Run in Google Colab View source on GitHub

# TensorFlow Hub

TensorFlow Hub is an online repository of already trained TensorFlow models that you can use. These models ca
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
the model unchanged, while adding and retraining the final layers, in order to get a different set of possible outpu

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

Before starting this Colab, you should reset the Colab environment by selecting `Runtime -> Reset all runti`

# Imports

This Colab will require us to use some things which are not yet in official releases of TensorFlow. So below, we're
version of TensorFlow as well as TensorFlow Hub.

This will switch your installation of TensorFlow in Colab to this TensorFlow version. Once you are finished with th
batch to the latest stable release of TensorFlow by doing selecting `Runtime -> Reset all runtimes...` in th
reset the Colab environment to its original state.

```
!pip install tf-nightly-gpu
!pip install "tensorflow_hub==0.4.0"
!pip install -U tensorflow_datasets
```

```
Requirement already satisfied: tf-nightly-gpu in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: tf-estimator-nightly in /usr/local/lib/python3.6/dist-
Requirement already satisfied: tb-nightly<1.15.0a0,>=1.14.0a0 in /usr/local/lib/pytho
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.6
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: numpy<2.0,>=1.14.5 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: keras-applications>=1.0.8 in /usr/local/lib/python3.6/
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from k
Requirement already satisfied: tensorflow_hub==0.4.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: protobuf>=3.4.0 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: numpy>=1.12.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (
Requirement already up-to-date: tensorflow_datasets in /usr/local/lib/python3.6/dist-
Requirement already satisfied, skipping upgrade: psutil in /usr/local/lib/python3.6/d
Requirement already satisfied, skipping upgrade: protobuf>=3.6.1 in /usr/local/lib/py
Requirement already satisfied, skipping upgrade: requests in /usr/local/lib/python3.6
Requirement already satisfied, skipping upgrade: dill in /usr/local/lib/python3.6/dis
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.6/dist
Requirement already satisfied, skipping upgrade: tqdm in /usr/local/lib/python3.6/dis
Requirement already satisfied, skipping upgrade: wrapt in /usr/local/lib/python3.6/di
Requirement already satisfied, skipping upgrade: absl-py in /usr/local/lib/python3.6/
Requirement already satisfied, skipping upgrade: promise in /usr/local/lib/python3.6/
Requirement already satisfied, skipping upgrade: numpy in /usr/local/lib/python3.6/di
Requirement already satisfied, skipping upgrade: future in /usr/local/lib/python3.6/d
Requirement already satisfied, skipping upgrade: termcolor in /usr/local/lib/python3.
Requirement already satisfied, skipping upgrade: tensorflow-metadata in /usr/local/li
```

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

```
from __future__ import absolute_import, division, print_function, unicode_literals

import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
tf.enable_eager_execution()

import tensorflow_hub as hub
import tensorflow_datasets as tfds

from tensorflow.keras import layers
```

```
import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)
```

# TODO: Download the Flowers Dataset using TensorFlow

In the cell below you will download the Flowers dataset using TensorFlow Datasets. If you look at the TensorFlow
you will see that the name of the Flowers dataset is `tf_flowers`. You can also see that this dataset is only split
will therefore have to use `tfds.splits` to split this training set into to a `training_set` and a `validation_set`.
that 70 corresponds to the `training_set` and 30 to the `validation_set`. Then load the `tf_flowers` dataset u
the `tfds.load` function uses the all the parameters you need, and also make sure it returns the dataset info, so
about the datasets.

```
splits = tfds.Split.TRAIN.subsplit([70, 30])

(training_set, validation_set), dataset_info = tfds.load('tf_flowers', with_info=True, as_supervi
```

# TODO: Print Information about the Flowers Dataset

Now that you have downloaded the dataset, use the dataset info to print the number of classes in the dataset, ar
that counts how many images we have in the training and validation sets.

```
num_classes = dataset_info.features['label'].num_classes

num_training_examples = 0
num_validation_examples = 0

for example in training_set:
  num_training_examples += 1

for example in validation_set:
  num_validation_examples += 1

print('Total Number of Classes: {}'.format(num_classes))
print('Total Number of Training Images: {}'.format(num_training_examples))
print('Total Number of Validation Images: {} \n'.format(num_validation_examples))
```

```
Total Number of Classes: 5
Total Number of Training Images: 2590
Total Number of Validation Images: 1080
```

