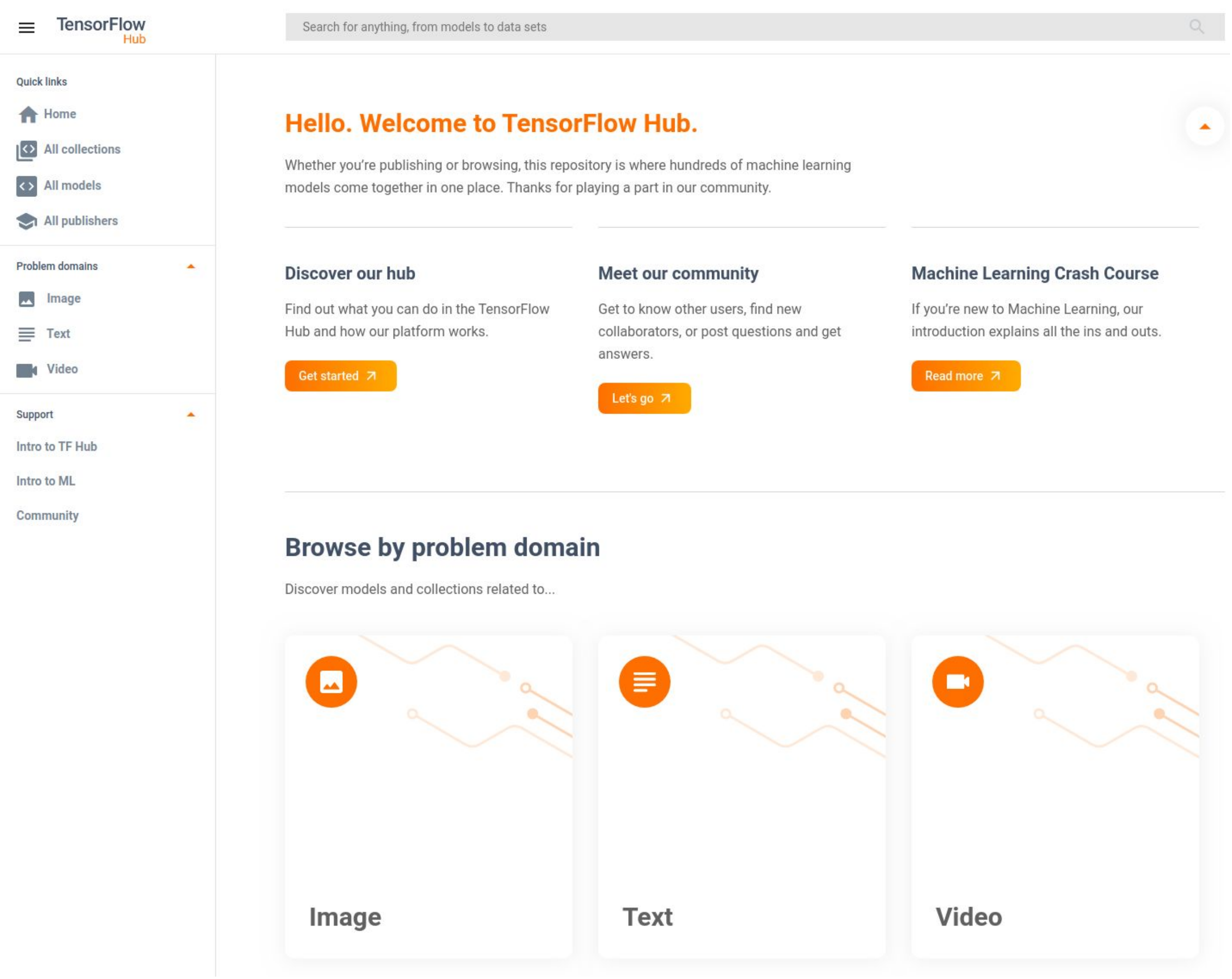
Copyright Notice

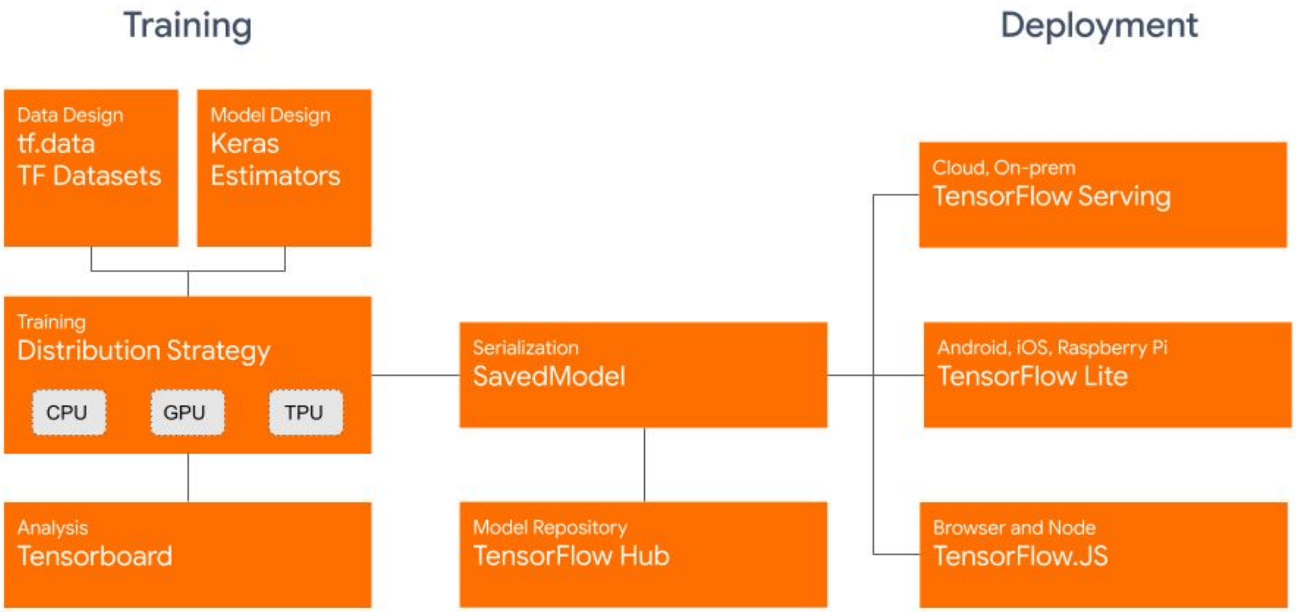
tfhub.dev


TensorFlow Hub
Search for anything, from models to data sets
Quick links
Home
All collections
All models
All publishers
Problem domains
Image
Text
Video
Support
Intro to TF Hub
Intro to ML
Community
Hello. Welcome to TensorFlow Hub.
Whether you're publishing or browsing, this repository is where hundreds of machine learning models come together in one place. Thanks for playing a part in our community.
Discover our hub
Find out what you can do in the TensorFlow Hub and how our platform works.
Get started
Meet our community
Get to know other users, find new collaborators, or post questions and get answers.
Let's go
Machine Learning Crash Course
If you're new to Machine Learning, our introduction explains all the ins and outs.
Read more
Browse by problem domain
Discover models and collections related to...
Image
Text
Video

TensorFlow Hub

- An extensive library of existing modules
- Well-tested modules designed for better accuracy
- Easy to use and integrate with your model APIs

Where does Hub fit in?



Where does Hub fit in?

