l06c01_tensorflow_hub_and_transfer_learning

September 14, 2021

Run in Google Colab

View source on GitHub

1 TensorFlow Hub and Transfer Learning

TensorFlow Hub is an online repository of already trained TensorFlow models that you can use. These models can either be used as is, or they can be used for Transfer Learning.

Transfer learning is a process where you take an existing trained model, and extend it to do additional work. This involves leaving the bulk of the model unchanged, while adding and retraining the final layers, in order to get a different set of possible outputs.

In this Colab we will do both.

Here, you can see all the models available in TensorFlow Module Hub.

1.1 Concepts that will be covered in this Colab

1. Use a TensorFlow Hub model for prediction.
2. Use a TensorFlow Hub model for Dogs vs. Cats dataset.
3. Do simple transfer learning with TensorFlow Hub.

Before starting this Colab, you should reset the Colab environment by selecting `Runtime -> Reset all runtimes...` from menu above.

2 Imports

Some normal imports we've seen before. The new one is importing tensorflow_hub which was installed above, and which this Colab will make heavy use of.

[2]:
```
import tensorflow as tf
```

[3]:
```
import matplotlib.pylab as plt

import tensorflow_hub as hub
import tensorflow_datasets as tfds

from tensorflow.keras import layers
```

[4]:
```
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

3 Part 1: Use a TensorFlow Hub MobileNet for prediction

In this part of the Colab, we'll take a trained model, load it into to Keras, and try it out.

The model that we'll use is MobileNet v2 (but any model from tf2 compatible image classifier URL from tfhub.dev would work).

3.1 Download the classifier

Download the MobileNet model and create a Keras model from it. MobileNet is expecting images of 224 $\times$ 224 pixels, in 3 color channels (RGB).

[5]:
```
CLASSIFIER_URL ="https://tfhub.dev/google/tf2-preview/mobilenet_v2/
 ↪classification/2"
IMAGE_RES = 224

model = tf.keras.Sequential([
    hub.KerasLayer(CLASSIFIER_URL, input_shape=(IMAGE_RES, IMAGE_RES, 3))
])
```

3.2 Run it on a single image

MobileNet has been trained on the ImageNet dataset. ImageNet has 1000 different output classes, and one of them is military uniforms. Let's get an image containing a military uniform that is not part of ImageNet, and see if our model can predict that it is a military uniform.

[6]:
```
import numpy as np
import PIL.Image as Image

grace_hopper = tf.keras.utils.get_file('image.jpg','https://storage.googleapis.
 ↪com/download.tensorflow.org/example_images/grace_hopper.jpg')
```

```
grace_hopper = Image.open(grace_hopper).resize((IMAGE_RES, IMAGE_RES))
grace_hopper
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.org/exa
mple_images/grace_hopper.jpg
65536/61306 [================================] - 0s 0us/step
73728/61306 [====================================] - 0s 0us/step
```

[6]:

[7]:
```
grace_hopper = np.array(grace_hopper)/255.0
grace_hopper.shape
```

[7]:
```
(224, 224, 3)
```

Remember, models always want a batch of images to process. So here, we add a batch dimension, and pass the image to the model for prediction.

[8]:
```
result = model.predict(grace_hopper[np.newaxis, ...])
result.shape
```

[8]:
```
(1, 1001)
```

The result is a 1001 element vector of logits, rating the probability of each class for the image.

So the top class ID can be found with argmax. But how can we know what class this actually is and in particular if that class ID in the ImageNet dataset denotes a military uniform or something else?

[9]:
```
predicted_class = np.argmax(result[0], axis=-1)
predicted_class
```

[9]: 653

3.3 Decode the predictions

To see what our predicted_class is in the ImageNet dataset, download the ImageNet labels and fetch the row that the model predicted.

