2. Scratch to CNN

참고자료

- https://blog.naver.com/beyondlegend/222644092397
- https://blog.naver.com/charzim0611/222948860899
- https://www.tensorflow.org/tutorials/images/classification?hl=ko
- https://www.tensorflow.org/tutorials/images/data_augmentation?hl=ko
- https://limitsinx.tistory.com/48
- https://github.com/kwotsin/TensorFlow-Xception/blob/master/xception.py

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 seaborn as sns
import math
import os
import datetime
import shutil
import glob
import pathlib
import cv2

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

# 출력한 내용 청소
from IPython.display import clear_output

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

# 랜덤 모듈
import random

# 딥러닝 라이브러리
import tensorflow as tf
import tensorflow.compat.v1 as tf1
from tensorflow.keras import Model

# 신경망 모델을 관리하는 객체
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import Sequence
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adagrad, Adadelta, Adam, AdamW

# 모델 시각화
from tensorflow.keras.utils import plot_model

# 선형 회귀 레이어
from tensorflow.keras.layers import Dense, Activation

# Callback
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, LearningRateScheduler

# GPU 사용 확인
from tensorflow.python.client import device_lib
from tensorflow.keras.layers import Flatten, Dropout, LeakyReLU
from tensorflow.keras.layers import Conv2D, Conv1D
from tensorflow.keras.layers import MaxPool2D, MaxPool1D
from tensorflow.keras.layers import Concatenate, Input, Add
from tensorflow.keras.layers import BatchNormalization, Rescaling, RandomRotation, RandomFlip
from tensorflow.keras.layers import GlobalAveragePooling2D, GlobalMaxPooling2D

from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import train_test_split

# CUDA
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
```

```
Sat Apr 15 08:21:50 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   66C    P8    10W /  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(h,w)
batch = image_size // 16    # 이미자시이즈 기준 배치사이즈 ex. 32 // 16 == 2, 512 // 16 == 32
channel = 3                 # 인풋 채널 수수
```

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 로드
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 10 classes.
Using 700 files for training.
```

In [ ]:
```
# Val 데이터 로드
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 10 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 10 classes.
```

Scratch to CNN의 경우

10-1 , 10-2 통합한 데이터로 실행했습니다.