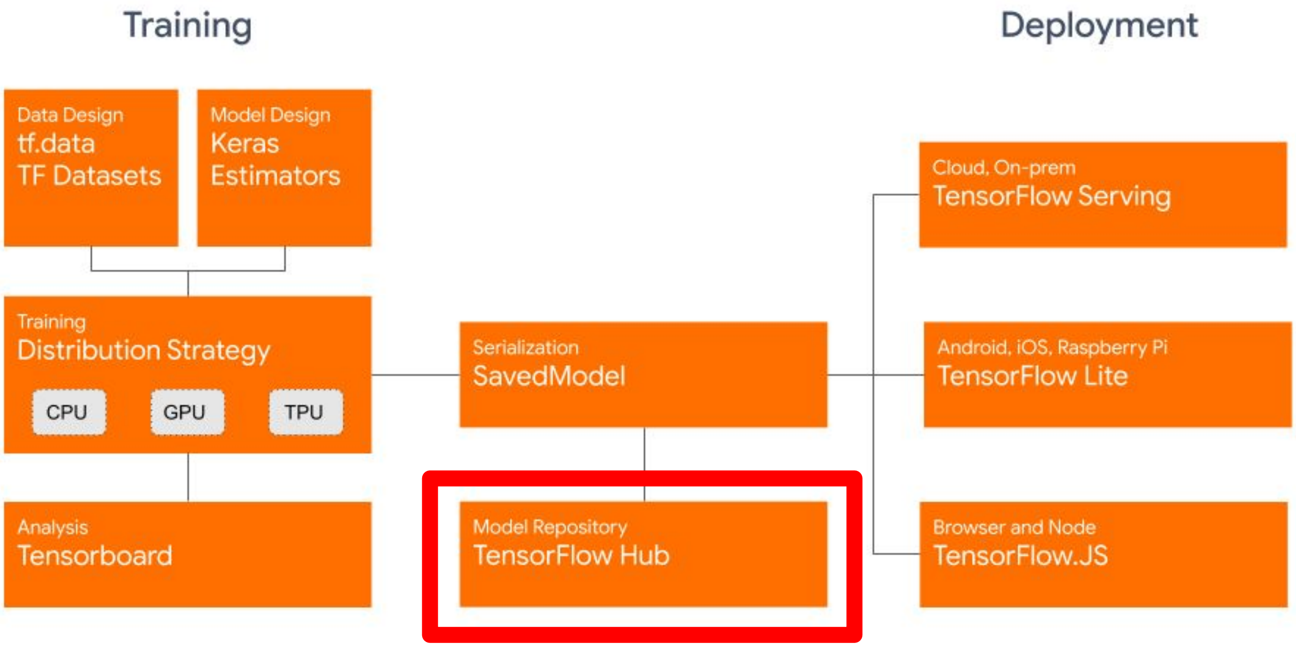


Image Credits

Who publishes the modules?

