3. Pretrained Model

참고자료

- https://blog.naver.com/beyondlegend/222644092397
- https://blog.naver.com/charzim0611/222948860899
- https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs210925
- https://www.tensorflow.org/tutorials/images/transfer_learning?hl=ko

In [ ]:
```
from google.colab import drive
drive.mount("/content/gdrive/")
```

```
Drive already mounted at /content/gdrive/; to attempt to forcibly remount, call drive.mount("/content/gdrive/",
force_remount=True).
```

In [ ]:
```
# 기본
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import os
import shutil
import glob
import cv2
import random

# 경고
import warnings
warnings.filterwarnings('ignore')

# 그래프 설정
plt.rcParams['figure.figsize'] = 20, 10
plt.rcParams['axes.unicode_minus'] = False

# tensor
import tensorflow as tf

# Model, Layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.applications import EfficientNetB6
from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications import Xception
from tensorflow.keras.utils import Sequence
from tensorflow.keras.layers import Flatten, Dropout, LeakyReLU, Activation, Dense, GlobalAveragePooling2D, Bat

# optimizer
from keras.optimizers import Adagrad, Adadelta, Adam

# 모델 시각화
from tensorflow.keras.utils import plot_model

# Call backs
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler, ReduceLROnPlateau

# GPU check
from tensorflow.python.client import device_lib

# torch
import torch
import torch.optim as optim
import torchvision.datasets   as datasets
from torch.utils.data import DataLoader, Dataset
import torchvision.transforms as transforms

# 컴퓨터에 있는 GPU 정보들을 가져온다.
# gpu가 있다면...

os.environ["CUDA_VISIBLE_DEVICES"]="0"
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        tf.config.experimental.set_memory_growth(gpus[0], True)
    except RuntimeError as e:
        print(e)
```

In [ ]:
```
!nvidia-smi
```

```
Sun Apr 16 11:14:36 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   39C    P8     9W /  70W |      3MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

In [ ]:
```
# 기본경로
PATH = '/content/gdrive/MyDrive/dataset/'
```

In [ ]:
```
# image size & bacth
image_size = 256            # 리사이징할 이미지 사이즈
batch = image_size // 16    # 이미자시이즈 기준 배치사이즈 ex. 32 // 16 == 2, 512 // 16 == 32
channel = 3                 # 인풋 채널 수
```

In [ ]:
```
# 경로 생성 및 확인
train_dir = os.path.join(PATH, 'train')
test_dir = os.path.join(PATH, 'test')
train_dir, test_dir
```

Out[ ]:
```
('/content/gdrive/MyDrive/dataset/train',
 '/content/gdrive/MyDrive/dataset/test')
```

In [ ]:
```
# Train 데이터 분할
train_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.1,
    subset = "training",
    label_mode='int',
    seed=123,
    shuffle=True,
    image_size = (image_size, image_size),
    batch_size = batch)
```

```
Found 777 files belonging to 11 classes.
Using 700 files for training.
```

In [ ]:
```
# Val 데이터 분할
val_ds = tf.keras.utils.image_dataset_from_directory(
    train_dir,
    validation_split=0.1,
    subset="validation",
    label_mode='int',
    seed=123,
    shuffle=True,
    image_size = (image_size, image_size),
    batch_size = batch)
```

```
Found 777 files belonging to 11 classes.
Using 77 files for validation.
```

In [ ]:
```
# Test 데이터 로드
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    label_mode='int',
    image_size = (image_size, image_size),
    batch_size = batch)
```