The images in the Flowers dataset are not all the same size.

```
for i, example in enumerate(training_set.take(5)):
  print('Image {} shape: {} label: {}'.format(i+1, example[0].shape, example[1]))
```

```
Image 1 shape: (335, 500, 3) label: 2
Image 2 shape: (257, 320, 3) label: 0
Image 3 shape: (213, 320, 3) label: 0
Image 4 shape: (213, 320, 3) label: 2
Image 5 shape: (240, 320, 3) label: 0
```

# TODO: Reformat Images and Create Batches

In the cell below create a function that reformats all images to the resolution expected by MobileNet v2 (224, 22
The function should take in an `image` and a `label` as arguments and should return the new `image` and correspo
create training and validation batches of size `32`.

```
IMAGE_RES = 224

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

BATCH_SIZE = 32

train_batches = training_set.shuffle(num_training_examples//4).map(format_image).batch(BATCH_SIZE

validation_batches = validation_set.map(format_image).batch(BATCH_SIZE).prefetch(1)
```

# Do Simple Transfer Learning with TensorFlow Hub

Let's now use TensorFlow Hub to do Transfer Learning. Remember, in transfer learning we reuse parts of an alre
change the final layer, or several layers, of the model, and then retrain those layers on our own dataset.

### TODO: Create a Feature Extractor

In the cell below create a `feature_extractor` using MobileNet v2. Remember that the partial model from Tens
final classification layer) is called a feature vector. Go to the [TensorFlow Hub documentation] to see a list of avai
on the `tf2-preview/mobilenet_v2/feature_vector`. Read the docuemntation and get the corresponding `URL`
feature vector. Finally, ceate a `feature_extractor` by using `hub.KerasLayer` with the correct `input_shape` pa

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

### TODO: Freeze the Pre-Trained Model

In the cell below freeze the variables in the feature extractor layer, so that the training only modifies the final clas

```
feature_extractor.trainable = False
```

### TODO: Attach a classification head

In the cell below create a `tf.keras.Sequential` model, and add the pre-trained model and the new classificatic
classification layer must have the same number of classes as our Flowers dataset. Finally print a sumary of the

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

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 5)                 6405
=================================================================
Total params: 2,264,389
Trainable params: 6,405
Non-trainable params: 2,257,984
_________________________________________________________________
```

### TODO: Train the model

In the cell bellow train this model like any other, by first calling `compile` and then followed by `fit`. Make sure you
when applying both methods. Train the model for only 6 epochs.

```
model.compile(
  optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

EPOCHS = 6

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

```
Epoch 1/6
81/81 [==============================] - 20s 248ms/step - loss: 0.9136 - acc: 0.6811
Epoch 2/6
81/81 [==============================] - 9s 113ms/step - loss: 0.5218 - acc: 0.8483 -
Epoch 3/6
81/81 [==============================] - 9s 114ms/step - loss: 0.4321 - acc: 0.8857 -
Epoch 4/6
81/81 [==============================] - 9s 114ms/step - loss: 0.3792 - acc: 0.9054 -
Epoch 5/6
81/81 [==============================] - 9s 114ms/step - loss: 0.3414 - acc: 0.9208 -
Epoch 6/6
81/81 [==============================] - 9s 114ms/step - loss: 0.3119 - acc: 0.9274 -
```

You can see we get ~88% validation accuracy with only 6 epochs of training, which is absolutely awesome. This
the model we created in the previous lesson, where we were able to get ~76% accuracy with 80 epochs of trainir
difference is that MobileNet v2 was carefully designed over a long time by experts, then trained on a massive da

# TODO: Plot Training and Validation Graphs.

In the cell below, plot the training and validation accuracy/loss graphs.