BigGAN

Inflated 3D

Wide ResNets

MobileNet

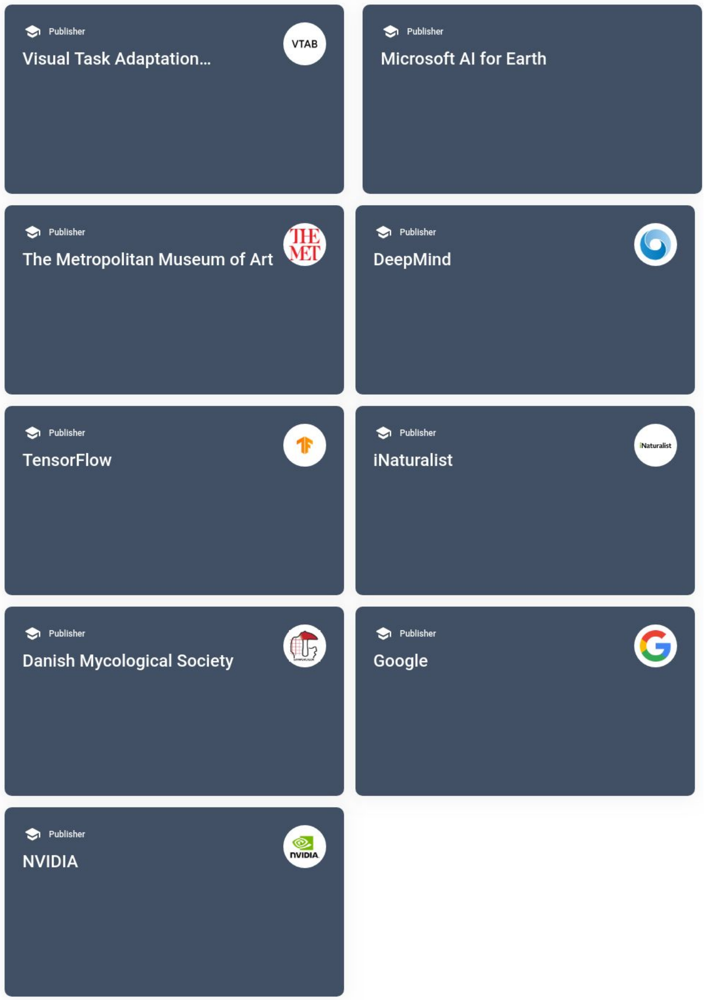
Inception

NASNet

TF-GAN

Image Credits

https://tfhub.dev/s?subtype=publisher



Transfer Learning with TF Hub

- Train models with less data
- Improve generalization
- Speed up training

Problem domains

Text

Embedding

Image

Classification

Feature vector

Augmentation

Object Detection

Generator

Style Transfer

Video

Classification

Installation

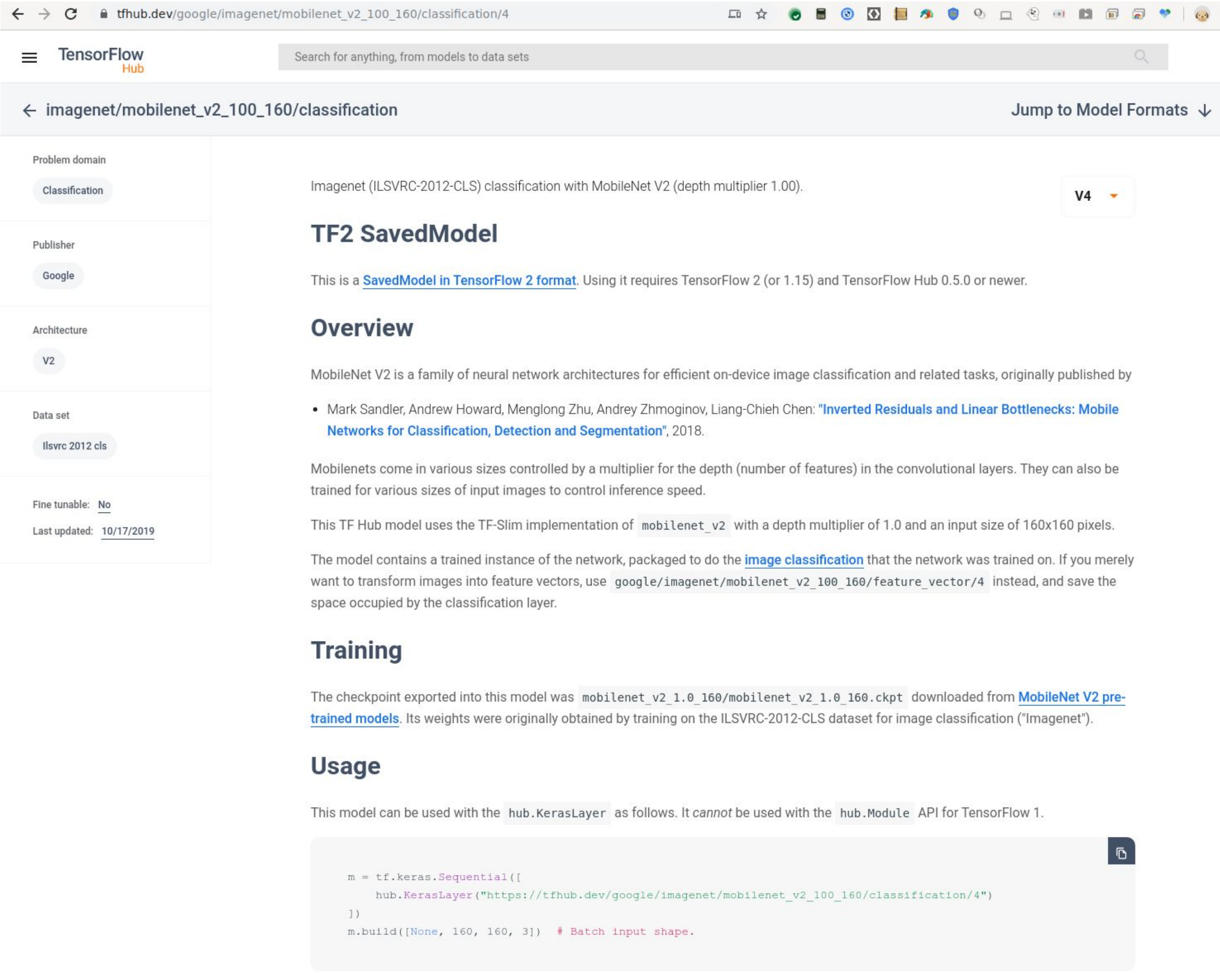
```
pip install tensorflow_hub
```

```python
import tensorflow_hub as hub
```