```
Found 330 files belonging to 11 classes.
```

Pretrain

사전학습모델의 경우 이전에 시도한 다양한 경험을 통해
EfficientNetB6를 1000 epochs 돌린 결과 합치지 않는 것이 좋을 것이라고 판단하여
라벨 명만 수정하고 11개의 클래스로 진행했습니다.
- 10-1 >> 0

- 10-2 >> 10
라벨 클래스 11개

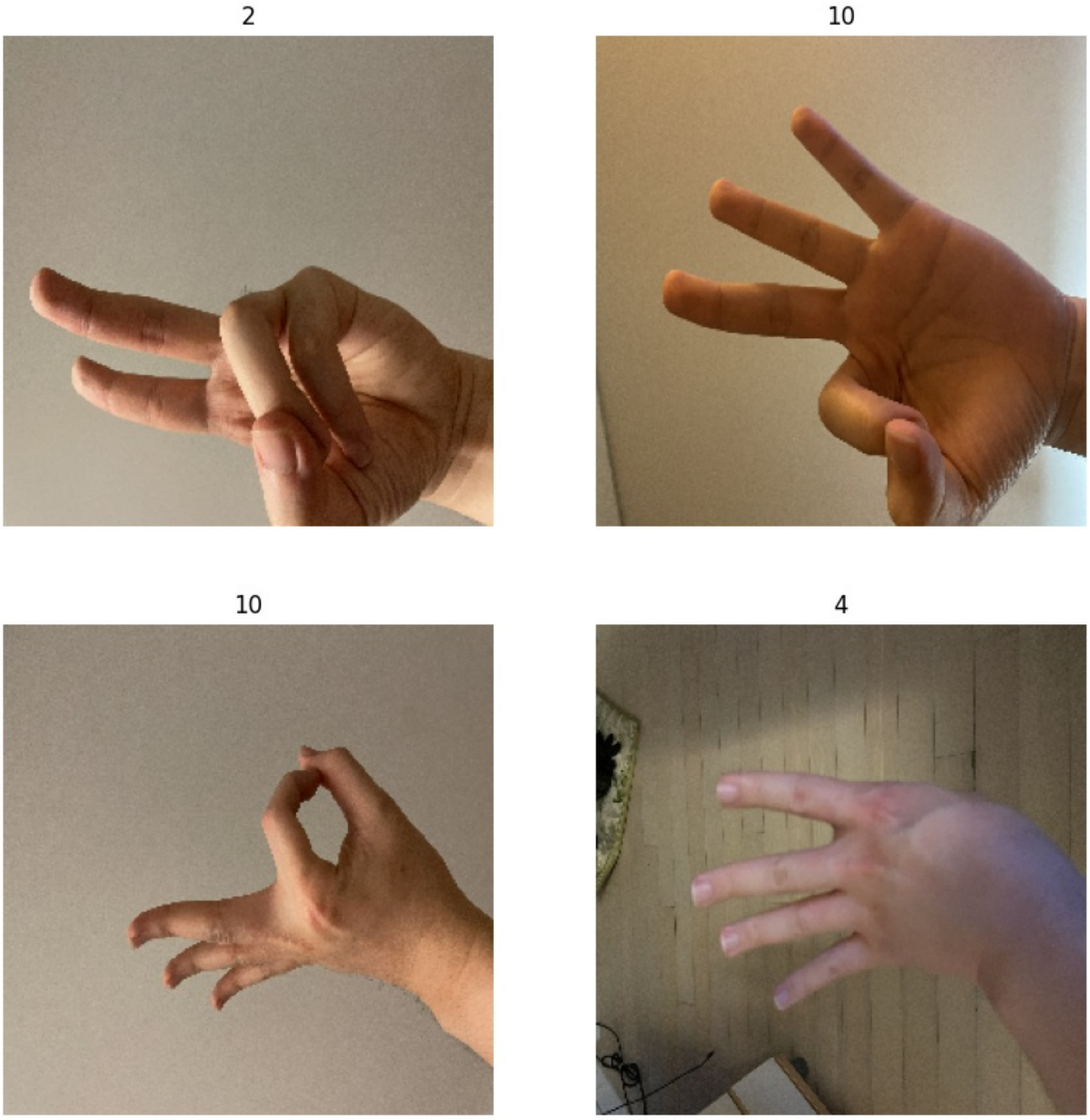
In [ ]:
```
# 라벨
class_names = train_ds.class_names
print(class_names)
```

```
['0', '1', '10', '2', '3', '4', '5', '6', '7', '8', '9']
```

In [ ]:
```
# 라벨 수
num_classes = len(class_names)
num_classes
```

Out[ ]:
```
11
```

In [ ]:
```
# train 데이터 확인
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(4):
    ax = plt.subplot(2, 2, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



In [ ]:
```
# 프리페치
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

In [ ]:
```
# 전이학습
base_model = Xception(weights = 'imagenet', include_top = False, input_shape = (image_size, image_size, channel
# base_model = InceptionResNetV2(weights = 'imagenet', include_top = False, input_shape = (image_size, image_si
```

```
# base_model = EfficientNetB6(weights = 'imagenet', include_top = False, input_shape = (image_size, image_size,
# base_model = tf.keras.applications.MobileNetV2(weights='imagenet', include_top=False, input_shape = (image_si
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xception/xception_weights_tf
_dim_ordering_tf_kernels_notop.h5
83683744/83683744 [==============================] - 3s 0us/step
```

In [ ]:
```
# Preprocessing
preprocess_input = tf.keras.applications.xception.preprocess_input
```

In [ ]:
```
# 모델 학습 여부
base_model.trainable = True
print(len(base_model.layers)) # 베이스모델의 층 갯수 확인
```