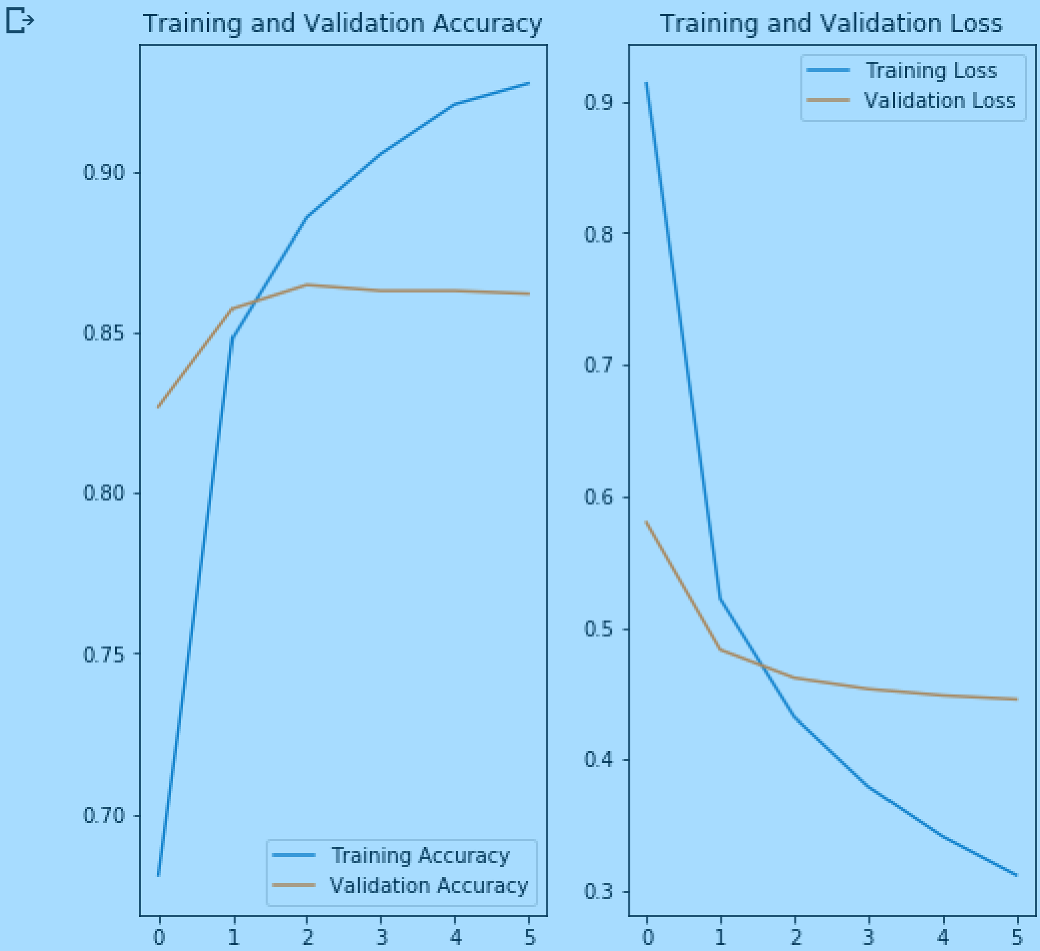
```
acc = history.history['acc']
val_acc = history.history['val_acc']

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

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

The bigger reason though is that we're reusing a large part of MobileNet which is already trained on Flower imag

# TODO: Check Predictions

In the cell below get the label names from the dataset info and convert them into a NumPy array. Print the array correct label names.

```
class_names = np.array(dataset_info.features['label'].names)
```

```
print(class_names)
```

```
['dandelion' 'daisy' 'tulips' 'sunflowers' 'roses']
```

### TODO: Create an Image Batch and Make Predictions

In the cell below, use the `next()` fucntion to create an `image_batch` and its corresponding `label_batch`. Conve and `label_batch` to numpy arrays using the `.numpy()` method. Then use the `.predict()` method to run the im model and make predictions. Then use the `np.argmax()` function to get the indices of the best prediction for ea the indices of the best predcitions to class names.

```
image_batch, label_batch = next(iter(train_batches))


image_batch = image_batch.numpy()
label_batch = label_batch.numpy()

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]