Loading a module

```
MODULE_HANDLE = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4'
module = hub.load(MODULE_HANDLE) # Load a the MobileNet image classification module
```

https://tfhub.dev/google/imagenet/mobilenet_v2_100_160/classification/4



tfhub.dev/google/imagenet/mobilenet_v2_100_160/classification/4

TensorFlow Hub

Search for anything, from models to data sets

imagenet/mobilenet_v2_100_160/classification

Jump to Model Formats

Problem domain

Classification

Publisher

Google

Architecture

V2

Data set

Ilsvrc 2012 cls

Fine tunable: No

Last updated: 10/17/2019

Imagenet (ILSVRC-2012-CLS) classification with MobileNet V2 (depth multiplier 1.00).

V4

TF2 SavedModel

This is a SavedModel in TensorFlow 2 format. Using it requires TensorFlow 2 (or 1.15) and TensorFlow Hub 0.5.0 or newer.

Overview

MobileNet V2 is a family of neural network architectures for efficient on-device image classification and related tasks, originally published by

- Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation", 2018.

Mobilenets come in various sizes controlled by a multiplier for the depth (number of features) in the convolutional layers. They can also be trained for various sizes of input images to control inference speed.

This TF Hub model uses the TF-Slim implementation of `mobilenet_v2` with a depth multiplier of 1.0 and an input size of 160x160 pixels.

The model contains a trained instance of the network, packaged to do the image classification that the network was trained on. If you merely want to transform images into feature vectors, use `google/imagenet/mobilenet_v2_100_160/feature_vector/4` instead, and save the space occupied by the classification layer.

Training

The checkpoint exported into this model was `mobilenet_v2_1.0_160/mobilenet_v2_1.0_160.ckpt` downloaded from MobileNet V2 pre-trained models. Its weights were originally obtained by training on the ILSVRC-2012-CLS dataset for image classification ("Imagenet").

