandomHorizontalFlip(),transforms.RandomRotation(10) 을 통해 모델이 수평방향와 회전에 대해 견고하도록 함.

```
!pip install --upgrade torch torchvision

Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu118)
Collecting torch
  Downloading torch-2.1.1-cp310-cp310-manylinux1_x86_64.whl (670.2 MB)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.16.0+cu118)
Collecting torchvision
  Downloading torchvision-0.16.1-cp310-cp310-manylinux1_x86_64.whl (6.8 MB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: Jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Downloading nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Downloading nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Downloading nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Downloading nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Downloading nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Downloading nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
```

- `!pip install --upgrade torch torchvision`을 통해 torchvision 에 대한 업데이트를 진행 이것을 하기 전까지는 모델을 실행함에 있어 오류가 발생하여 매 런타임마다 실행해 주었어야 했음.

```
[ ] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

!cd /content/drive/MyDrive/Term_Project/

/content/drive/MyDrive/Term_Project

[ ] !unzip skeleton_code.zip

/root/.cache/torch/hub/checkpoints/

[ ] %cd /root/.cache/torch/hub/checkpoints/

[ ] !ls

checkpoints  model.py      result.txt   Term_Project.ipynb  test.py      Untitled1.ipynb
data         __pycache__  run.py       test_images         train.py     utils
```

- `from google.colab import drive`
`drive.mount('/content/drive')`
으로 드라이브를 마운트
- `%cd /content/drive/MyDrive/Term_Project/`
프로젝트로 이동
- `!unzip skeleton_code.zip` 으로 압축파일을 해제하였지만 확인해보니 데이터가 모두 업로드 되지 않았아서 나중에 다시 따로 업로드 하였음.
- 기존 basemodel 를 확인해 보고자 실행해서 생성된 model.pth 파일이 남아있었는지 efficient_b3 모델을 실행하려 했더니 에러가 발생하여 해당 경로로 가서 삭제해줌
- 이후 `!ls` 통해 파일이 다 있음을 확인하고 `!train.py` 로 학습을 진행함.

```

train_accuracy : 99.107
Validation accuracy after epoch 35: 93.201%
[Epoch 36 / 40]
100% 1309/1309 [05:26<00:00, 4.01it/s]
Epoch 36
train_loss : 0.03776512121643706
train_accuracy : 98.840
Validation accuracy after epoch 36: 93.506%
[Epoch 37 / 40]
100% 1309/1309 [05:25<00:00, 4.02it/s]
Epoch 37
train_loss : 0.030970885841223712
train_accuracy : 98.916
Validation accuracy after epoch 37: 93.965%
[Epoch 38 / 40]
100% 1309/1309 [05:26<00:00, 4.01it/s]
Epoch 38
train_loss : 0.030565600623436894
train_accuracy : 99.012
Validation accuracy after epoch 38: 93.602%
[Epoch 39 / 40]
100% 1309/1309 [05:26<00:00, 4.01it/s]
Epoch 39
train_loss : 0.028393446641552814
train_accuracy : 99.126
Validation accuracy after epoch 39: 93.583%
[Epoch 40 / 40]
100% 1309/1309 [05:27<00:00, 3.99it/s]
Epoch 40
train_loss : 0.02980819861148924
train_accuracy : 99.112
Validation accuracy after epoch 40: 93.182%

```

(train 실행결과)

- 총 40 번의 epoch 를 실행하였으며 train 데이터셋에 대해서는 epoch_39 가 99.126%의 정확도로 높은 정확성을 보여주었음.
- 하지만 검증데이터셋에 대해서는 epoch_37 이 93.965%의 가장 높은 정확도를 보여주었으며, epoch_39 에 비해서 검증 데이터셋에 대한 더 높은 정확도를 보여주었음.
- 목적이 test dataset 에 대해 높은 정확성을 보여주는 것이므로 검증 데이터셋에 대해 견고한 모델이 적합하다고 판단하여 epoch_37.pth 를 model.pth 로 결정하게 됨.
- 첫 번째 에포크의 경우 많은 시간이 소요되었으며 이후 두 번째 에포크 부터는 5 분 30 초가량의 시간이 소요됨을 볼 수 있었음.
- train 중에 런타임 연결이 끊어지는 것을 방지하기 위해 다음의 코드를 실행하였지만,

```

function PreventDisconnection(){
    document.querySelector("colab-toolbar-button#connect").click()

    console.log("클릭이 완료되었습니다.");
}
setInterval(PreventDisconnection, 60 * 10000)

```

epoch_22 쯤에 끊어져서 epoch_22 를 로드하여 epoch_23 부터 다시 train 해야 했었음.

```

[ ] #import torch

[ ] #torch.cuda.is_available()

[ ] #!python run_tta.py
100% 7/7 [00:08<00:00, 1.27s/it]

[ ] !python run.py
100% 7/7 [00:11<00:00, 1.69s/it]

```

내 드라이브 > Term_Project

이름	소유자	마지막으로 수정한 날짜	파일 크기
data	나	2023. 11. 27. 나	-
test_ima...	나	2023. 11. 27. 나	-
utils	나	2023. 11. 27. 나	-
model.py	나	오전 1:55 나	509바이트
result.txt	나	오전 2:42 나	199바이트

- Train 이 모두 진행된 후 run.py 를 실행하고 result.txt 파일이 생성됨을 확인함.