```
132
```

In [ ]:
```
# 모델에서 n층까진 w값을 조정하지 않고, 그 다음부터 w값을조정
for layer in base_model.layers[:100]:
    layer.trainable = False

# 학습가능한 변수층 수
len(base_model.trainable_variables)
```

Out[ ]:
```
39
```

In [ ]:
```
# 특징 추출
image_batch, label_batch = next(iter(train_ds))
feature_batch = base_model(image_batch)
print(feature_batch.shape)

global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)

prediction_layer = tf.keras.layers.Dense(num_classes, activation='softmax')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

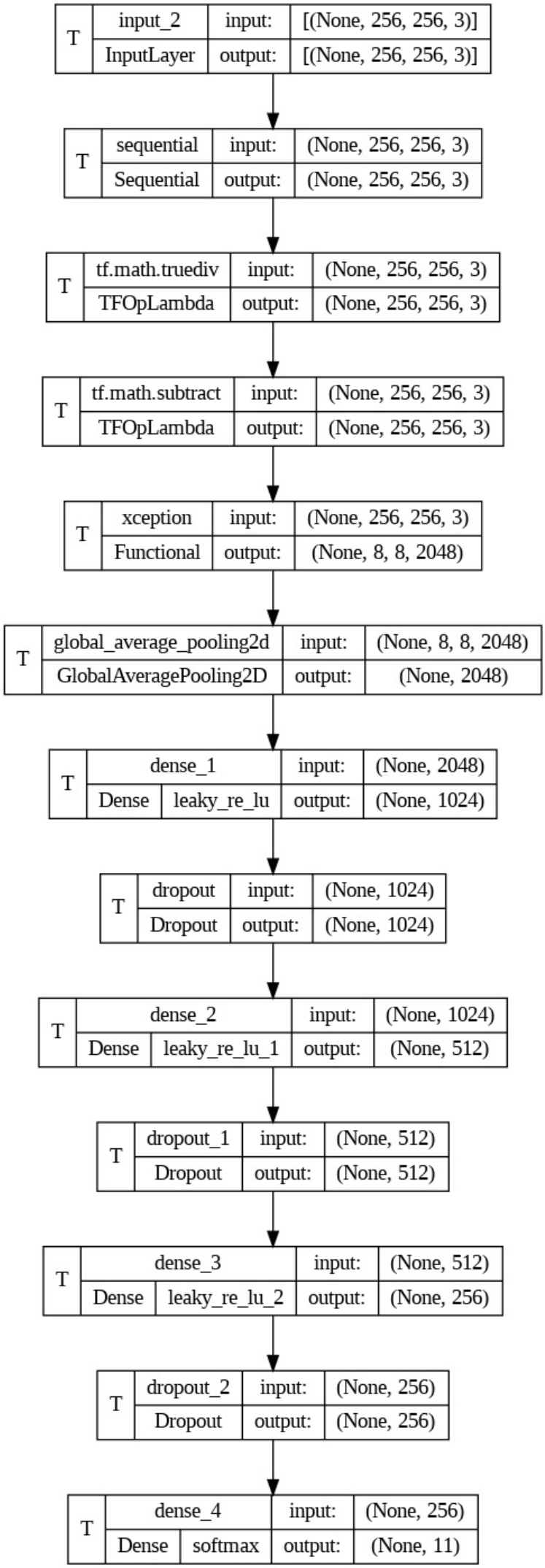
```
(16, 8, 8, 2048)
(16, 2048)
(16, 11)
```

In [ ]:
```
# Pretrained Model
inp = tf.keras.Input(shape=(image_size, image_size, channel))

data_aug = Sequential([tf.keras.layers.RandomFlip("horizontal_and_vertical"),
                       tf.keras.layers.RandomRotation(0.2),
                       tf.keras.layers.RandomZoom(0.2)
                       ])(inp)

x = preprocess_input(data_aug)

x = base_model(x)
x = global_average_layer(x)

x = Dense(image_size * 4, activation =  LeakyReLU(alpha=0.2))(x)
x = Dropout(0.5)(x)
x = Dense(image_size * 2, activation =  LeakyReLU(alpha=0.2))(x)
x = Dropout(0.5)(x)
x = Dense(image_size, activation =  LeakyReLU(alpha=0.2))(x)
x = Dropout(0.5)(x)

outp = Dense(num_classes, activation='softmax')(x)
model = tf.keras.Model(inp, outp)
```