라벨 클래스 10개

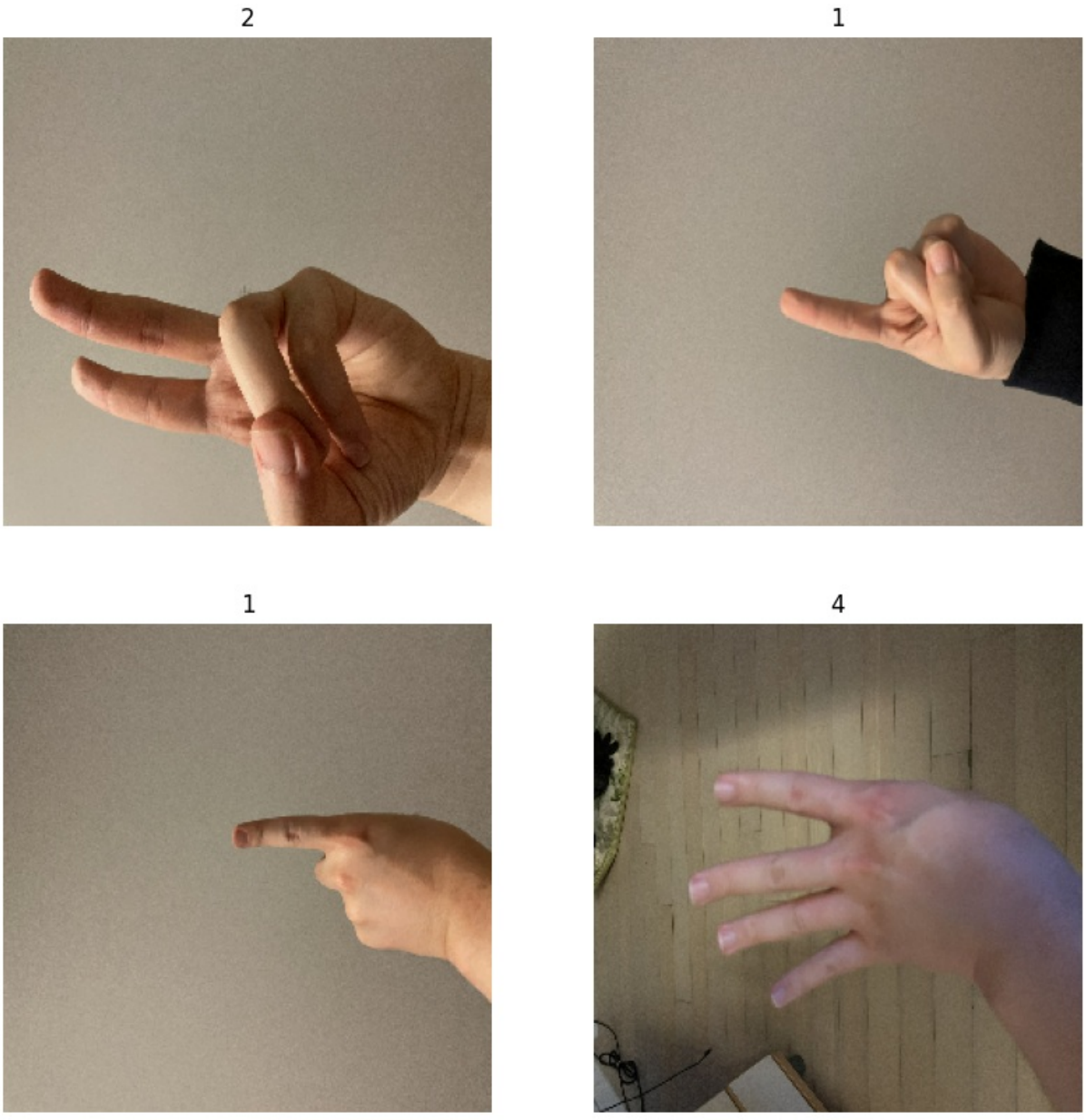
In [ ]:
```
# 라벨
class_names = train_ds.class_names
print(class_names)
```

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

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

Out[ ]:
```
10
```

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")
```



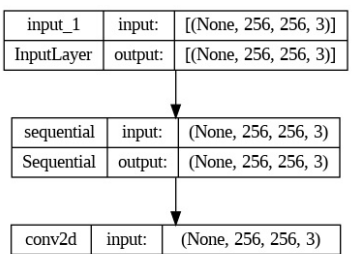
In [ ]:
```
# 프리페치
AUTOTUNE = tf.data.AUTOTUNE