Usage

This model can be used with the `hub.KerasLayer` as follows. It *cannot* be used with the `hub.Module` API for TensorFlow 1.

```
m = tf.keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/imagenet/mobilenet_v2_100_160/classification/4")
])
m.build([None, 160, 160, 3])  # Batch input shape.
```

Running inference on the module

```
MODULE_HANDLE = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4'
module = hub.load(MODULE_HANDLE)

images = ... # Batches of images
predictions = tf.nn.softmax(module(images))
```

Label	Probability
Labrador retriever	0.58
kuvasz	0.08
Great Pyrenees	0.07

Using a module with Keras

```
MODULE_HANDLE = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4'
OUTPUT_SIZE = 1001
IMAGE_SIZE = (224, 224)

hub.KerasLayer(MODULE_HANDLE,
               output_shape=[OUTPUT_SIZE],
               input_shape=IMAGE_SIZE + (3,))
```

Using a module with Keras

```
MODULE_HANDLE = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/classification/4'
OUTPUT_SIZE = 1001
IMAGE_SIZE = (224, 224)

model = tf.keras.Sequential([
    hub.KerasLayer(MODULE_HANDLE,
                   output_shape=[OUTPUT_SIZE], input_shape=IMAGE_SIZE + (3,)),
    tf.keras.layers.Activation('softmax')
])

images = ... # Batches of images
predictions = model.predict(images)
```

Using a Feature Vector

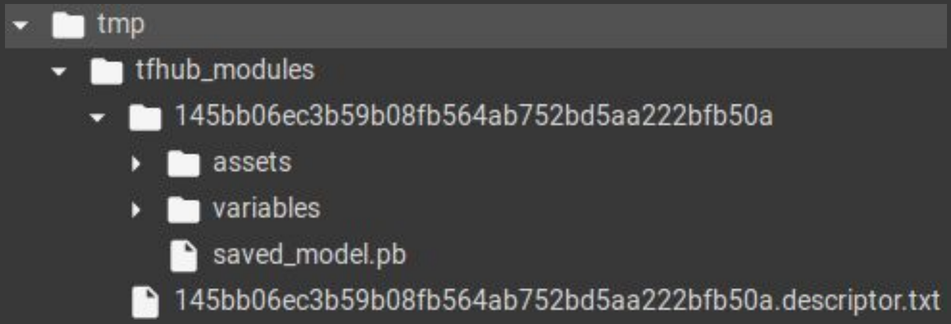
```
MODULE_HANDLE = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4"
FV_SIZE = 1280
IMAGE_SIZE = (224, 224)

model = tf.keras.Sequential([
    hub.KerasLayer(MODULE_HANDLE,
                   output_shape=[FV_SIZE],
                   input_shape=IMAGE_SIZE + (3,)),
    tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')
])

images = ... # Batches of images
predictions = model.predict(images)
```

Where do the modules get stored?

```
module = hub.load(...)
```



Saving a module for local use

```
MODULE_HANDLE = 'https://tfhub.dev/...?tf-hub-format=compressed'
!wget $MODULE_HANDLE

# Untar the tarball and load it with hub
hub_module = hub.load('/path/to/saved_model')
```

Inspecting the Hub module

```
SAVED_MODEL_DIR = ...
model = tf.saved_model.load(SAVED_MODEL_DIR, tags='serve')

images = ... # Batch of images
predictions = model(images)

>>> print(predictions)
Tensor("StatefulPartitionedCall_2:0", shape=(1, 1001), dtype=float32)
```

Relocating TF Hub modules

```python
import os
os.environ['TFHUB_CACHE_DIR'] = '/home/hub_cache_dir'
```

```bash
export TFHUB_CACHE_DIR='/home/hub_cache_dir'
```

Relocating TF Hub modules

```
module = hub.load(...)
```

```
home
  hub_cache_dir
    145bb06ec3b59b08fb564ab752bd5aa222bfb50a
      assets
      variables
      saved_model.pb
    145bb06ec3b59b08fb564ab752bd5aa222bfb50a.descriptor.txt
```

IMDB Reviews

```
train_validation_split = tfds.Split.TRAIN.subsplit([6, 4])

(train_data, validation_data), test_data = tfds.load(
    name="imdb_reviews",
    split=(train_validation_split, tfds.Split.TEST),
    as_supervised=True)
```

Explore the dataset

```
train_examples_batch, train_labels_batch = next(iter(train_data.batch(10)))
>>> train_examples_batch.numpy()

array([b"As a lifelong fan of Dickens, I have invariably been disappointed by ...",
       b"I absolutely LOVED this movie when I was a kid. I cried every time I ...",
       ...
       dtype=object)
```

Building a model for text classification

Architectural decisions

- How to **represent** the text?
- How many **layers** to use in the model?
- How many **hidden units** to use for each **layer**?