```
[10]: labels_path = tf.keras.utils.get_file('ImageNetLabels.txt','https://storage.
       ↪googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt')
      imagenet_labels = np.array(open(labels_path).read().splitlines())

      plt.imshow(grace_hopper)
      plt.axis('off')
      predicted_class_name = imagenet_labels[predicted_class]
      _ = plt.title("Prediction: " + predicted_class_name.title())
```

```
Downloading data from
https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt
16384/10484 [==============================================] - 0s 0us/step
24576/10484
[======================================================================] - 0s
0us/step
```

Prediction: Military Uniform

Bingo. Our model correctly predicted military uniform!

4 Part 2: Use a TensorFlow Hub models for the Cats vs. Dogs dataset

Now we'll use the full MobileNet model and see how it can perform on the Dogs vs. Cats dataset.

4.1 Dataset

We can use TensorFlow Datasets to load the Dogs vs Cats dataset.

[11]:
```
(train_examples, validation_examples), info = tfds.load(
    'cats_vs_dogs',
    with_info=True,
    as_supervised=True,
    split=['train[:80%]', 'train[80%:]'],
)

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

```
Downloading and preparing dataset cats_vs_dogs/4.0.0 (download: 786.68 MiB,
generated: Unknown size, total: 786.68 MiB) to
/root/tensorflow_datasets/cats_vs_dogs/4.0.0…

Dl Completed…: 0 url [00:00, ? url/s]

Dl Size…: 0 MiB [00:00, ? MiB/s]



0 examples [00:00, ? examples/s]

WARNING:absl:1738 images were corrupted and were skipped

Shuffling and writing examples to
/root/tensorflow_datasets/cats_vs_dogs/4.0.0.incomplete6UDIR9/cats_vs_dogs-
train.tfrecord

  0%|          | 0/23262 [00:00<?, ? examples/s]

Dataset cats_vs_dogs downloaded and prepared to
/root/tensorflow_datasets/cats_vs_dogs/4.0.0. Subsequent calls will reuse this
data.
```

The images in the Dogs vs. Cats dataset are not all the same size.

[12]:
```
for i, example_image in enumerate(train_examples.take(3)):
  print("Image {} shape: {}".format(i+1, example_image[0].shape))
```

```
Image 1 shape: (262, 350, 3)
Image 2 shape: (409, 336, 3)
Image 3 shape: (493, 500, 3)
```

So we need to reformat all images to the resolution expected by MobileNet (224, 224).

The `.repeat()` and `steps_per_epoch` here is not required, but saves ~15s per epoch, since the shuffle-buffer only has to cold-start once.

[13]:
```
def format_image(image, label):
  image = tf.image.resize(image, (IMAGE_RES, IMAGE_RES))/255.0
  return image, label

BATCH_SIZE = 32

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)
```

4.2 Run the classifier on a batch of images

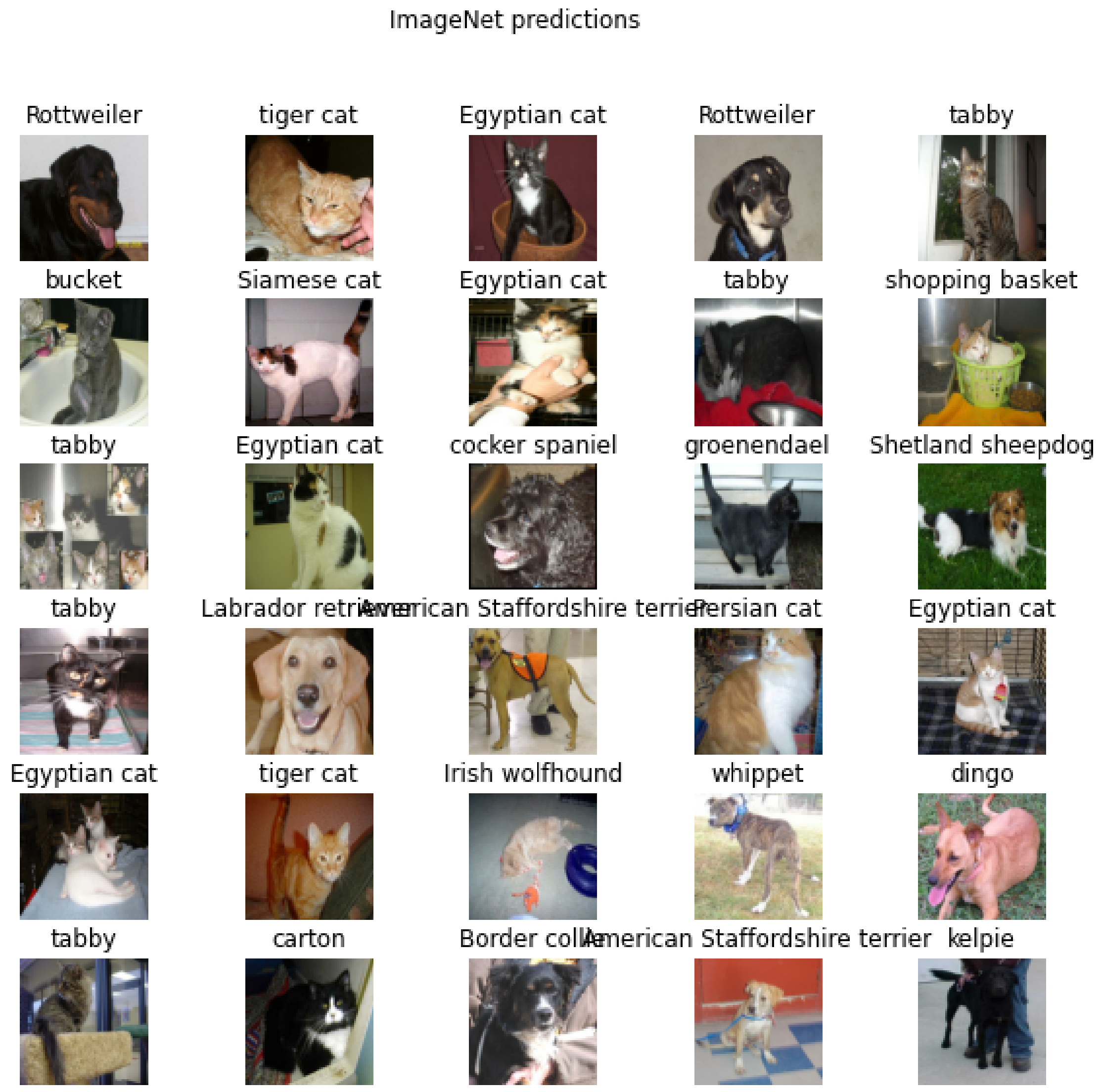
Remember our `model` object is still the full MobileNet model trained on ImageNet, so it has 1000 possible output classes. ImageNet has a lot of dogs and cats in it, so let's see if it can predict the images in our Dogs vs. Cats dataset.

