Open in Colab

(https://colab.research.google.com/github/sergejhorvat/TensorFlow-Data-and-Deployment-Specialization/blob/master/Device-based%20Models%20with%20TensorFlow/Week%202/Exercise/TFLite_Week2_Exercise.ipynb)

In [0]:
```
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

Rock, Paper & Scissors with TensorFlow Hub - TFLite

Run in Google Colab (https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/TensorFlow%20Deployment/Course%202%20-%20TensorFlow%20Lite/Week%202/Exercise/TFLite_Week2_Exercise.ipynb)	View source on GitHub (https://github.com/lmoroney/dlaicourse/blob/master/TensorFlow%20Deployment/Course%202%20-%20TensorFlow%20Lite/Week%202/Exercise/TFLite_Week2_Exercise.ipynb)

Setup

In [0]:
```
try:
    %tensorflow_version 2.x
except:
    pass
```

In [0]:
```
import numpy as np
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

from tqdm import tqdm

print("\u2022 Using TensorFlow Version:", tf.__version__)
print("\u2022 Using TensorFlow Hub Version: ", hub.__version__)
print('\u2022 GPU Device Found.' if tf.test.is_gpu_available() else '\u2022 GPU Device Not Foun
d. Running on CPU')
```

Select the Hub/TF2 Module to Use

Hub modules for TF 1.x won't work here, please use one of the selections provided.

In [0]:
```
module_selection = ("mobilenet_v2", 224, 1280) #@param ["(\"mobilenet_v2\", 224, 1280)", "(\"in
ception_v3\", 299, 2048)"] {type:"raw", allow-input: true}
handle_base, pixels, FV_SIZE = module_selection
MODULE_HANDLE ="https://tfhub.dev/google/tf2-preview/{}/feature_vector/4".format(handle_base)
IMAGE_SIZE = (pixels, pixels)
print("Using {} with input size {} and output dimension {}".format(MODULE_HANDLE, IMAGE_SIZE, F
V_SIZE))
```

Data Preprocessing

Use TensorFlow Datasets (http://tensorflow.org/datasets) to load the cats and dogs dataset.

This `tfds` package is the easiest way to load pre-defined data. If you have your own data, and are interested in importing using it with TensorFlow see loading image data (../load_data/images.ipynb)

In [0]:
```
import tensorflow_datasets as tfds
tfds.disable_progress_bar()
```

The `tfds.load` method downloads and caches the data, and returns a `tf.data.Dataset` object. These objects provide powerful, efficient methods for manipulating data and piping it into your model.

Since `"cats_vs_dog"` doesn't define standard splits, use the subsplit feature to divide it into (train, validation, test) with 80%, 10%, 10% of the data respectively.

In [0]:
```
splits = tfds.Split.ALL.subsplit(weighted=(80, 10, 10))

# Go to the TensorFlow Dataset's website and search for the Rock, Paper, Scissors dataset and l
oad it here
splits, info = tfds.load( 'rock_paper_scissors:1.0.0',with_info=True,as_supervised=True,split=s
plits )   # YOUR CODE HERE

(train_examples, validation_examples, test_examples) = splits

num_examples = info.splits['train'].num_examples
num_classes = info.features['label'].num_classes

print("num_examples: " , num_examples)
print("num_classes: " , num_classes)
```

Format the Data

Use the `tf.image` module to format the images for the task.

Resize the images to a fixes input size, and rescale the input channels

In [0]:
```
def format_image(image, label):
    image = tf.image.resize(image, IMAGE_SIZE) / 255.0
    return  image, label
```

Now shuffle and batch the data

In [0]:
```
BATCH_SIZE = 32 #@param {type:"integer"}
```

In [0]:
```
# Prepare the examples by preprocessing the them and then batching them (and optionally prefetc
hing them)

# If you wish you can shuffle train set here
train_batches = train_examples.shuffle(num_examples // 4).map(format_image).batch(BATCH_SIZE).p
refetch(1)# YOUR CODE HERE

validation_batches = validation_examples.map(format_image).batch(BATCH_SIZE).prefetch(1)# YOUR
CODE HERE

test_batches = test_examples.map(format_image).batch(1)# YOUR CODE HERE
```

Inspect a batch

In [0]:
```
for image_batch, label_batch in train_batches.take(1):
    pass

image_batch.shape
```

Defining the Model

All it takes is to put a linear classifier on top of the `feature_extractor_layer` with the Hub module.

For speed, we start out with a non-trainable `feature_extractor_layer`, but you can also enable fine-tuning for greater accuracy.

In [0]:
```
do_fine_tuning = True #@param {type:"boolean"}
```

In [0]:
```
feature_extractor = hub.KerasLayer(MODULE_HANDLE,
                                   input_shape=IMAGE_SIZE + (3,),
                                   output_shape=[FV_SIZE],
                                   trainable=do_fine_tuning)
```

In [0]:
```
print("Building model with", MODULE_HANDLE)