In [ ]:
```
# optimizer = tf.keras.optimizers.SGD(momentum = True, nesterov = True)
# optimizer = tf.keras.optimizers.RMSprop()
optimizer = tf.keras.optimizers.Adam(beta_1 = 0.9, beta_2 = 0.999)

model.compile(loss = "sparse_categorical_crossentropy",
              optimizer = optimizer,
              metrics = ['accuracy'])
```

In [ ]:
```
# 모델 플롯
plot_model(model, show_layer_names= True, show_trainable = True, show_layer_activations= True, show_shapes= Tru
```

Out[ ]:


T
input_2
InputLayer
input:
output:
[(None, 256, 256, 3)]
[(None, 256, 256, 3)]
T
sequential
Sequential
input:
output:
(None, 256, 256, 3)
(None, 256, 256, 3)
T
tf.math.truediv
TFOpLambda
input:
output:
(None, 256, 256, 3)
(None, 256, 256, 3)
T
tf.math.subtract
TFOpLambda
input:
output:
(None, 256, 256, 3)
(None, 256, 256, 3)
T
xception
Functional
input:
output:
(None, 256, 256, 3)
(None, 8, 8, 2048)
T
global_average_pooling2d
GlobalAveragePooling2D
input:
output:
(None, 8, 8, 2048)
(None, 2048)
T
dense_1
Dense
leaky_re_lu
input:
output:
(None, 2048)
(None, 1024)
T
dropout
Dropout
input:
output:
(None, 1024)
(None, 1024)
T
dense_2
Dense
leaky_re_lu_1
input:
output:
(None, 1024)
(None, 512)
T
dropout_1
Dropout
input:
output:
(None, 512)
(None, 512)
T
dense_3
Dense
leaky_re_lu_2
input:
output:
(None, 512)
(None, 256)
T
dropout_2
Dropout
input:
output:
(None, 256)
(None, 256)
T
dense_4
Dense
softmax
input:
output:
(None, 256)
(None, 11)

In [ ]:
```
# 모델 구조 확인
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 256, 256, 3)]     0

 sequential (Sequential)     (None, 256, 256, 3)       0

 tf.math.truediv (TFOpLambda  (None, 256, 256, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 256, 256, 3)      0
 a)

 xception (Functional)       (None, 8, 8, 2048)        20861480

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 dense_1 (Dense)             (None, 1024)              2098176

 dropout (Dropout)           (None, 1024)              0

 dense_2 (Dense)             (None, 512)               524800

 dropout_1 (Dropout)         (None, 512)               0

 dense_3 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_4 (Dense)             (None, 11)                2827

=================================================================
Total params: 23,618,611
Trainable params: 12,235,475
Non-trainable params: 11,383,136
_________________________________________________________________
```

In [ ]:
```
# 학습모델을 저장할 경로
path = PATH + '/models/pretrained/'
```

In [ ]:
```
# 폴더 생성 및 체크
if os.path.isdir(path) :
  pass
else :
  os.makedirs(os.path.join(path))
```

In [ ]:
```
# parameters 세팅
epoch_n = 400
```

In [ ]:
```
# 스케줄러
def lr_step_decay(epoch, lr):
    drop_rate = 0.899                       # 학습률 드랍 비율
    epochs_drop = round(math.sqrt(epoch_n))
    initial_lr = 0.0001                      # 사전학습시 학습률 낮게
    lr = initial_lr * math.pow(drop_rate, math.floor(epoch / epochs_drop))
    return lr

scheduler = LearningRateScheduler(lr_step_decay, verbose=1)
```

In [ ]:
```
# 조기 종료
early_stop = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',                          # 모니터링 지표표
    min_delta = 0.001,
    patience = epoch_n // 2,       # 기준횟수
    restore_best_weights = True,               # 최상 weights 복원
    verbose=1)
```

In [ ]:
```
# ReduceLROnPlateau
reduce_lr = ReduceLROnPlateau(monitor = 'val_accuracy', patience = 1,verbose = 1)
```