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

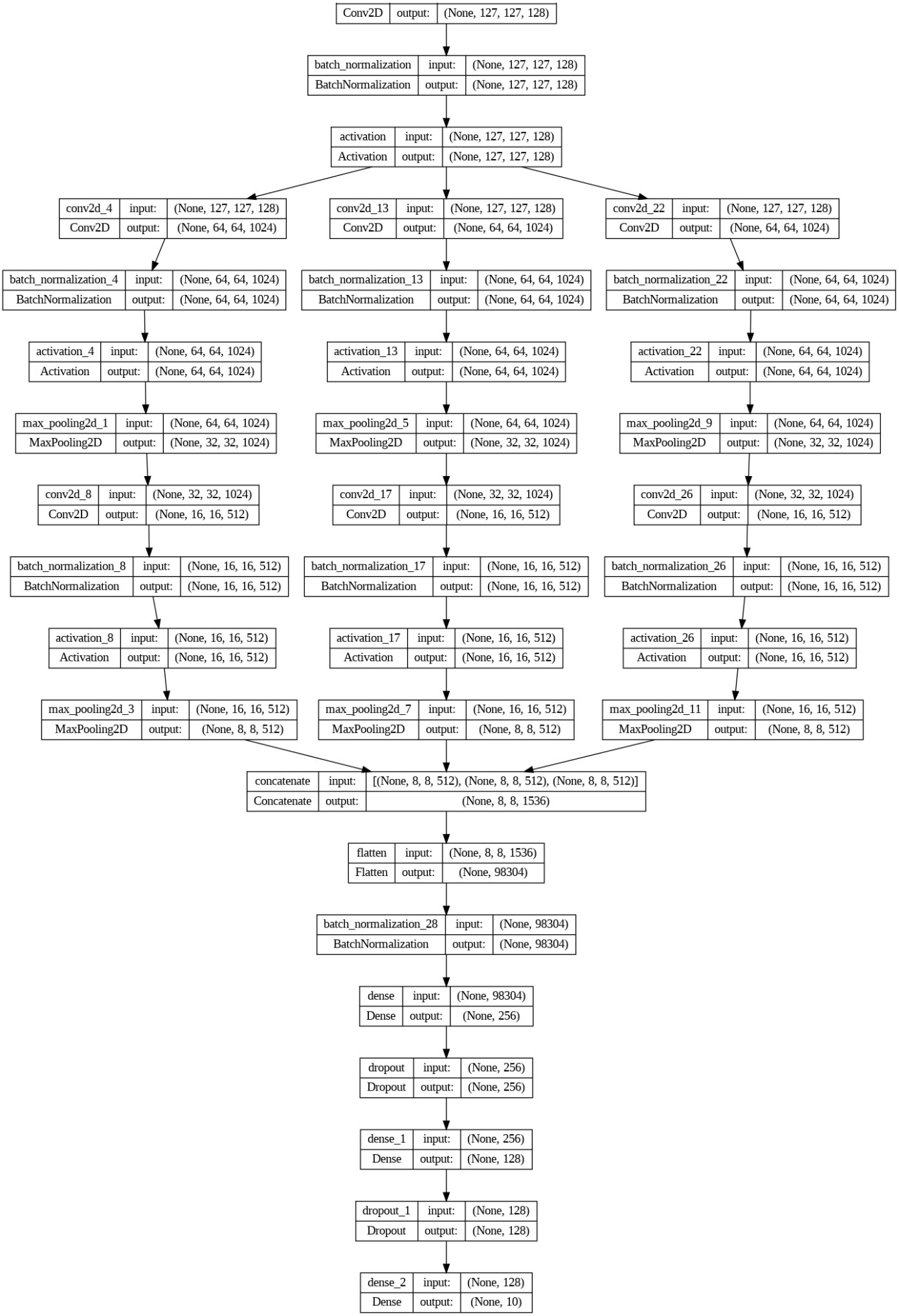
Custom model

In [ ]:
```
### Scratch to CNN custom ###
ft = image_size // 2
branch = []

inp = Input(shape=[image_size, image_size, channel])

# Initialising Augmentation
x = Sequential([tf.keras.layers.RandomFlip("horizontal_and_vertical"),
                tf.keras.layers.RandomRotation(0.2),
                tf.keras.layers.Rescaling(scale=1./255, offset = 0)])(inp)

# Initialising Convolution
x = Conv2D(ft, (4, 4), strides=(2, 2), input_shape = [image_size, image_size, channel])(x)
x = BatchNormalization(momentum = 0.95, epsilon = 1e-3)(x)
x = Activation('LeakyReLU')(x)

for i in range(round(math.sqrt(num_classes))):
  # Convolutional Layer
  for _ in range(2, num_classes, 2) :
    a1 = Conv2D(_ * ft, (_//2, _//2), strides=(2, 2), padding='same', kernel_initializer='he_normal')(x)
    a1 = BatchNormalization(momentum = 0.95, epsilon = 1e-3)(a1)
    a1 = Activation('LeakyReLU')(a1)

    if _ % 4 == 0 :
      # Pooling
      a1 = MaxPool2D((4, 4), strides = (2, 2), padding='same')(a1)

  # Convolutional Layer
  for _ in range(num_classes, 1, -2) :
    b1 = Conv2D(_ * ft, (_//2, _//2), strides=(2, 2), padding='same', kernel_initializer='he_normal')(a1)
    b1 = BatchNormalization(momentum = 0.95, epsilon = 1e-3)(b1)
    b1 = Activation('LeakyReLU')(b1)

    if _ % 4 == 0 :
      # Pooling Layer
      globals()['p'+str(i)] = MaxPool2D((4, 4), strides = (2, 2), padding='same')(b1)

  branch.append(globals()['p'+str(i)])

# Concatenate
c1 = Concatenate(axis = -1)(branch)

# Flattening
f1 = Flatten()(c1)
bat1 = BatchNormalization(momentum = 0.95, epsilon = 1e-3)(f1)

# Full Connection
out_put = Dense(ft * 2, activation = None, kernel_initializer='he_normal')(bat1)
out_put = Dropout(0.3)(out_put)

out_put = Dense(ft, activation = None, kernel_initializer='he_normal')(out_put)
out_put = Dropout(0.3)(out_put)

# Output Layer
out_put = Dense(num_classes, activation='softmax')(out_put)

model = Model(inputs = inp, outputs=out_put)
```