model = tf.keras.Sequential([
        feature_extractor,
        tf.keras.layers.Dense(num_classes, activation='softmax')
])

model.summary()
```

In [0]:
```
#@title (Optional) Unfreeze some layers
NUM_LAYERS = 10 #@param {type:"slider", min:1, max:50, step:1}

if do_fine_tuning:
    feature_extractor.trainable = True

    for layer in model.layers[-NUM_LAYERS:]:
        layer.trainable = True

else:
    feature_extractor.trainable = False
```

Training the Model

In [0]:
```
if do_fine_tuning:
    model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.002, momentum=0.9),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])
else:
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
```

In [0]:
```
EPOCHS = 5

hist = model.fit(train_batches,
                 epochs=EPOCHS,
                 validation_data=validation_batches)
```

Export the Model

In [0]:
```
RPS_SAVED_MODEL = "rps_saved_model"
```

Export the SavedModel

In [0]:
```
# Use TensorFlow's SavedModel API to export the SavedModel from the trained Keras model
# YOUR CODE HERE
tf.saved_model.save(model,RPS_SAVED_MODEL)
```

In [0]:
```
%%bash -s $RPS_SAVED_MODEL
saved_model_cli show --dir $1 --tag_set serve --signature_def serving_default
```

In [0]:
```
loaded = tf.saved_model.load(RPS_SAVED_MODEL)
```

In [0]:
```
print(list(loaded.signatures.keys()))
infer = loaded.signatures["serving_default"]
print(infer.structured_input_signature)
print(infer.structured_outputs)
```

Convert Using TFLite's Converter

In [0]:
```
# Intialize the TFLite converter to load the SavedModel
# YOUR CODE HERE
converter = tf.lite.TFLiteConverter.from_saved_model(RPS_SAVED_MODEL)

# Set the optimization strategy for 'size' in the converter
# YOUR CODE HERE
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Use the tool to finally convert the model
# YOUR CODE HERE
tflite_model = converter.convert()
```

In [0]:
```
tflite_model_file = 'converted_model.tflite'

with open(tflite_model_file, "wb") as f:
    f.write(tflite_model)
```

Test the TFLite Model Using the Python Interpreter

In [0]:
```
# Load TFLite model and allocate tensors.
with open(tflite_model_file, 'rb') as fid:
    tflite_model = fid.read()

interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

input_index = interpreter.get_input_details()[0]["index"]
output_index = interpreter.get_output_details()[0]["index"]
```

In [0]:
```
# Gather results for the randomly sampled test images
predictions = []

test_labels, test_imgs = [], []
for img, label in tqdm(test_batches.take(10)):
    interpreter.set_tensor(input_index, img)
    interpreter.invoke()
    predictions.append(interpreter.get_tensor(output_index))

    test_labels.append(label.numpy()[0])
    test_imgs.append(img)
```

In [0]:
```
#@title Utility functions for plotting
# Utilities for plotting

class_names = ['rock', 'paper', 'scissors']

def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    img = np.squeeze(img)

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)

    print(type(predicted_label), type(true_label))

    if predicted_label == true_label:
        color = 'green'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                         100*np.max(predictions_array),
                                         class_names[true_label]), color=color)
```

In [0]:
```
#@title Visualize the outputs { run: "auto" }
index = 7 #@param {type:"slider", min:0, max:9, step:1}
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(index, predictions, test_labels, test_imgs)
plt.show()
```

Create a file to save the labels.

In [0]:
```
with open('labels.txt', 'w') as f:
    f.write('\n'.join(class_names))
```

If you are running this notebook in a Colab, you can run the cell below to download the model and labels to your local disk.

Note: If the files do not download when you run the cell, try running the cell a second time. Your browser might prompt you to allow multiple files to be downloaded.

In [0]:
```
try:
    from google.colab import files
    files.download('converted_model.tflite')
    files.download('labels.txt')
except:
    pass
```

Prepare the Test Images for Download (Optional)

This part involves downloading additional test images for the Mobile Apps only in case you need to try out more samples

In [0]:
```
!mkdir -p test_images
```

In [0]:
```
from PIL import Image

for index, (image, label) in enumerate(test_batches.take(50)):
    image = tf.cast(image * 255.0, tf.uint8)
    image = tf.squeeze(image).numpy()
    pil_image = Image.fromarray(image)
    pil_image.save('test_images/{}_{}.jpg'.format(class_names[label[0]], index))
```

In [0]:
```
!ls test_images
```

In [0]:
```
!zip -qq rps_test_images.zip -r test_images/
```

If you are running this notebook in a Colab, you can run the cell below to download the Zip file with the images to your local disk.

Note: If the Zip file does not download when you run the cell, try running the cell a second time.

In [0]:
```
try:
    files.download('rps_test_images.zip')
except:
    pass
```