[14]:
```
image_batch, label_batch = next(iter(train_batches.take(1)))
image_batch = image_batch.numpy()
label_batch = label_batch.numpy()

result_batch = model.predict(image_batch)

predicted_class_names = imagenet_labels[np.argmax(result_batch, axis=-1)]
predicted_class_names
```

[14]:
```
array(['Rottweiler', 'tiger cat', 'Egyptian cat', 'Rottweiler', 'tabby',
       'bucket', 'Siamese cat', 'Egyptian cat', 'tabby',
       'shopping basket', 'tabby', 'Egyptian cat', 'cocker spaniel',
       'groenendael', 'Shetland sheepdog', 'tabby', 'Labrador retriever',
       'American Staffordshire terrier', 'Persian cat', 'Egyptian cat',
       'Egyptian cat', 'tiger cat', 'Irish wolfhound', 'whippet', 'dingo',
       'tabby', 'carton', 'Border collie',
       'American Staffordshire terrier', 'kelpie', 'tabby', 'tabby'],
      dtype='<U30')
```

The labels seem to match names of Dogs and Cats. Let's now plot the images from our Dogs vs Cats dataset and put the ImageNet labels next to them.

[15]:
```
plt.figure(figsize=(10,9))
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.subplots_adjust(hspace = 0.3)
  plt.imshow(image_batch[n])
  plt.title(predicted_class_names[n])
  plt.axis('off')
_ = plt.suptitle("ImageNet predictions")
```



[15]:
```

```

5 Part 3: Do simple transfer learning with TensorFlow Hub

Let's now use TensorFlow Hub to do Transfer Learning.

With transfer learning we reuse parts of an already trained model and change the final layer, or several layers, of the model, and then retrain those layers on our own dataset.

In addition to complete models, TensorFlow Hub also distributes models without the last classification layer. These can be used to easily do transfer learning. We will continue using MobileNet v2 because in later parts of this course, we will take this model and deploy it on a mobile device using TensorFlow Lite. Any image feature vector URL from tfhub.dev would work here.

We'll also continue to use the Dogs vs Cats dataset, so we will be able to compare the performance of this model against the ones we created from scratch earlier.

Note that we're calling the partial model from TensorFlow Hub (without the final classification layer) a `feature_extractor`. The reasoning for this term is that it will take the input all the way to a layer containing a number of features. So it has done the bulk of the work in identifying the content of an image, except for creating the final probability distribution. That is, it has extracted the features of the image.

