DACON [교통 동작 인식 AI 경진대회]

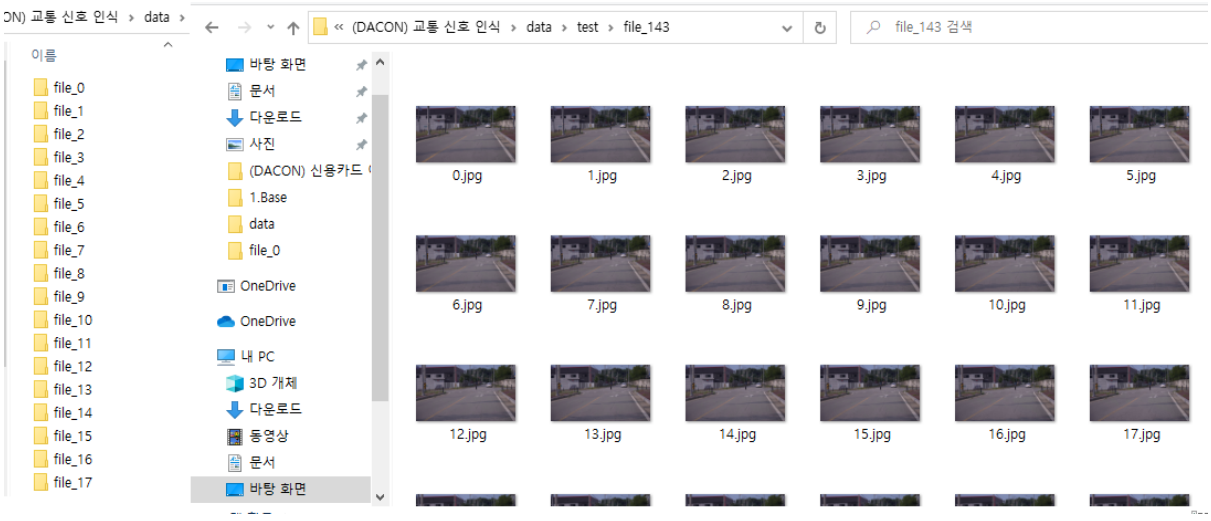
2021 교통 수(手)신호 동작 인식 AI 경진대회

상금 : 총 500만원 340명 마감 안녕하세요 여러분!🙌 교통 수(手)신호 동작 인식 AI 경진대회 에 오신 것을 환영합니다. 교통 수신호 동작 인식 데이터는 자율주행 자동차 등의 기술의 기본이 되는 데이터입니다. 위 데이터를 더욱 다양하게 다량으로 구축할 수 있다면, 전방 센서 인식을 통한 교통 상황의 판단 등

https://dacon.io/competitions/official/235806/overview/description

Dataset

- 교통 수신호를 인식하는 모델을 개발하는 대회입니다.
- JsonFile 형식으로 데이터가 주어지는데, dataset내에는 **학습에 악영향을 미치는 noise data**가 존재합니다. (**Image data training 시 주의!**)



▼ CODE

Import Library

```
import pandas as pd
import numpy as np
from glob import glob

from PIL import Image
import cv2
from tqdm import tqdm

import os
import json
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter("ignore")
import matplotlib.pyplot as plt
```

Load Data

```
data_path = 'data/'
train_path = data_path + 'train'
test_path = data_path + 'test'

train_folders = sorted(glob(train_path + '/*'), key = lambda x : int(x.split('file_')[-1]))
test_folders = sorted(glob(test_path + '/*'), key = lambda x : int(x.split('file_')[-1]))

new_image_directory = data_path + 'new_images/'
new_train_image_directory = new_image_directory + 'train'
new_test_image_directory = new_image_directory + 'test'

action_information = pd.read_csv(data_path + 'action_information.csv')
sample_submission = pd.read_csv(data_path + 'sample_submission.csv')
```

Make Library

```
def make_new_dir(path):
    if os.path.isdir(path) == False:
        os.makedir(path)

make_new_dir(new_image_directory)
make_new_dir(new_train_image_directory)
make_new_dir(new_test_image_directory)
```