In [ ]:
```
# histroy 초기화
model_history = []
```

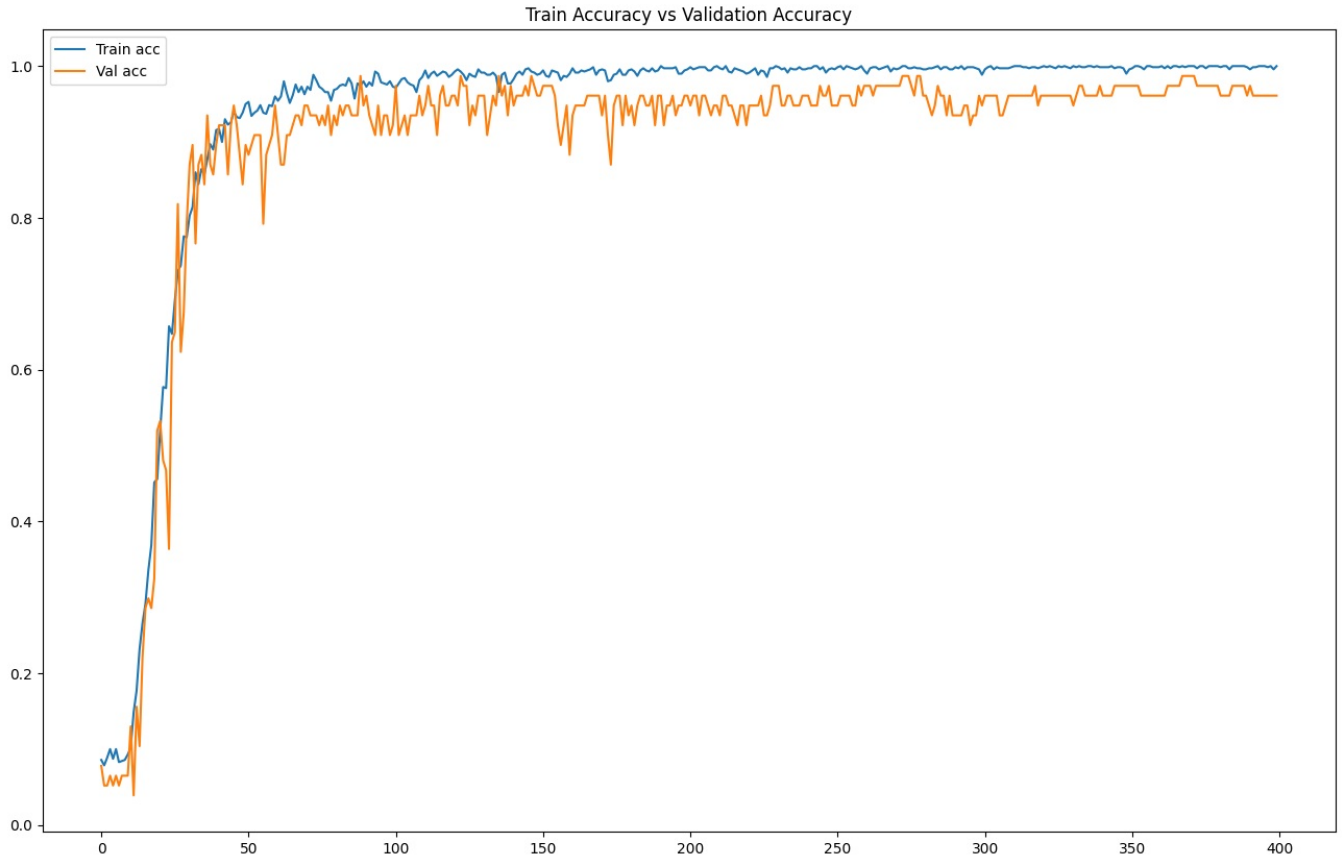
In [ ]:
```
# model_fit
history = model.fit(train_ds, validation_data = val_ds,
                    workers = 8,
                    epochs = epoch_n,
```

```
                    callbacks = [scheduler, reduce_lr, early_stop]
                    )

model_history.append(history)
```

