Open in Colab

IMPORTANT NOTE: This notebook is designed to run as a Colab. Click the button on top that says, `Open in Colab`, to run this notebook as a Colab. Running the notebook on your local machine might result in some of the code blocks throwing errors.

```
import os
import zipfile
import random
import tensorflow as tf
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from shutil import copyfile
```

```
# If the URL doesn't work, visit https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765
# And right click on the 'Download Manually' link to get a new URL to the dataset

# Note: This is a very large dataset and will take time to download

!wget --no-check-certificate \
    "https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip" \
    -O "/tmp/cats-and-dogs.zip"

local_zip = '/tmp/cats-and-dogs.zip'
zip_ref   = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/tmp')
zip_ref.close()
```

```
--2021-10-20 10:47:29--  https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_3367a.zip
Resolving download.microsoft.com (download.microsoft.com)... 184.51.240.115, 2600:140e:6:b8d::e59, 2600:140e:6:ba1::e59
Connecting to download.microsoft.com (download.microsoft.com)|184.51.240.115|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 824894548 (787M) [application/octet-stream]
Saving to: ‘/tmp/cats-and-dogs.zip’

/tmp/cats-and-dogs. 100%[===================>] 786.68M   157MB/s    in 5.1s

2021-10-20 10:47:34 (155 MB/s) - ‘/tmp/cats-and-dogs.zip’ saved [824894548/824894548]
```

```
print(len(os.listdir('/tmp/PetImages/Cat/')))
print(len(os.listdir('/tmp/PetImages/Dog/')))

# Expected Output:
# 12501
# 12501
```

```
12501
12501
```

```
# Use os.mkdir to create your directories
# You will need a directory for cats-v-dogs, and subdirectories for training
# and testing. These in turn will need subdirectories for 'cats' and 'dogs'
try:

  ### START YOUR CODE HERE
  os.mkdir('/tmp/cats-v-dogs')
  os.mkdir('/tmp/cats-v-dogs/training')
  os.mkdir('/tmp/cats-v-dogs/testing')
  os.mkdir('/tmp/cats-v-dogs/training/cats')
  os.mkdir('/tmp/cats-v-dogs/training/dogs')
  os.mkdir('/tmp/cats-v-dogs/testing/cats')
  os.mkdir('/tmp/cats-v-dogs/testing/dogs')

  ### alternate solution
  # classes = ['cats','dogs']

  # for class_name in classes:
  #     os.makedirs(os.path.join("/tmp/cats-v-dogs", "training", class_name))
  #     os.makedirs(os.path.join("/tmp/cats-v-dogs", "testing", class_name))

  ### END YOUR CODE HERE
except OSError:
    pass


# Write a python function called split_data which takes
# a SOURCE directory containing the files
# a TRAINING directory that a portion of the files will be copied to
# a TESTING directory that a portion of the files will be copie to
# a SPLIT SIZE to determine the portion
```

```
# The files should also be randomized, so that the training set is a random
# X% of the files, and the test set is the remaining files
# SO, for example, if SOURCE is PetImages/Cat, and SPLIT SIZE is .9
# Then 90% of the images in PetImages/Cat will be copied to the TRAINING dir
# and 10% of the images will be copied to the TESTING dir
# Also -- All images should be checked, and if they have a zero file length,
# they will not be copied over
#
# os.listdir(DIRECTORY) gives you a listing of the contents of that directory
# os.path.getsize(PATH) gives you the size of the file
# copyfile(source, destination) copies a file from source to destination
# random.sample(list, len(list)) shuffles a list
def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):


  files = []
  for filename in os.listdir(SOURCE):
    file = SOURCE + filename
    if os.path.getsize(file) > 0:
      files.append(filename)
    else:
      print(filename + " is zero length, so ignoring.")

  training_length = int(len(files) * SPLIT_SIZE)
  testing_length = int(len(files) - training_length)
  shuffled_set = random.sample(files, len(files))
  training_set = shuffled_set[0:training_length]
  testing_set = shuffled_set[-testing_length:]

  for filename in training_set:
    this_file = SOURCE + filename
    destination = TRAINING + filename
    copyfile(this_file, destination)

  for filename in testing_set:
    this_file = SOURCE + filename
    destination = TESTING + filename
    copyfile(this_file, destination)



  ### alternate solution
```

```
  # # YOUR CODE STARTS HERE
  # files = os.listdir(SOURCE)
  # files = [file for file in files if os.path.getsize(os.path.join(SOURCE, file))>0]

  # files = random.sample(files, len(files))
  # limit = len(files) * SPLIT_SIZE

  # for index, file in enumerate(files):
  #   source = os.path.join(SOURCE, file)
  #   if index < limit:
  #     destination = os.path.join(TRAINING, file)
  #   else:
  #     destination = os.path.join(TESTING, file)

  #   copyfile(source, destination)
  # # YOUR CODE ENDS HERE


CAT_SOURCE_DIR = "/tmp/PetImages/Cat/"
TRAINING_CATS_DIR = "/tmp/cats-v-dogs/training/cats/"
TESTING_CATS_DIR = "/tmp/cats-v-dogs/testing/cats/"
DOG_SOURCE_DIR = "/tmp/PetImages/Dog/"
TRAINING_DOGS_DIR = "/tmp/cats-v-dogs/training/dogs/"
TESTING_DOGS_DIR = "/tmp/cats-v-dogs/testing/dogs/"

split_size = .9
split_data(CAT_SOURCE_DIR, TRAINING_CATS_DIR, TESTING_CATS_DIR, split_size)
split_data(DOG_SOURCE_DIR, TRAINING_DOGS_DIR, TESTING_DOGS_DIR, split_size)

# Expected output
# 666.jpg is zero length, so ignoring
# 11702.jpg is zero length, so ignoring

     666.jpg is zero length, so ignoring.
     11702.jpg is zero length, so ignoring.


print(len(os.listdir('/tmp/cats-v-dogs/training/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/training/dogs/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/cats/')))
print(len(os.listdir('/tmp/cats-v-dogs/testing/dogs/')))

# Expected output:
```

```
# 11250
# 11250
# 1250
# 1250

     11250
     11250
     1250
     1250


# DEFINE A KERAS MODEL TO CLASSIFY CATS V DOGS
# USE AT LEAST 3 CONVOLUTION LAYERS
model = tf.keras.models.Sequential([
    ### START CODE HERE
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    ### END CODE HERE
])

model.compile(optimizer=RMSprop(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])



TRAINING_DIR = "/tmp/cats-v-dogs/training/" #YOUR CODE HERE
train_datagen = ImageDataGenerator(rescale=1.0/255.) #YOUR CODE HERE
train_generator = train_datagen.flow_from_directory(TRAINING_DIR, #YOUR CODE HERE
                                                    batch_size=100,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

VALIDATION_DIR = "/tmp/cats-v-dogs/testing/" #YOUR CODE HERE
validation_datagen = ImageDataGenerator(rescale=1.0/255.) #YOUR CODE HERE
validation_generator = validation_datagen.flow_from_directory(VALIDATION_DIR, #YOUR CODE HERE
                                                              batch_size=100,
                                                              class_mode='binary',
```

```
                                                  target_size=(150, 150))

# Expected Output:
# Found 22498 images belonging to 2 classes.
# Found 2500 images belonging to 2 classes.
```

```
Found 22498 images belonging to 2 classes.
Found 2500 images belonging to 2 classes.
```