[16]:
```
URL = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/2"
feature_extractor = hub.KerasLayer(URL,
                                   input_shape=(IMAGE_RES, IMAGE_RES,3))
```

Let's run a batch of images through this, and see the final shape. 32 is the number of images, and 1280 is the number of neurons in the last layer of the partial model from TensorFlow Hub.

[17]:
```
feature_batch = feature_extractor(image_batch)
print(feature_batch.shape)
```

```
(32, 1280)
```

Freeze the variables in the feature extractor layer, so that the training only modifies the final classifier layer.

[18]:
```
feature_extractor.trainable = False
```

5.1 Attach a classification head

Now wrap the hub layer in a `tf.keras.Sequential` model, and add a new classification layer.

[19]:
```
model = tf.keras.Sequential([
  feature_extractor,
  layers.Dense(2)
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,562
Non-trainable params: 2,257,984
_________________________________________________________________
```

5.2 Train the model

We now train this model like any other, by first calling `compile` followed by `fit`.

[20]:
```python
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])

EPOCHS = 6
history = model.fit(train_batches,
                    epochs=EPOCHS,
                    validation_data=validation_batches)
```

```
Epoch 1/6
582/582 [==============================] - 66s 100ms/step - loss: 0.0572 -
accuracy: 0.9809 - val_loss: 0.0339 - val_accuracy: 0.9880
Epoch 2/6
582/582 [==============================] - 61s 97ms/step - loss: 0.0297 -
accuracy: 0.9898 - val_loss: 0.0312 - val_accuracy: 0.9893
Epoch 3/6
582/582 [==============================] - 61s 97ms/step - loss: 0.0252 -
accuracy: 0.9919 - val_loss: 0.0313 - val_accuracy: 0.9886
Epoch 4/6
582/582 [==============================] - 61s 97ms/step - loss: 0.0224 -
accuracy: 0.9925 - val_loss: 0.0307 - val_accuracy: 0.9901
Epoch 5/6
582/582 [==============================] - 61s 97ms/step - loss: 0.0192 -
accuracy: 0.9939 - val_loss: 0.0317 - val_accuracy: 0.9899
Epoch 6/6
582/582 [==============================] - 62s 97ms/step - loss: 0.0170 -
accuracy: 0.9947 - val_loss: 0.0327 - val_accuracy: 0.9897
```

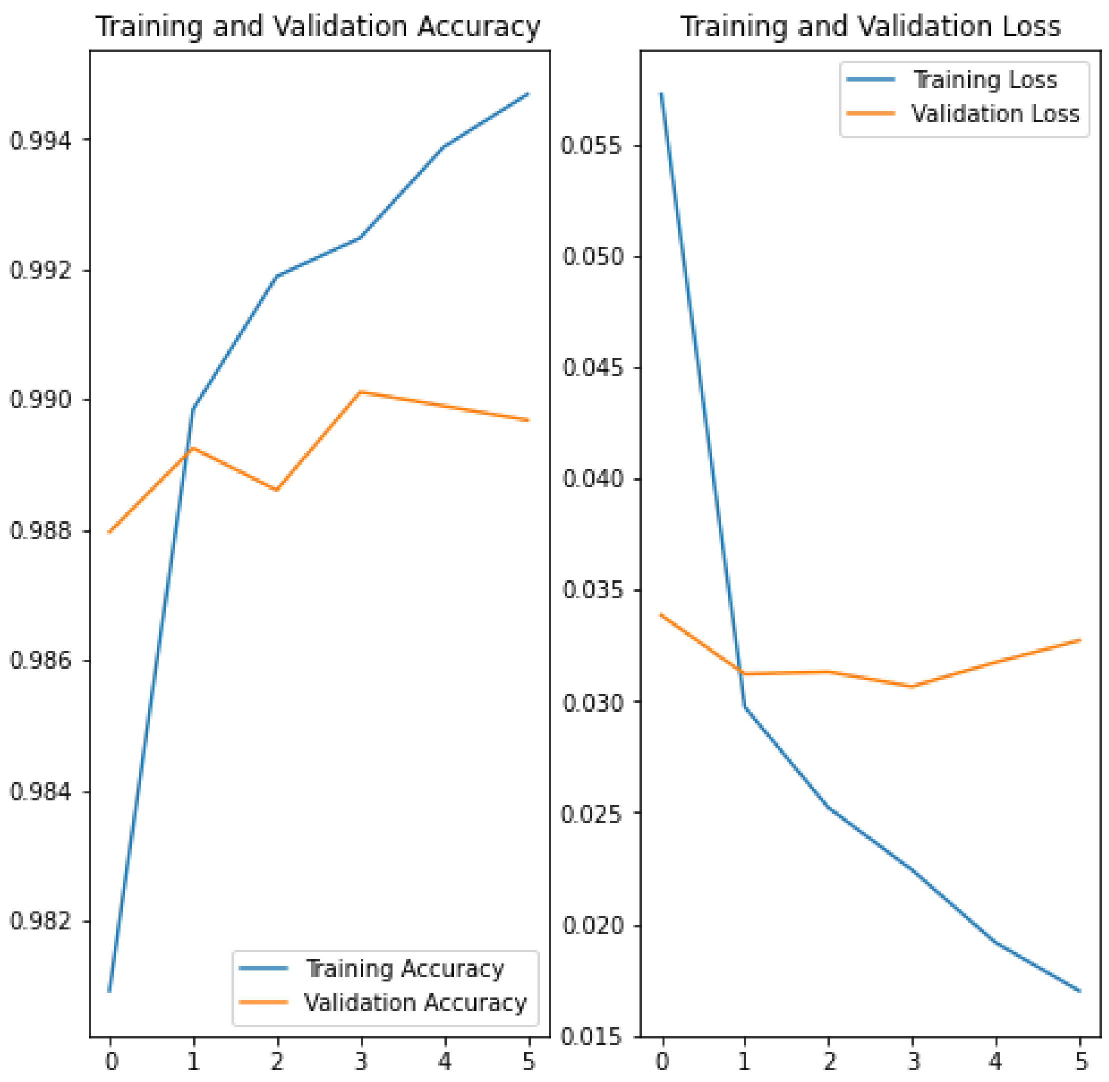
You can see we get ~97% validation accuracy, which is absolutely awesome. This is a huge improvement over the model we created in the previous lesson, where we were able to get ~83% accuracy. The reason for this difference is that MobileNet was carefully designed over a long time by experts, then trained on a massive dataset (ImageNet).