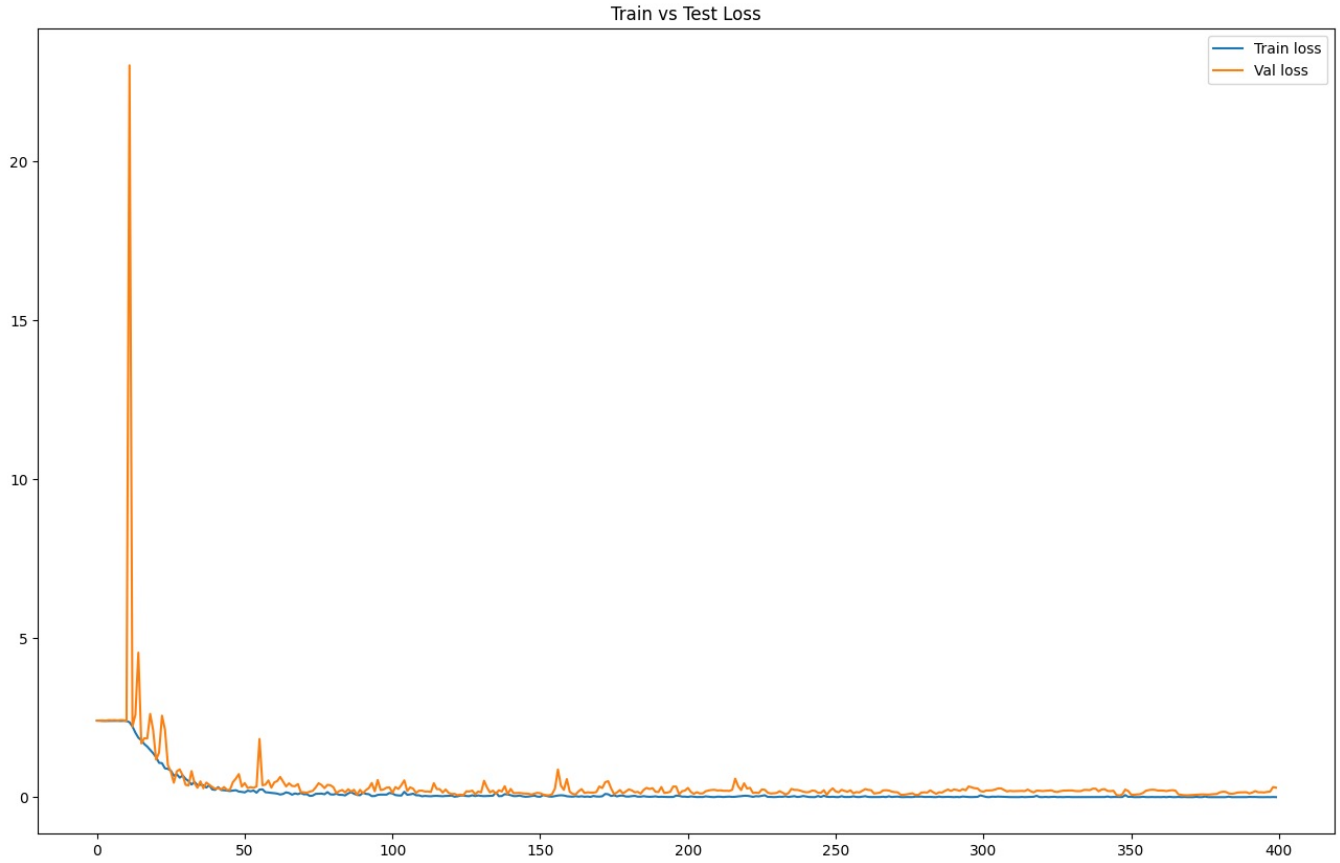
[학습 로그 부분 삭제]

In [ ]:
```
# 리스트를 추출한다.
loss_list = history.history['loss']
accuracy_list = history.history['accuracy']
val_loss_list = history.history['val_loss']
val_accuracy_list = history.history['val_accuracy']
```

In [ ]:
```
# 정확도 그래프
plt.figure(figsize= (16, 10))
plt.title('Train Accuracy vs Validation Accuracy')
plt.plot(accuracy_list, label='Train acc')
plt.plot(val_accuracy_list, label= 'Val acc')
plt.legend()
plt.show()
```



In [ ]:
```
# 손실률
plt.figure(figsize= (16, 10))
plt.title('Train vs Test Loss')
plt.plot(loss_list, label='Train loss')
plt.plot(val_loss_list, label='Val loss')
plt.legend()
plt.show()
```



In [ ]:
```
# 테스트셋 적용
model.evaluate(test_ds)
```

```
21/21 [==============================] - 89s 3s/step - loss: 1.0950 - accuracy: 0.8939
```

Out[ ]:
```
[1.0949515104293823, 0.8939393758773804]
```

In [ ]:
```
//정지
```

```
  File "<ipython-input-35-af96ed8a25f1>", line 1
    //정지
    ^
SyntaxError: invalid syntax
```

In [ ]:
```
# 검증 셋 추가학습
less_epochs = 50
total_epochs =  epoch_n + less_epochs

history_fine1 = model.fit(val_ds,
                          epochs = total_epochs,
                          initial_epoch = history.epoch[-1],
                          )
```

In [ ]:
```
loss_list += history_fine1.history['loss']
accuracy_list += history_fine1.history['accuracy']
```

In [ ]:
```
# 추가학습 후 테스트
model.evaluate(test_ds)
```

```
21/21 [==============================] - 15s 494ms/step - loss: 0.3939 - accuracy: 0.9576
```