In [ ]:
```
# 모델 형태 요약
plot_model(model, "multi_cnn_model.png", show_shapes=True)
```

Out[ ]:





In [ ]:
```
# Model summary
adam = tf.keras.optimizers.legacy.Adam(learning_rate=0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = None, decay
model.compile(loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
   optimizer = adam,
   metrics=['accuracy'])

print()
```

```
model.summary()
```

```
Model: "model"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
sequential (Sequential)	(None, 256, 256, 3)	0	['input_1[0][0]']
conv2d (Conv2D)	(None, 127, 127, 128)	6272	['sequential[0][0]']
batch_normalization (BatchNormalization)	(None, 127, 127, 128)	512	['conv2d[0][0]']
activation (Activation)	(None, 127, 127, 128)	0	['batch_normalization[0][0]']
conv2d_4 (Conv2D)	(None, 64, 64, 1024)	2098176	['activation[0][0]']
conv2d_13 (Conv2D)	(None, 64, 64, 1024)	2098176	['activation[0][0]']
conv2d_22 (Conv2D)	(None, 64, 64, 1024)	2098176	['activation[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 64, 64, 1024)	4096	['conv2d_4[0][0]']
batch_normalization_13 (BatchNormalization)	(None, 64, 64, 1024)	4096	['conv2d_13[0][0]']
batch_normalization_22 (BatchNormalization)	(None, 64, 64, 1024)	4096	['conv2d_22[0][0]']
activation_4 (Activation)	(None, 64, 64, 1024)	0	['batch_normalization_4[0][0]']
activation_13 (Activation)	(None, 64, 64, 1024)	0	['batch_normalization_13[0][0]']
activation_22 (Activation)	(None, 64, 64, 1024)	0	['batch_normalization_22[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 1024)	0	['activation_4[0][0]']
max_pooling2d_5 (MaxPooling2D)	(None, 32, 32, 1024)	0	['activation_13[0][0]']
max_pooling2d_9 (MaxPooling2D)	(None, 32, 32, 1024)	0	['activation_22[0][0]']
conv2d_8 (Conv2D)	(None, 16, 16, 512)	2097664	['max_pooling2d_1[0][0]']
conv2d_17 (Conv2D)	(None, 16, 16, 512)	2097664	['max_pooling2d_5[0][0]']
conv2d_26 (Conv2D)	(None, 16, 16, 512)	2097664	['max_pooling2d_9[0][0]']
batch_normalization_8 (BatchNormalization)	(None, 16, 16, 512)	2048	['conv2d_8[0][0]']
batch_normalization_17 (BatchNormalization)	(None, 16, 16, 512)	2048	['conv2d_17[0][0]']
batch_normalization_26 (BatchNormalization)	(None, 16, 16, 512)	2048	['conv2d_26[0][0]']
activation_8 (Activation)	(None, 16, 16, 512)	0	['batch_normalization_8[0][0]']
activation_17 (Activation)	(None, 16, 16, 512)	0	['batch_normalization_17[0][0]']
activation_26 (Activation)	(None, 16, 16, 512)	0	['batch_normalization_26[0][0]']
max_pooling2d_3 (MaxPooling2D)	(None, 8, 8, 512)	0	['activation_8[0][0]']
max_pooling2d_7 (MaxPooling2D)	(None, 8, 8, 512)	0	['activation_17[0][0]']
max_pooling2d_11 (MaxPooling2D)	(None, 8, 8, 512)	0	['activation_26[0][0]']