print(predicted_class_names)
```

```
['dandelion' 'tulips' 'sunflowers' 'tulips' 'daisy' 'sunflowers' 'daisy'
 'tulips' 'dandelion' 'tulips' 'roses' 'tulips' 'roses' 'tulips' 'roses'
 'sunflowers' 'roses' 'dandelion' 'tulips' 'roses' 'daisy' 'sunflowers'
 'dandelion' 'daisy' 'sunflowers' 'tulips' 'daisy' 'sunflowers'
 'sunflowers' 'tulips' 'dandelion' 'tulips']
```

### TODO: Print True Labels and Predicted Indices

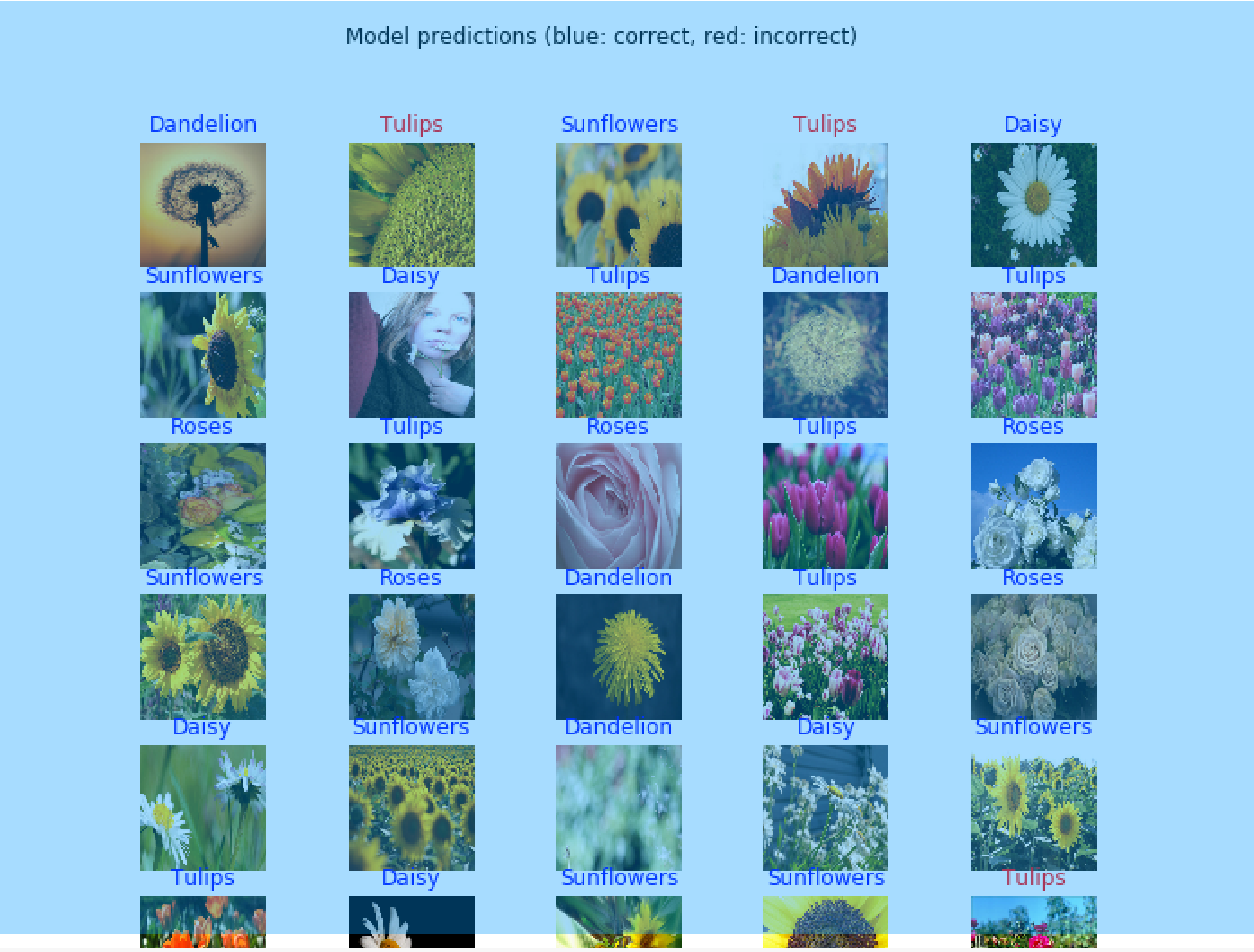
In the cell below, print the the true labels and the indices of predicted labels.

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

```
Labels:  [0 3 3 3 1 3 1 2 0 2 4 2 4 2 4 3 4 0 2 4 1 3 0 1 3 2 1 3 3 4 0 2]
Predicted labels:  [0 2 3 2 1 3 1 2 0 2 4 2 4 2 4 3 4 0 2 4 1 3 0 1 3 2 1 3 3 2 0 2]
```

# Plot Model Predictions

```
plt.figure(figsize=(10,9))
for n in range(30):
  plt.subplot(6,5,n+1)
  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)
Dandelion
Tulips
Sunflowers
Tulips
Daisy
Sunflowers
Daisy
Tulips
Dandelion
Tulips
Roses
Tulips
Roses
Tulips
Roses
Sunflowers
Roses
Dandelion
Tulips
Roses
Daisy
Sunflowers
Dandelion
Daisy
Sunflowers
Tulips
Daisy
Sunflowers
Sunflowers
Tulips