Although not equivalent to TensorFlow Hub, you can check out how to create MobileNet in Keras here.

Let's plot the training and validation accuracy/loss graphs.

[21]:
```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



What is a bit curious here is that validation performance is better than training performance, right from the start to the end of execution.

One reason for this is that validation performance is measured at the end of the epoch, but training performance is the average values across the epoch.

The bigger reason though is that we're reusing a large part of MobileNet which is already trained on Dogs and Cats images. While doing training, the network is still performing image augmentation on the training images, but not on the validation dataset. This means the training images may be harder to classify compared to the normal images in the validation dataset.

5.3 Check the predictions

To redo the plot from before, first get the ordered list of class names.

[22]:
```
class_names = np.array(info.features['label'].names)
class_names
```

[22]:
```
array(['cat', 'dog'], dtype='<U3')
```

Run the image batch through the model and convert the indices to class names.

[23]:
```
predicted_batch = model.predict(image_batch)
predicted_batch = tf.squeeze(predicted_batch).numpy()
predicted_ids = np.argmax(predicted_batch, axis=-1)
predicted_class_names = class_names[predicted_ids]
predicted_class_names
```

[23]:
```
array(['dog', 'cat', 'cat', 'dog', 'cat', 'cat', 'cat', 'cat', 'cat',
       'cat', 'cat', 'cat', 'dog', 'cat', 'dog', 'cat', 'dog', 'dog',
       'cat', 'cat', 'cat', 'cat', 'cat', 'dog', 'dog', 'cat', 'cat',
       'dog', 'dog', 'dog', 'cat', 'cat'], dtype='<U3')
```

Let's look at the true labels and predicted ones.

[24]:
```
print("Labels: ", label_batch)
print("Predicted labels: ", predicted_ids)
```

```
Labels:  [1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 0 0]
Predicted labels:  [1 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 0 1 1 1
0 0]
```

[25]:
```
plt.figure(figsize=(10,9))
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.subplots_adjust(hspace = 0.3)
  plt.imshow(image_batch[n])
  color = "blue" if predicted_ids[n] == label_batch[n] else "red"
  plt.title(predicted_class_names[n].title(), color=color)
  plt.axis('off')
_ = plt.suptitle("Model predictions (blue: correct, red: incorrect)")
```

Model predictions (blue: correct, red: incorrect)