Out[ ]:
```
[0.39391404390335083, 0.9575757384300232]
```

In [ ]:

In [ ]:

In [ ]:

이하 휴지통 (초기 모델 및 데이터 전처리, not TFDS)

In [ ]:
```
# # 이미지 처리
# # heif 확장자 처리
# !pip install pillow-heif
```

```
# from PIL import Image
# from pillow_heif import register_heif_opener
# register_heif_opener()
```

In [ ]:
```
# # 하위 폴더 들의 이름을 가져온다(결과 데이터)
# categories = list(os.walk(PATH +'train'))[0][1]
```

In [ ]:
```
# categories
```

In [ ]:
```
# num_classes = len(categories)
# num_classes
```

In [ ]:
```
# # image size
# image_size = 128
# channel = 3
```

In [ ]:
```
# # Imgdatagen
# idg = ImageDataGenerator(horizontal_flip = True,
#                          vertical_flip = True,
#                          rotation_range = 0.2,
#                          height_shift_range = 0.2,
#                          width_shift_range = 0.2,
#                         )
```

In [ ]:
```
# 매우 느리고 무거움..
# X = []
# y = []
# averArr = np.zeros(shape=(image_size, image_size, channel))

# for idx, category in enumerate(categories):
#     # one-hot
#     label = [0 for i in range(num_classes)]
#     label[idx] = 1

#     image_dir = PATH + '/train/' + category
#     files = glob.glob(image_dir + '/*')

#     for i, j in enumerate(files):
#         # 원본 저장
#         img_org = Image.open(j)
#         img1 = img_org.resize((image_size, image_size))
#         data = np.asarray(img1) / 255
#         averArr += np.asarray(img_org.resize((image_size, image_size)))

#         X.append(data)
#         y.append(label)

#         # Augmentation
#         for k in range(0,2):
#             img2 = np.expand_dims(img1, axis = 0)
#             datagen = idg.flow(img2)
#             aug_img = next(datagen)
#             img2 = np.squeeze(aug_img)
#             data = np.asarray(img2) / 255

#             X.append(data)
#             y.append(label)

#         if i == 199:
#             averArr = (np.trunc((averArr / 200))) / 255
#             X.append(averArr)
#             averArr = np.zeros(shape=(image_size, image_size, channel))

#     print(category, 'done')

# X = np.array(X)
# y = np.array(y)
```

In [ ]:
```
# X.shape, y.shape
```

In [ ]:
```
# len(X)
```

In [ ]:
```
# # X 샘플확인
# plt.figure(figsize=(16,10))
# for g in range(len(X[:16])):
#     plt.subplot(4,4,g+1)
#     plt.imshow(X[g])
#     plt.axis('off')
```

In [ ]:
```
# # 전이학습
```

```
# base_model = Xception(weights = 'imagenet', include_top = False, input_shape = (image_size, image_size, channe
# # base_model = InceptionResNetV2(weights = 'imagenet', include_top = False, input_shape = (image_size, image_s
# # base_model = EfficientNetV2S(weights = 'imagenet', include_top = False, input_shape = (image_size, image_si
```

In [ ]:
```
# # 모델 학습 여부
# base_model.trainable = True
# print(len(base_model.layers)) # 베이스모델의 층 갯수 확인
```

In [ ]:
```
# # 모델에서 n층까진 w값을 조정하지 않고, 그 다음부터 w값을조정
# for layer in base_model.layers[:100]:
#     layer.trainable = False

# # 학습가능한 변수층 수
# len(base_model.trainable_variables)
```

In [ ]:
```
# model = Sequential()
# model.add(base_model)

# model.add(Flatten())

# model.add(Dense(image_size * 2, activation =  LeakyReLU(alpha=0.2)))
# model.add(Dropout(0.5))

# model.add(Dense(image_size * 4, activation =  LeakyReLU(alpha=0.2)))
# model.add(Dropout(0.5))

# model.add(Dense(image_size * 2, activation =  LeakyReLU(alpha=0.2)))
# model.add(Dropout(0.5))

# model.add(Dense(num_classes, activation='softmax'))

# # optimizer = tf.keras.optimizers.SGD(lr, momentum = True, nesterov = True)
# # optimizer = tf.keras.optimizers.RMSprop(lr, rho = 0.9)
# optimizer = tf.keras.optimizers.Adam(beta_1 = 0.9, beta_2 = 0.999)

# model.compile(loss = "categorical_crossentropy",
#               optimizer = optimizer,
#               metrics = ['accuracy'])

# model.summary()
```