Note: You can ignore the `UserWarning: Possibly corrupt EXIF data.` warnings.

```
# Note that this may take some time.
history = model.fit(train_generator,
                              epochs=50,
                              verbose=1,
                              validation_data=validation_generator)
# The expectation here is that the model will train, and that accuracy will be > 95% on both training and validation
# i.e. acc:A1 and val_acc:A2 will be visible, and both A1 and A2 will be > .9
```

```
Epoch 1/50
  8/225 [>.............................] - ETA: 1:07 - loss: 2.7813 - accuracy: 0.5225/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning:
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to read 5 bytes but only got 0. Skipping tag 27
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to read 8 bytes but only got 0. Skipping tag 27
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to read 8 bytes but only got 0. Skipping tag 28
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to read 8 bytes but only got 0. Skipping tag 28
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to read 20 bytes but only got 0. Skipping tag 3
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:770: UserWarning: Possibly corrupt EXIF data.  Expecting to read 48 bytes but only got 0. Skipping tag 5
  " Skipping tag %s" % (size, len(data), tag)
/usr/local/lib/python3.7/dist-packages/PIL/TiffImagePlugin.py:788: UserWarning: Corrupt EXIF data.  Expecting to read 2 bytes but only got 0.
  warnings.warn(str(msg))
225/225 [==============================] - 118s 389ms/step - loss: 0.7167 - accuracy: 0.6452 - val_loss: 0.5804 - val_accuracy: 0.6824
Epoch 2/50
225/225 [==============================] - 86s 382ms/step - loss: 0.5182 - accuracy: 0.7414 - val_loss: 0.4653 - val_accuracy: 0.7772
Epoch 3/50
225/225 [==============================] - 86s 381ms/step - loss: 0.4412 - accuracy: 0.7903 - val_loss: 0.4526 - val_accuracy: 0.7884
Epoch 4/50
225/225 [==============================] - 86s 382ms/step - loss: 0.3756 - accuracy: 0.8296 - val_loss: 0.4236 - val_accuracy: 0.8156
Epoch 5/50
225/225 [==============================] - 87s 385ms/step - loss: 0.3130 - accuracy: 0.8631 - val_loss: 0.3800 - val_accuracy: 0.8392
```

```
Epoch 6/50
225/225 [==============================] - 88s 390ms/step - loss: 0.2381 - accuracy: 0.9003 - val_loss: 0.4108 - val_accuracy: 0.8396
Epoch 7/50
225/225 [==============================] - 88s 391ms/step - loss: 0.1705 - accuracy: 0.9312 - val_loss: 0.4929 - val_accuracy: 0.8380
Epoch 8/50
225/225 [==============================] - 87s 388ms/step - loss: 0.1083 - accuracy: 0.9600 - val_loss: 0.7138 - val_accuracy: 0.7928
Epoch 9/50
225/225 [==============================] - 86s 383ms/step - loss: 0.0746 - accuracy: 0.9740 - val_loss: 0.6512 - val_accuracy: 0.8392
Epoch 10/50
225/225 [==============================] - 87s 386ms/step - loss: 0.0553 - accuracy: 0.9813 - val_loss: 0.8781 - val_accuracy: 0.8324
Epoch 11/50
225/225 [==============================] - 87s 385ms/step - loss: 0.0487 - accuracy: 0.9856 - val_loss: 0.7933 - val_accuracy: 0.8196
Epoch 12/50
225/225 [==============================] - 87s 386ms/step - loss: 0.0465 - accuracy: 0.9880 - val_loss: 1.2845 - val_accuracy: 0.8044
Epoch 13/50
225/225 [==============================] - 86s 381ms/step - loss: 0.0487 - accuracy: 0.9875 - val_loss: 1.0043 - val_accuracy: 0.8236
Epoch 14/50
225/225 [==============================] - 87s 384ms/step - loss: 0.0384 - accuracy: 0.9894 - val_loss: 1.2925 - val_accuracy: 0.8172
Epoch 15/50
225/225 [==============================] - 86s 383ms/step - loss: 0.0535 - accuracy: 0.9844 - val_loss: 1.1708 - val_accuracy: 0.8276
Epoch 16/50
201/225 [=========================>....] - ETA: 8s - loss: 0.0447 - accuracy: 0.9888
```

```
%matplotlib inline

import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc)) # Get number of epochs

#------------------------------------------------
# Plot training and validation accuracy per epoch
#------------------------------------------------
plt.plot(epochs, acc, 'r', "Training Accuracy")
```

```
plt.plot(epochs, val_acc, 'b', "Validation Accuracy")
plt.title('Training and validation accuracy')
plt.figure()

#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r', "Training Loss")
plt.plot(epochs, val_loss, 'b', "Validation Loss")
plt.figure()


# Desired output. Charts with training and validation metrics. No crash :)
```