Making Bounding Data using Json

- JSON 파일 내에 bounding data가 존재하는데, Image에 적용해 줍니다.

```
train_directories = np.array(sorted(glob(train_path + '/*'), key = lambda x : int(x.split('/')[-1].split('_')[-1])))

for train_directory in tqdm(train_directories, total = len(train_directories)):
    file_name = train_directorys.split('\\')[-1]
    make_new_dir(new_train_image_directory + '\\' + file_name)

    image_paths = sorted(glob(train_directory + '\*jpg'), key = lambda x : int(x.split('\\')[-1].replace('.jpg','')))
    json_path = glob(train_directory + '\*.json')[0]

    js = json.load(open(json_path))
    target = js.get('action')
    target = classes[target]
    bounding_boxes = js.get('sequence').get('bounding_box')
    bounding_boxes = [(float(a), float(b), float(c), float(d)) for a, b, c, d in bounding_boxes]

    for image_path, bounding_box in zip(image_path, bounding_boxes):
        image = Image.open(image_path)
        image = image.crop(bounding_box)
        image = image.resize((224, 224))
        image.save(new_image_directory + image_path.split('/train')[1])

test_directoires = np.array(sorted(glob(test_path + '/*'), key = lambda x : int(x.split('/')[-1].split('_')[-1])))
```

```
for test_directory in tqdm(test_directoires, total = len(test_directoires)):
    file_name = test_directory.split('\\')[-1]
    make_new_dir(new_test_image_directory + '\\' + file_name)

    image_paths = sorted(glob(test_directory + '\*jpg'), key = lambda x : int(x.split('\\')[-1].replace('.jpg','')))
    json_path = glob(test_directory + '\*.json')[0]

    js = json.load(open(json_path))
    target = js.get('action')
    target = classes[target]
    bounding_boxes = js.get('sequence').get('bounding_box')
    bounding_boxes = [(float(a), float(b), float(c), float(d)) for a, b, c, d in bounding_boxes]

    for image_path, bounding_box in zip(image_paths, bounding_boxes):
        image = Image.open(image_path)
        image = image.crop(bounding_box)
        image = image.resize((224, 224))
        image.save(new_test_image_directory + image_path.split('/test')[1])
```

Training Data

```
import pandas as pd
import numpy as np
from glob import glob

from PIL import Image
import cv2
from tqdm import tqdm

import os
import shutil
import json

import torch
import torch.nn as nn
from torchvision import models
from torch.utils.data import DataLoader, Dataset
from torch.optim.lr_scheduler import ReduceLROnPlateau
import albumentations as A
from efficientnet_pytorch import EfficientNet

import os
import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter('ignore')
import matplotlib.pyplot as plt

from sklearn.model_selection import StratifiedKFold

import torch
import random
import torch.backends.cudnn as cudnn

torch.manual_seed(42)
torch.cuda.manual_seed(42)
torch.cuda.manual_seed_all(42)
np.random.seed(42)
cudnn.benchmark = False
cudnn.deterministic = True
random.seed(42)

data_path = 'data'
train_path = data_path + '/train'
train_image_path = data_path + '/new_images/train'
action_information = pd.read_csv(data_path + '/action_information.csv')
```

```
train_folders = sorted(glob(train_path + '/*'), key = lambda x : int(x.split('\\file_')[-1]))
train_img_folders = sorted(glob(train_image_path + '/*'), key = lambda x : int(x.split('\\file_')[-1]))
```

```
imgs = []

for i in train_img_folders:
    train_imgs = sorted(glob(i+"/*.jpg"), key = lambda x : int(x.split('.jpg')[0].split("\\")[-1]))
    for j in train_imgs[25:45]:
        imgs.append(j)
    for j in train_imgs[-45:-25]:
        imgs.append(j)
```

```
answers = []
for train_folder in train_folders :
    json_path = glob(train_folder + '\*.json')[0]
    js = json.load(open(json_path))
    cat = js.get('action')
    answers.append([train_folder.replace(data_path,''),cat])

# 데이터 오류가 존재함
answers = pd.DataFrame(answers, columns = ['train_path','answer'])
answers
answers["answer"][33] = 5
```

```
labels = np.zeros(len(imgs))
for i in tqdm(range(answers.shape[0])):
    for j in range(len(imgs)):
        if answers["train_path"][i] in imgs[j]:
            labels[j] = answers["answer"][i]
```

```
device = torch.device("cuda:0")
dropout_rate = 0.1
class_num = 6
learning_rate = 1e-4
BATCH_SIZE = 16
EPOCHS = 25
MODELS = 'efficientnet-b0'
save_path = f"models/Final_{EPOCHS}"
FOLDS = 5
```

```
imgs = np.array(imgs)
labels = np.array(labels)

from sklearn.utils import shuffle
imgs, labels = shuffle(imgs, labels, random_state=42)

albumentations_transform = A.Compose([
    A.RandomGamma(),
    A.ShiftScaleRotate(),
    A.GaussianBlur(),
    A.GaussNoise()
])
```