```
 concatenate (Concatenate)      (None, 8, 8, 1536)   0           ['max_pooling2d_3[0][0]',
                                                                  'max_pooling2d_7[0][0]',
                                                                  'max_pooling2d_11[0][0]']

 flatten (Flatten)              (None, 98304)        0           ['concatenate[0][0]']

 batch_normalization_28 (BatchN  (None, 98304)       393216      ['flatten[0][0]']
 ormalization)

 dense (Dense)                  (None, 256)          25166080    ['batch_normalization_28[0][0]']

 dropout (Dropout)              (None, 256)          0           ['dense[0][0]']

 dense_1 (Dense)                (None, 128)          32896       ['dropout[0][0]']

 dropout_1 (Dropout)            (None, 128)          0           ['dense_1[0][0]']

 dense_2 (Dense)                (None, 10)           1290        ['dropout_1[0][0]']

==================================================================================================
Total params: 38,206,218
Trainable params: 38,000,138
Non-trainable params: 206,080
__________________________________________________________________________________________________
```

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

In [ ]:
```
# model 폴더 확인
if os.path.isdir(path) :
  pass

else :
  os.makedirs(os.path.join(path))
```

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

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

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

model_history.append(history)
```

[학습 로그 부분 삭제]

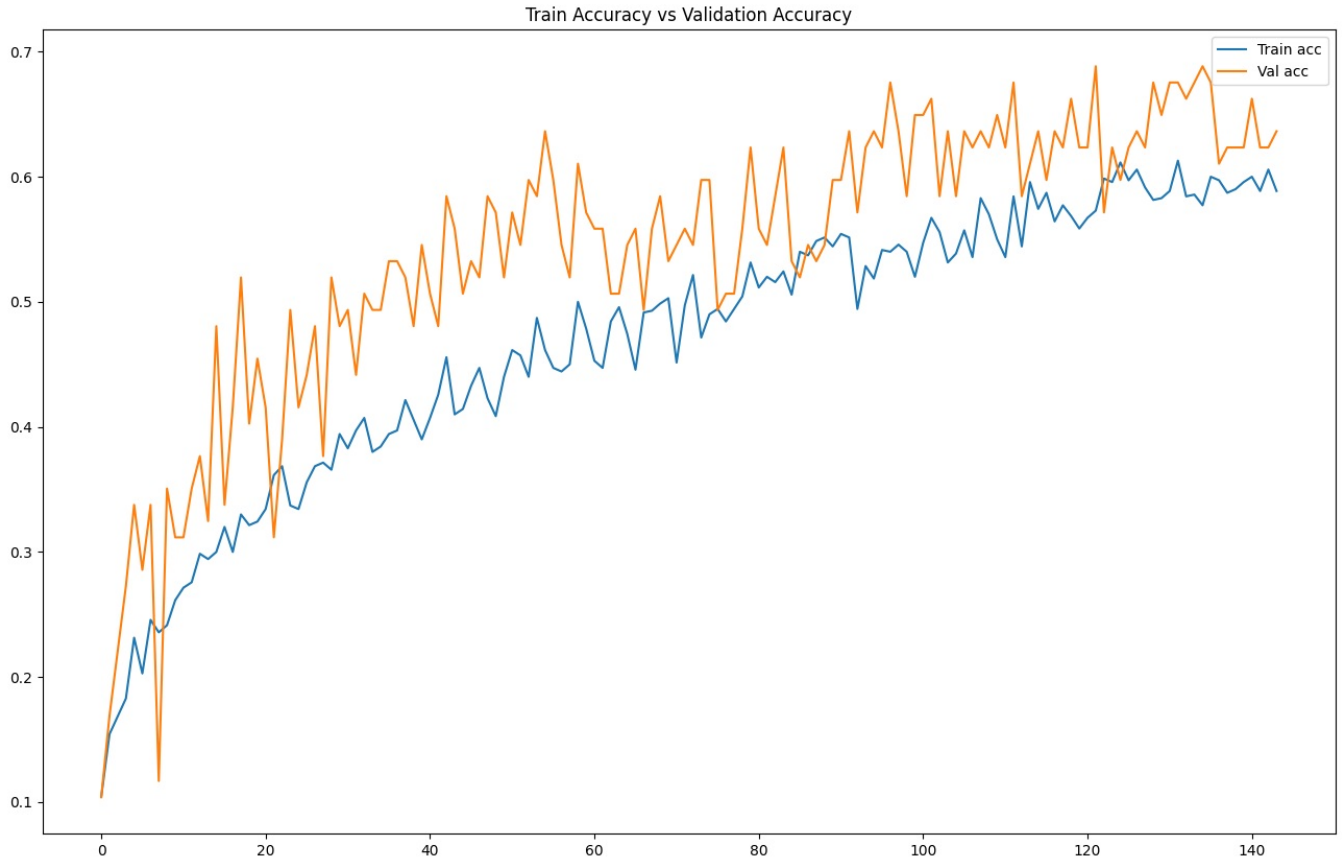
In [ ]:
```
len(model_history)
```

Out[ ]:
```
1
```

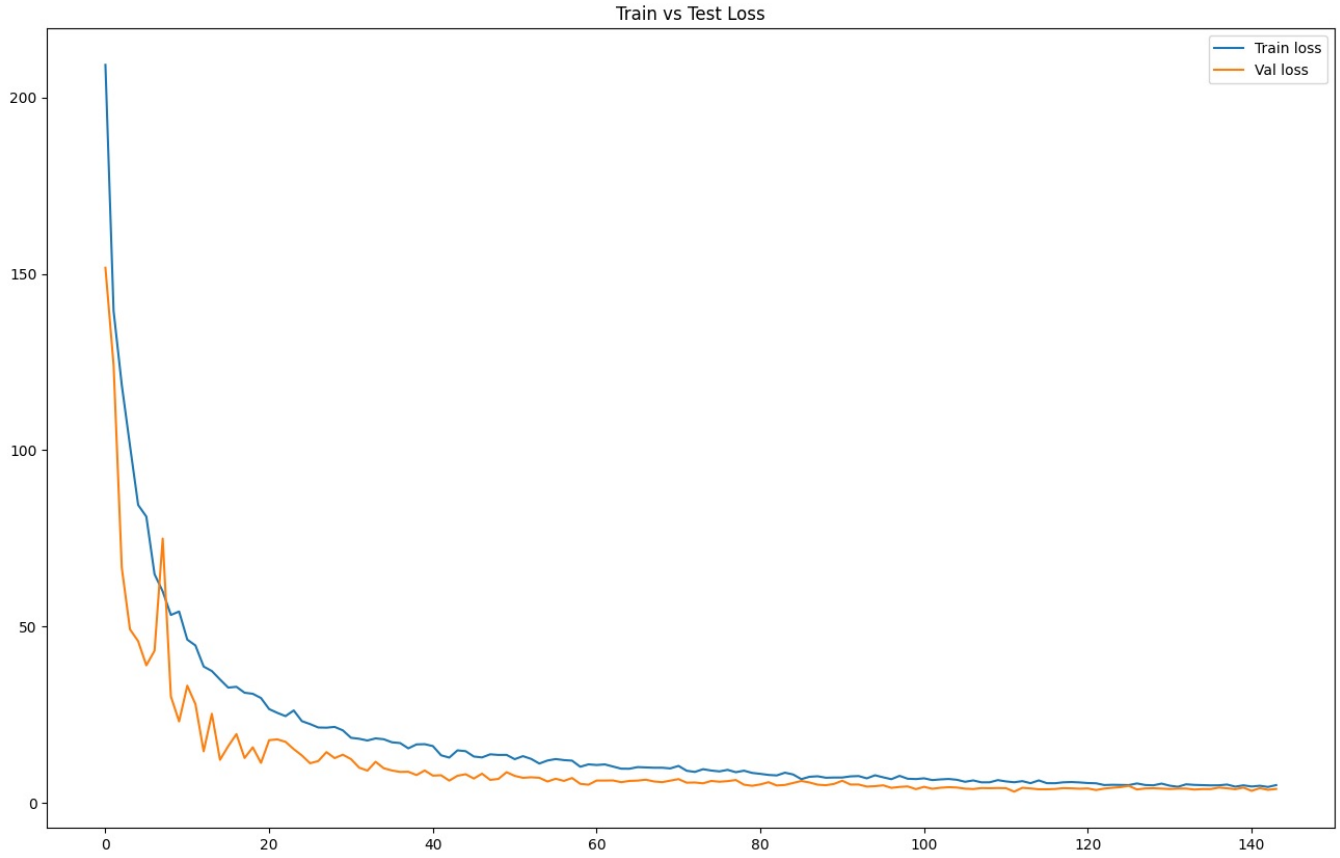
In [ ]:
```
# 리스트를 추출한다.
loss_list = history.history['loss']
accuracy_list = history.history['accuracy']
val_loss_list = history.history['val_loss']
```

```
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 [==============================] - 19s 638ms/step - loss: 8.0716 - accuracy: 0.4000
```

