



 Open in Colab

(https://colab.research.google.com/github/sergejhorvat/TensorFlow-Data-and-Deployment-Specialization/blob/master/Device-based%20Models%20with%20TensorFlow/Week%201/Examples/TFLite_Week1_Transfer_Learning.ipynb)

Copyright 2018 The TensorFlow Authors.

```
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.
```

Transfer Learning with TensorFlow Hub for TFLite

 Run in Google Colab (https://colab.research.google.com/github/Imoroney/dlaicourse/blob/master/TensorFlow%20Deployment/Course%202%20-%20TensorFlow%20Lite/Week%201/Examples/TFLite_Week1_Transfer_Learning.ipynb)	 View source on GitHub (https://github.com/Imoroney/dlaicourse/blob/master/TensorFlow%20Deployment/Course%202%20-%20TensorFlow%20Lite/Week%201/Examples/TFLite_Week1_Transfer_Learning.ipynb)
--	--

Setup

```
In [0]: !pip3 uninstall tensorflow
!pip3 install tf-nightly
try:
    %tensorflow_version 2.x
except:
    pass

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

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
tfds.disable_progress_bar()

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 Found. 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 [{"(\u201cmobilenet_v2\u201c", 224, 1280)", "(\u201cinception_v3\u201c", 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, FV_SIZE))
```

Data Preprocessing

Use [TensorFlow Datasets](http://tensorflow.org/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](#)).

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))

splits, info = tfds.load('cats_vs_dogs:2.0.1', with_info=True, as_supervised=True, split = splits)

(train_examples, validation_examples, test_examples) = splits

num_examples = info.splits['train'].num_examples
num_classes = info.features['label'].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]: train_batches = train_examples.shuffle(num_examples // 4).map(format_image).batch(BATCH_SIZE).prefetch(1)
validation_batches = validation_examples.map(format_image).batch(BATCH_SIZE).prefetch(1)
test_batches = test_examples.map(format_image).batch(1)
```

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 = False #@param {type:"boolean"}
```

Load TFHub Module

```
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]: CATS_VS_DOGS_SAVED_MODEL = "exp_saved_model"
```

Export the SavedModel

```
In [0]: tf.saved_model.save(model, CATS_VS_DOGS_SAVED_MODEL)
```

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

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

Load the TFLiteConverter with the SavedModel

```
In [0]: converter = tf.lite.TFLiteConverter.from_saved_model(CATS_VS_DOGS_SAVED_MODEL)
```

Post-Training Quantization

The simplest form of post-training quantization quantizes weights from floating point to 8-bits of precision. This technique is enabled as an option in the TensorFlow Lite converter. At inference, weights are converted from 8-bits of precision to floating point and computed using floating-point kernels. This conversion is done once and cached to reduce latency.

To further improve latency, hybrid operators dynamically quantize activations to 8-bits and perform computations with 8-bit weights and activations. This optimization provides latencies close to fully fixed-point inference. However, the outputs are still stored using floating point, so that the speedup with hybrid ops is less than a full fixed-point computation.

```
In [0]: converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

Post-Training Integer Quantization

We can get further latency improvements, reductions in peak memory usage, and access to integer only hardware accelerators by making sure all model math is quantized. To do this, we need to measure the dynamic range of activations and inputs with a representative data set. You can simply create an input data generator and provide it to our converter.

```
In [0]: def representative_data_gen():
        for input_value, _ in test_batches.take(100):
            yield [input_value]
```

```
In [0]: converter.representative_dataset = representative_data_gen
```

The resulting model will be fully quantized but still take float input and output for convenience.

Ops that do not have quantized implementations will automatically be left in floating point. This allows conversion to occur smoothly but may restrict deployment to accelerators that support float.

Full Integer Quantization

To require the converter to only output integer operations, one can specify:

```
In [0]: converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
```

Finally convert the model

```
In [0]: tf_lite_model = converter.convert()
        tf_lite_model_file = 'converted_model.tflite'

        with open(tf_lite_model_file, "wb") as f:
            f.write(tf_lite_model)
```

Test the TFLite Model Using the Python Interpreter

```
In [0]: # Load TFLite model and allocate tensors.

        interpreter = tf.lite.Interpreter(model_path=tf_lite_model_file)
        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 = ['cat', 'dog']

        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)

            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)
```

NOTE: Colab runs on server CPUs. At the time of writing this, TensorFlow Lite doesn't have super optimized server CPU kernels. For this reason post-training full-integer quantized models may be slower here than the other kinds of optimized models. But for mobile CPUs, considerable speedup can be observed.

```
In [0]: #@title Visualize the outputs { run: "auto" }
        index = 0 #@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]: labels = ['cat', 'dog']

        with open('labels.txt', 'w') as f:
            f.write('\n'.join(labels))
```

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 cats_vs_dogs_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('cats_vs_dogs_test_images.zip')
except:
    pass
```