Compare Albumentations and ImageDataGenerator

Explore and run machine learning code with Kaggle Notebooks | Using data from multiple data sources

https://www.kaggle.com/code/kalelpark/compare-albumentations-and-imagedatagenerator

Custom Dataset

```
class CustomDataset(Dataset):
    def __init__(self, imgs, labels = None, tranformer = None, mode = "train"):
```

```
        self.imgs = imgs
        self.transformer = transformer
        self.mode = mode
        if self.mode == "train":
            self.labels = labels

    def __len__(self):
        return len(self.imgs)

    def __getitem__(self, i):
        img = cv2.imread(self.imgs[i]).astype(np.float32) / 255
        img = cv2.resize(img, dsize = (224, 224))
        if self.mode == "train":
            if self.transformer != None:
                img = self.transformer(image = img)
                img = np.transpose(img["image"], (2, 0, 1))
            else:
                img = np.transpose(img["image"], (2, 0, 1))
            return {
                "img" : torch.tensor(img, dtype = torch.float32)
                "label" : torch.tensor(self.labels[i], dtype = torch.long)
            }
        else:
            img = np.transpose(img, (2, 0, 1))
            return {
                "img" : torch.tensor(img, dtype = torch.float32)
            }
```

Training Loop

```
def train_step(batch_item, epoch, batch, training):
    img = batch_item['img'].to(device)
    label = batch_item['label'].to(device)
    if training is True:
        model.train()
        optimizer.zero_grad()
        with torch.cuda.amp.autocast():
            output = model(img)
            loss = criterion(output, label)
        loss.backward()
        optimizer.step()

        return loss
    else:
        model.eval()
        with torch.no_grad():
            output = model(img)
            loss = criterion(output, label)

        return loss
```

```
skf = StratifiedKFold(n_splits = FOLDS, random_state = 42, shuffle = True)

n_iter = 0

for train_idx, val_idx in skf.split(imgs, labels):
    model = EfficientNet.from_pretrained(MODELS, num_classes=class_num, advprop = True)
    model._dropout.p = dropout_rate
    model = model.to(device)

    optimizer = torch.optim.AdamW(model.parameters(), lr = learning_rate)
    criterion = nn.CrossEntropyLoss()
    scheduler = ReduceLROnPlateau(optimizer, 'min', patience = 5)

    n_iter += 1

    train_dataset = CustomDataset(imgs[train_idx], labels[train_idx], transformer= albumentations_transform)
    val_dataset = CustomDataset(imgs[val_idx], labels[val_idx], transformer= albumentations_transform)

    train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle= True)
    val_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle= True)

    loss_plot, val_loss_plot = [], []
```

```
    for epoch in range(EPOCHS):
        total_loss, total_val_loss = 0, 0

        tqdm_dataset = tqdm(enumerate(train_dataloader))
        training = True
        for batch, batch_item in tqdm_dataset:
            batch_loss = train_step(batch_item, epoch, batch, training)
            total_loss += batch_loss

            tqdm_dataset.set_postfix({
                'Epoch' : epoch + 1,
                'Loss' : '{:06f}'.format(batch_loss.item()),
                'Total Loss' : '{:06f}'.format(total_loss / (batch + 1))
            })
        loss_plot.append(total_loss / (batch + 1))

        tqdm_dataset = tqdm(enumerate(val_dataloader))
        training = False
        for batch, batch_item in tqdm_dataset:
                batch_loss = train_step(batch_item, epoch, batch, training)
                total_val_loss += batch_loss

                tqdm_dataset.set_postfix({
                    'Epoch': epoch + 1,
                    'Val Loss': '{:06f}'.format(batch_loss.item()),
                    'Total Val Loss' : '{:06f}'.format(total_val_loss/(batch+1))
                })
        val_loss_plot.append(total_val_loss/(batch+1))
        scheduler.step(total_val_loss/(batch+1))
        if min(val_loss_plot) == val_loss_plot[-1]:
            torch.save(model, save_path + f"_{n_iter}.pt")
            print("## Model Save")
    break
```