Out[ ]:
```
[8.071637153625488, 0.4000000059604645]
```

In [ ]:
```
# 정지
//정지
```

In [ ]:

In [ ]:

이하 휴지통 (초기 모델 및 데이터 전처리, not TFDS)

In [ ]:
```
# # 라벨 체크
# categories = list(os.walk(PATH+'test'))[0][1]
# categories
```

In [ ]:
```
# # Imgdatagen
# idg = ImageDataGenerator(horizontal_flip = True,
#                          vertical_flip = True,
#                          rotation_range = 45,
#                          zoom_range = 0.2,
#                          height_shift_range = 0.2,
#                          width_shift_range = 0.2,
#                         )
```

In [ ]:
```
# # 데이터 저장

# X = []
# y = []

# 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):
#         img_org = Image.open(j)
#         img1 = img_org.resize((image_w, image_h))
#         data = np.asarray(img1) / 255

#         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)

#     print(category, 'done')

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

In [ ]:
```
# # 개수 확인 (770 * Augmentation)
# X.shape, y.shape
```

In [ ]:
```
# # 분포확인 (불균형 체크, 컬럼명 다름 주의)
# plt.figure(figsize=(9,5))
# sns.barplot(x = pd.DataFrame(y).columns, y = pd.DataFrame(y).value_counts(), palette = "rocket")
# plt.title("Number of pictures of each category", fontsize = 15)
# plt.show()
```

In [ ]:
```
# # X 샘플확인
# plt.figure(figsize=(16,10))
# for g in range(len(X[:64:4])):
#     plt.subplot(4,4,g+1)
#     plt.imshow(X[g])
#     plt.axis('off')
```

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 [ ]:
```
# ## 3호 ###
# ft = image_size // 2

# # Initialising Augmentation
# model = Sequential([tf.keras.layers.RandomFlip("horizontal_and_vertical", input_shape=(image_size, image_size,
#                     tf.keras.layers.RandomRotation(0.2),
#                     tf.keras.layers.RandomContrast(0.1),
#                     tf.keras.layers.RandomBrightness(0.1),
#                     tf.keras.layers.Rescaling(scale=1./255, offset = 0)
# ])

# # Initialising Convolution
# model.add(Conv2D(ft, (4, 4), strides=(2, 2), input_shape = [image_size, image_size, channel]))
# model.add(BatchNormalization(momentum = 0.95, epsilon = 1e-3))
```

```
# model.add(Activation('LeakyReLU'))


# # Convolutional Layer
# for _ in range(2, num_classes, 2) :
#   model.add(Conv2D(_ * ft, (_//2, _//2), strides=(2, 2), padding='same', kernel_initializer='he_normal'))
#   model.add(BatchNormalization(momentum = 0.95, epsilon = 1e-3))
#   model.add(Activation('LeakyReLU'))

#   if _ % 4 == 0 :
#     # Pooling
#     model.add(MaxPool2D((4, 4), strides = (2, 2), padding='same'))


# # Convolutional Layer
# for _ in range(num_classes, 1, -2) :
#   model.add(Conv2D(_ * ft, (_//2, _//2), strides=(2, 2), padding='same', kernel_initializer='he_normal'))
#   model.add(BatchNormalization(momentum = 0.95, epsilon = 1e-3))
#   model.add(Activation('LeakyReLU'))

