Final Programming Exam. (part 2)
Deep Learning, 2022

姓名：吳佳展　　學號：110318517

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt

os.environ["CUDA_VISIBLE_DEVICES"] = "2"
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (Dense, Dropout, Activation,
                                     Flatten, GlobalAveragePooling2D)
import tensorflow as tf

data_path = 'Data_2/train/'

x_data_list = []
y_data_list = []
for roots, dirs, files in os.walk(data_path):
    for each in files:
        if each.find('checkpoint') == -1:
            x_data_list.append(os.path.join(roots.split("/")[-1], each))
            y_data_list.append(roots.split("/")[-1])

            data_list = pd.DataFrame({})
data_list['img_path'] = x_data_list
data_list['label'] = y_data_list

data_list['label'] = data_list['label'].astype('string')

from sklearn.model_selection import train_test_split

train_list, test_list = train_test_split(data_list,
                                         test_size=0.1,
                                         random_state=40,
                                         stratify=data_list['label'].values)
```

```
img_size = 256
num_class = len(data_list['label'].unique())
from tensorflow.keras.applications.resnet import (ResNet50,ResNet101,
preprocess_input)
resnet_model = ResNet101(weights='imagenet', include_top=False,
                  input_shape=(img_size, img_size, 3))

x = GlobalAveragePooling2D()(resnet_model.output)
x = Dropout(0.27)(x)
outputs = Dense(num_class, activation='softmax')(x)
model = Model(inputs=resnet_model.inputs, outputs=outputs)
model.summary()

learning_rate = 1e-5
optimizer = keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(loss='categorical_crossentropy',
            optimizer=optimizer,
            metrics=['accuracy'])

batch_size = 32
num_steps = len(train_list) // batch_size + 1
num_epochs = 100

train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
test_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)

img_shape = (img_size, img_size)

train_generator = train_datagen.flow_from_dataframe(
                                            dataframe=train_list,
                                            directory=data_path,
                                            x_col="img_path",
                                            y_col="label",
                                            target_size=img_shape,
                                            batch_size=batch_size,
                                            class_mode='categorical')

valid_generator = test_datagen.flow_from_dataframe(
                                                    dataframe=test_list,
                                                    directory=data_path,
                                                    x_col="img_path",
                                                    y_col="label",
                                                    target_size=img_shape,
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=False)
model_dir = 'Q2_model/'
if not os.path.exists(model_dir):
    os.makedirs(model_dir)

# logfiles = model_dir + '/{}-{}'.format('basic_model',
#                                          model.__class__.__name__)
# model_cbk = keras.callbacks.TensorBoard(log_dir=logfiles,
#                                           histogram_freq=1)

modelfiles = model_dir + 'vbn.h5'
model_mckp = keras.callbacks.ModelCheckpoint(modelfiles,
                                             monitor='val_accuracy',
                                             save_best_only=True)

earlystop = keras.callbacks.EarlyStopping(monitor='val_loss',
                                          patience=20,
                                          verbose=1)
# callbacks_list = [model_cbk, model_mckp, earlystop]
callbacks_list = [model_mckp, earlystop]

history = model.fit_generator(train_generator,
                              steps_per_epoch=num_steps,
                              epochs=num_epochs,
                              validation_data=valid_generator,
                              callbacks=callbacks_list)

loss, acc = model.evaluate_generator(valid_generator, verbose=2)
train_history = ['loss', 'val_loss', 'accuracy', 'val_accuracy']
name_history = ['training_loss', 'val_loss', 'training_acc', 'val_acc']
```

```
plt.figure(figsize=(10, 6))
for eachx, eachy, i in zip(train_history, name_history, range(4)):
    if i % 2 == 0:
        plt.subplot(1, 2, i//2+1)
    l_x = len(history.history[eachx])
    plt.plot(np.arange(l_x), history.history[eachx], label=eachy)
    plt.legend(loc='best')
    plt.title('model'+eachy)
plt.savefig(f'Q2_model/type2.png')
plt.show()
```