In [ ]:
```
# # 모델 플롯
# plot_model(model, show_layer_names= True, show_trainable = True, show_layer_activations= True, show_shapes= T
```

In [ ]:
```
# # 학습모델을 저장할 경로
# path = PATH + '/models/pretrained/'
```

In [ ]:
```
# # 폴더 생성 및 체크
# if os.path.isdir(path) :
#   pass
# else :
#   os.makedirs(os.path.join(path))
```

In [ ]:
```
# # 데이터로더
# class Dataloader(Sequence):

#     def __init__(self, x_set, y_set, batch_size, shuffle=False):
#         self.x, self.y = x_set, y_set
#         self.batch_size = batch_size
#         self.shuffle = shuffle
#         self.on_epoch_end()

#     def __len__(self):
#         return math.ceil(len(self.x) / self.batch_size)


#     def __getitem__(self, idx):
#         indices = self.indices[idx * self.batch_size : (idx + 1) * self.batch_size]

#         batch_x = [self.x[i] for i in indices]
#         batch_y = [self.y[i] for i in indices]

#         return np.array(batch_x), np.array(batch_y)

#     def on_epoch_end(self):
#         self.indices = np.arange(len(self.x))
#         if self.shuffle == True:
#             np.random.shuffle(self.indices)
```

In [ ]:
```
# 모델_체크포인트
# path1 = path + '/{epoch}-{val_loss}.h5'
```

```
# path2 = path + '/best_model.h5'

# # 저장 콜백
# call1 = ModelCheckpoint(filepath = path1, monitor = 'val_loss', save_best_only = True) # 용량주의
# call2 = ModelCheckpoint(filepath = path2, monitor = 'val_loss', save_best_only = True) # 용량주의
```

In [ ]:
```
# # parameters 세팅
# epoch_n = 1000
# batch = 32
```

In [ ]:
```
# # 스케쥴러
# def lr_step_decay(epoch, lr):
#     drop_rate = 0.9876                          # 학습률 드랍 비율
#     epochs_drop = round(math.sqrt(epoch_n))
#     initial_lr = 0.001
#     lr = initial_lr * math.pow(drop_rate, math.floor(epoch / epochs_drop))
#     return lr

# scheduler = LearningRateScheduler(lr_step_decay, verbose=1)
```

In [ ]:
```
# # 조기 종료
# early_stop = tf.keras.callbacks.EarlyStopping(
#       monitor='val_loss',                           # 모니터링 지표표
#     min_delta = 0.001,
#     patience = round(math.sqrt(epoch_n)),       # 기준횟수
#     restore_best_weights = True,                # 최상 weights 복원
#     verbose=1)
```

In [ ]:
```
# # Train, Val
# X_train, X_test, y_train, y_test = train_test_split(X, y,
#                                                     random_state = 1,
#                                                     shuffle = True,
#                                                     test_size = 0.2)

# X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

In [ ]:
```
# model_history = []

# # Data loader
# train_loader = Dataloader(X_train, y_train, batch, shuffle = True)
# test_loader = Dataloader(X_test, y_test, batch)

# history = model.fit(train_loader, validation_data = test_loader,
#                     epochs = epoch_n,
#                     workers = 8,
#                    callbacks= [scheduler, early_stop]
#                    )

# model_history.append(history)
```

In [ ]:
```
# len(model_history)
```

In [ ]:
```
# # Val 정확도 확인
# a1 = model.evaluate(X_test, y_test)
# print(f' 테스트 손실률 : {a1[0]}')
# print(f' 테스트 정확도 : {a1[1]}')
```

In [ ]:
```
# # 리스트를 추출
# loss_list = history.history['loss']
# accuracy_list = history.history['accuracy']
# val_loss_list = history.history['val_loss']
# val_accuracy_list = history.history['val_accuracy']
```

In [ ]:
```
# # Acc
# plt.figure(figsize= (16, 10))
# plt.title('Train Accuracy vs Validation Accuracy')
# plt.plot(accuracy_list, label='Train acc')
# plt.plot(val_accuracy_list, label= 'Val acc')
# plt.legend()
# plt.show()
```

In [ ]:
```
# # Loss
# plt.figure(figsize= (16, 10))
# plt.title('Train vs Test Loss')
# plt.plot(loss_list, label='Train loss')
# plt.plot(val_loss_list, label='Val loss')
# plt.legend()
# plt.show()
```

In [ ]:
```
# len(y_train), len(y_test)
```

In [ ]:
```
# # 모델 세이브
# model.save(PATH + 'models/pretrained/best_model.h5')
```

In [ ]:
```

```