#   if _ % 4 == 0 :
#     # Pooling
#     model.add(MaxPool2D((4, 4), strides = (2, 2), padding='same'))


# # Flattening
# model.add(Flatten())

# # Full Connection
# model.add(Dense(ft * 2, activation = None, kernel_initializer='he_normal'))
# model.add(Dropout(0.3))

# model.add(Dense(ft, activation = None, kernel_initializer='he_normal'))
# model.add(Dropout(0.3))

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

In [ ]:
```
# # 모델 형태 요약
# plot_model(model, "multi_cnn_model.png", show_shapes=True)
```

In [ ]:
```
# ### 4호 ###
# ft = image_size // 2
# branch = []

# repeat = math.ceil(math.sqrt(num_classes))

# # Initialising Convolution
# inp = Input(shape=[image_size, image_size, channel])
# x = Sequential([tf.keras.layers.RandomFlip("horizontal_and_vertical"),
#                 tf.keras.layers.RandomRotation(0.2),
#                 tf.keras.layers.RandomBrightness(0.1),
#                 tf.keras.layers.Rescaling(scale=1./255, offset = 0)])(inp)

# for i in range(2, 8, 2):
#   # Initialising Augmentation
#   x = Conv2D(repeat, (4, 4), strides=(4, 4), padding = 'same')(x)
#   x = Conv2D(repeat * 2, (4, 4), strides=(3, 3), padding = 'same')(x)
#   x = Conv2D(repeat * 4, (4, 4), strides=(2, 2), padding = 'same')(x)
#   x = Activation('LeakyReLU')(x)
#   globals()['a'+str(i)] = MaxPool2D((4, 4), strides = (2, 2), padding='same')(x)

#   for _ in range(2, 6, 2) :
#     globals()['b'+str(i)+str(_)] = Conv2D(repeat * 2, (4, 4), strides=(4, 4), padding='same', kernel_initiali
#     globals()['b'+str(i)+str(_)] = Conv2D(repeat * 2, (3, 3), strides=(3, 3), padding='same', kernel_initiali
#     globals()['b'+str(i)+str(_)] = Conv2D(repeat * 4, (2, 2), strides=(2, 2), padding='same', kernel_initiali
#     globals()['b'+str(i)+str(_)] = Activation('LeakyReLU')(globals()['b'+str(i)+str(_)])
#     globals()['b'+str(i)+str(_)] = MaxPool2D((4, 4), strides = (2, 2), padding='same')(globals()['b'+str(i)+st
#     branch.append((globals()['b'+str(i)+str(_)]))

# c1 = Concatenate(axis = -1)(branch)
# c1 = BatchNormalization(momentum = 0.95, epsilon = 1e-3)(c1)
# c1 = Flatten()(c1)


# # Full Connection
# d1 = Dense(ft * num_classes, activation = None, kernel_initializer='he_normal')(c1)
# d1 = Dropout(0.5)(d1)
# d1 = Dense(ft, activation = None, kernel_initializer='he_normal')(d1)
# d1 = Dropout(0.5)(d1)

# d2 = Dense(ft * num_classes, activation = None, kernel_initializer='he_normal')(d1)
# d2 = Dropout(0.5)(d2)
```

```
# d2 = Dense(ft, activation = None, kernel_initializer='he_normal')(d2)
# d2 = Dropout(0.5)(d2)

# # Output Layer
# out_put = Concatenate(axis = -1)([d1, d2])
# out_put = Dense(num_classes, activation='softmax')(out_put)

# model = Model(inputs = inp, outputs =  out_put)
```

In [ ]:
```
# # Train, Val 분리
# X_train, X_test, y_train, y_test = train_test_split(X, y,
#                                                     random_state = 18,
#                                                     shuffle = True,
#                                                     test_size = 0.3)
```

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

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

# 조기 중단
# call3 = EarlyStopping(monitor = 'val_loss', patience = 50 )
```

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,
#                     workers = 8,
#                    callbacks= [call3]
#                    )

# model_history.append(history)
```