save model

```
MODELS = 'efficientnet-b0'
save_path = f"models/Final"
device = torch.device("cuda:0")
class_num = 6
FOLDS = 1

test_folders = sorted(glob(test_path + '/*'), key = lambda x : int(x.split('\\file_')[-1]))
test_img_folders = sorted(glob(test_img_path + '/*'), key = lambda x : int(x.split('\\file_')[-1]))

imgs = []

for i in test_img_folders:
    test_imgs = sorted(glob(i+"/*.jpg"), key = lambda x : int(x.split('.jpg')[0].split("\\")[-1]))
    for j in test_imgs[30:45]:
        imgs.append(j)

model = EfficientNet.from_pretrained(MODELS, num_classes=class_num, advprop=True)
model = model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()
```

```
def predict(dataset):
    model.eval()
    result = []
    for batch_item in dataset:
        img = batch_item['img'].to(device)
        with torch.no_grad():
            output = model(img)
        output = output.cpu().numpy()
        result.extend(output)

    return result

def softmax(x):
    f_x = np.exp(x) / np.sum(np.exp(x))
    return f_x
```

```
answers = []
for test_folder in test_folders :
    json_path = glob(test_folder + '/*.json')[0]
    js = json.load(open(json_path))
    answers.append([test_folder.replace(data_path,'')])

answers = pd.DataFrame(answers, columns = ['test_path'])
```

```
submission = pd.read_csv(data_path + '/sample_submission.csv')
sub_ALL = []
sub_ALL_list = []

for i in tqdm(range(FOLDS)):

    model = torch.load(save_path + f"_{EPOCHS}_1.pt", map_location=device)

    test_dataset = CustomDataset(imgs=imgs, labels=None, mode='test')
    test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=BATCH_SIZE)

    pred = predict(test_dataloader)

    a = []
    for i in range(len(pred)):
        a.append(softmax(pred[i]))
    a = np.array(a)

    sub_ALL = []

    for i in range(answers.shape[0]):
        sub = 0
        num_sub = 0
        for j in range(len(imgs)):
            if answers["test_path"][i] in imgs[j]:
                num_sub = num_sub + 1
                sub = sub + a[j]
        sub = sub / num_sub
        sub_ALL.append(sub)

    sub_ALL_list.append(sub_ALL)

sub_ALL_list = np.array(sub_ALL_list)
sub_ALL_list = np.mean(sub_ALL_list, axis=0)
submission.iloc[:,1:] = sub_ALL_list

SAVE_CSV_NAME = f'submit/Final.csv'
submission.to_csv(SAVE_CSV_NAME, index=False)
```

What I learned

- 1개의 예측값을 구할 때, 여러 데이터를 앙상블 형식으로 구하기

  EX> 1번의 값을 예측할 때, 사용되는 데이터가 여러가지 있으면, 데이터들을 모두 사용하여, 평균화하여, 예측하기

- 이미지 데이터 내에도 노이즈 값이 존재할 수 있으므로. 파일로 데이터 파악해보기!