LƯU MODEL – KERAS

```
#install lib
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from oauth2client.client import GoogleCredentials
```

Mount Google Drive

```
from google.colab import auth
#@title Mount Google Drive { vertical-output: true }

## Make project folder in Virtual Machine of Colab (Colab VM)
## Note that this is just a temporary folder for a session.
%cd /content
# !mkdir -p <project_name>
# %cd /content/<project_name>/
# For example, <project_name> = "MNIST"
!mkdir -p mnist
%cd /content/mnist/

## Mount google drive into current Colab VM
from google.colab import drive
drive.mount('/content/drive')

## Create symbolic links to Google Drive
# !ln -s <path_to_folder_in_Google_Drive> <path_to_folder_in_VM>

!ln -s /content/drive/My\ Drive/gd_mnist/src /content/mnist
!ln -s /content/drive/My\ Drive/gd_mnist/log /content/mnist
!ln -s /content/drive/My\ Drive/gd_mnist/model /content/mnist

## From now you can see your linked GoogleDrive folder in "Files" menu.
# Whenever you make any change in linked folder, the linked Google Drive f
# older
```

New Section

```
pip install httplib2==0.15.0
#authenticate
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
```

```
drive = GoogleDrive(gauth)


#if we load model in different notebook, we have to define a new model fir
# st
#for example, I will define a model as a model I saved:
#download inception & import lib
!wget --no-check-certificate \
      https://storage.googleapis.com/mledu-
datasets/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5 \
    -O /tmp/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
from tensorflow.keras.applications.inception_v3 import InceptionV3
local_weights_file = '/tmp/inception_v3_weights_tf_dim_ordering_tf_kernels
_notop.h5'
pre_trained_model = InceptionV3(
    input_shape=(150, 150, 3), include_top=False, weights=None)
pre_trained_model.load_weights(local_weights_file)
import os
from tensorflow.keras import layers
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os


#links (dir & path)
path = "/content/gdrive/My Drive/CMND"
files = os.listdir(path)
for layer in pre_trained_model.layers:
  layer.trainable = False
last_layer = pre_trained_model.get_layer('mixed7')
print('last layer output shape:', last_layer.output_shape)
last_output = last_layer.output

# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)
# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)
# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

#Configure the model
model = Model(pre_trained_model.input, x)
```

```
path = "/content/gdrive/My Drive/CMND"
files = os.listdir(path)

# Define our example directories and files
train_dir = os.path.join(path, 'train')
validation_dir = os.path.join(path, 'validation')

# Directory with our training cat pictures
train_real_dir = os.path.join(train_dir, 'real')

# Directory with our training dog pictures
train_fake_dir = os.path.join(train_dir, 'fake')

# Directory with our validation cat pictures
validation_real_dir = os.path.join(validation_dir, 'real')

# Directory with our validation dog pictures
validation_fake_dir = os.path.join(validation_dir, 'fake')

train_real_fnames = os.listdir(train_real_dir)
train_fake_fnames = os.listdir(train_fake_dir)

# Add our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(train_dir,
                                   rescale=1/255,
                                   rotation_range=45,
                                   width_shift_range=.4)

# Note that the validation data should not be augmented!
test_datagen = ImageDataGenerator(validation_dir,
                                   rescale=1/255,
                                   rotation_range=45,
                                   width_shift_range=.4)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=100,
                                                    target_size=(150,150),
                                                    class_mode='binary')


# Flow validation images in batches of 20 using test_datagen generator
validation_generator =  test_datagen.flow_from_directory(validation_dir,
                                                         batch_size=100,
```

```
                                                    target_size=(150,1
50),
                                                    class_mode='binary
')

unfreeze = False

# Unfreeze all models after "mixed6"
for layer in pre_trained_model.layers:
  if unfreeze:
    layer.trainable = True
  if layer.name == 'mixed6':
    unfreeze = True


#save model
model.save('name_model.h5')
model_file = drive.CreateFile({'link_model' : 'name_model.h5'})

model_file.SetContentFile('name_model.h5')
drive.CreateFile({'name_model.h5': model_file.get('name_model.h5')})


#load model
import tensorflow
# Recreate the exact same model, including its weights and the optimizer
new_model = tensorflow.keras.models.load_model('name_model.h5')

# Show the model architecture
new_model.summary()


import datetime as dt

dt.time

#The arguments are hour, minute, second, microsecond in order.
#If you do not supply a value of 0 is assumed as the default. You can call
 dt.time() with no arguments to get 00:00:00
print("1 Hour")
print (dt.time(1))

print("2 and a half hour")
print(dt.time(2, 30))

print("3 hours, 45 minutes and 30 seconds")
```

```
print(dt.time(3, 45, 30))

#system datetime
ts_now = dt.datetime.now()
print(ts_now)
```

A customed Callback class

```
#@title A customed Callback class

class WeightsSaverByEpoch(Callback):
  def __init__(self, N):
    self.N = N
    self.epoch = 0

  def on_epoch_end(self, epoch, logs={}):
    if self.epoch % self.N == 0:
        name = ('weights%04d.hdf5') % self.epoch
        self.model.save_weights(name)
    self.epoch += 1

class WeightsSaverByBatch(Callback):
  def __init__(self, N):
    self.N = N
    self.batch = 0

  def on_batch_end(self, batch, logs={}):
    if self.batch % self.N == 0:
        name = 'weights%08d.h5' % self.batch
        self.model.save_weights(name)
    self.batch += 1

callbacks_list = [WeightsSaverByEpoch(10)] #save every 10 epochs
## OR callbacks_list = [WeightsSaverByBatch(10)] #save every 10 batches
model.fit(train_X,train_Y,epochs=n_epochs,batch_size=batch_size, callbacks
=callbacks_list )
```

RECONNECT ERRORS chu kỳ 90mins & 12hours

Trong một số trường hợp, Colab có thể bị ngắt kết nối trước 90 phút. Có thể ấn `RECONNECT` để tiếp tục train hoặc khắc phục như sau:

Ấn `Ctrl+ Shift + i` và vào tab `Console` paste dòng lệnh sau:

```
function ClickConnect(){
console.log("Working");
document.querySelector("colab-toolbar-button#connect").click()
```

```
}
setInterval(ClickConnect,60000)
```

https://colab.research.google.com/drive/1RCK8kWln4fSYeHvY6rGbozB938iqYmVQ#scrollTo=kfealoxHTj6A

HẾT BỘ NHỚ GPU

Sử dụng lệnh `!df -h` để kiểm tra dung lượng chúng ta đã sử dụng.

```
!df -h
```

Khi bộ nhớ chính sử dụng quá 80% sẽ có hội thoại cảnh báo. Khi đó chúng ta sử dụng `free` để giải phỏng bộ nhớ.

```
!free -h
```

Kiểm tra trạng thái sư dụng GPU:

```
!nvidia-smi
```

Sau khi bị disconnect, có thể train lại như sau:

```
# load model
model = keras.model.load_model("saved_link/name_model.h5")

# thực hiện train tiếp
model.fit(...)
```