Important Note: Due to some compatibility issues, the following code block will result in an error after you select the images(s) to upload if you are running this notebook as a `Colab` on the `Safari` browser. For `all other broswers`, continue with the next code block and ignore the next one after it.

The ones running the `Colab` on `Safari`, comment out the code block below, uncomment the next code block and run it.

```
# Here's a codeblock just for fun. You should be able to upload an image here
# and have it classified without crashing
import numpy as np
from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():

  # predicting images
  path = '/content/' + fn
  img = image.load_img(path, target_size=(150, 150))
  x = image.img_to_array(img)
  x = np.expand_dims(x, axis=0)

  images = np.vstack([x])
  classes = model.predict(images, batch_size=10)
  print(classes[0])
  if classes[0]>0.5:
```

```
    print(fn + " is a dog")
  else:
    print(fn + " is a cat")
```

For those running this `Colab` on `Safari` broswer can upload the images(s) manually. Follow the instructions, uncomment the code block below and run it.

Instructions on how to upload image(s) manually in a Colab:

1. Select the `folder` icon on the left `menu bar`.
2. Click on the `folder with an arrow pointing upwards` named `..`
3. Click on the `folder` named `tmp`.
4. Inside of the `tmp` folder, `create a new folder` called `images`. You'll see the `New folder` option by clicking the `3 vertical dots` menu button next to the `tmp` folder.
5. Inside of the new `images` folder, upload an image(s) of your choice, preferably of either a horse or a human. Drag and drop the images(s) on top of the `images` folder.
6. Uncomment and run the code block below.

```
import numpy as np
from keras.preprocessing import image
import os

images = os.listdir("/tmp/images")

print(images)

for i in images:
 print()
 # predicting images
 path = '/tmp/images/' + i
 img = image.load_img(path, target_size=(150, 150))
 x = image.img_to_array(img)
 x = np.expand_dims(x, axis=0)

 images = np.vstack([x])
 classes = model.predict(images, batch_size=10)
 print(classes[0])
 if classes[0]>0.5:
```

```
  print(i +   is a dog ")
else:
  print(i + " is a cat")
```

23m 36s completed at 4:15 PM