Embeddings to choose from

- google/tf2-preview/**gnews-swivel-20dim**(-with-oov)/1
- google/tf2-preview/**nnlm-en-dim50**/1
- google/tf2-preview/**nnlm-en-dim128**/a1

Experimenting with embeddings

```
embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"

hub_layer = hub.KerasLayer(embedding, input_shape=[], dtype=tf.string, trainable=True)


>>> hub_layer(train_examples_batch[:3])
<tf.Tensor: id=910, shape=(3, 20), dtype=float32, numpy=
array([[ 3.9819887 , -4.4838037 ,  5.177359  , -2.3643482 , -3.2938678 ,
        -3.5364532 , -2.4786978 ,  2.5525482 ,  6.688532  , -2.3076782 ,
        -1.9807833 ,  1.1315885 , -3.0339816 , -0.7604128 , -5.743445  ,
         3.4242578 ,  4.790099  , -4.03061   , -5.992149  , -1.7297493 ],
         ...
      dtype=float32)>
```

Experimenting with embeddings

```
embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"

hub_layer = hub.KerasLayer(embedding, input_shape=[], dtype=tf.string, trainable=True)


>>> hub_layer(train_examples_batch[:3])
<tf.Tensor: id=910, shape=(3, 20), dtype=float32, numpy=
array([[ 3.9819887 , -4.4838037 ,  5.177359  , -2.3643482 , -3.2938678 ,
        -3.5364532 , -2.4786978 ,  2.5525482 ,  6.688532  , -2.3076782 ,
        -1.9807833 ,  1.1315885 , -3.0339816 , -0.7604128 , -5.743445  ,
         3.4242578 ,  4.790099  , -4.03061   , -5.992149  , -1.7297493 ],
         ...
      dtype=float32)>
```

Experimenting with embeddings

```
embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"

hub_layer = hub.KerasLayer(embedding, input_shape=[], dtype=tf.string, trainable=True)
```

```
>>> hub_layer(train_examples_batch[:3])
<tf.Tensor: id=910, shape=(3, 20), dtype=float32, numpy=
array([[ 3.9819887 , -4.4838037 ,  5.177359  , -2.3643482 , -3.2938678 ,
        -3.5364532 , -2.4786978 ,  2.5525482 ,  6.688532  , -2.3076782 ,
        -1.9807833 ,  1.1315885 , -3.0339816 , -0.7604128 , -5.743445  ,
         3.4242578 ,  4.790099  , -4.03061   , -5.992149  , -1.7297493 ],
         ...
      dtype=float32)>
```

Experimenting with embeddings

```
embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"

hub_layer = hub.KerasLayer(embedding, input_shape=[], dtype=tf.string, trainable=True)
```

```
>>> hub_layer(train_examples_batch[:3])
<tf.Tensor: id=910, shape=(3, 20), dtype=float32, numpy=
array([[ 3.9819887 , -4.4838037 ,  5.177359  , -2.3643482 , -3.2938678 ,
        -3.5364532 , -2.4786978 ,  2.5525482 ,  6.688532  , -2.3076782 ,
        -1.9807833 ,  1.1315885 , -3.0339816 , -0.7604128 , -5.743445  ,
         3.4242578 ,  4.790099  , -4.03061   , -5.992149  , -1.7297493 ],
         ...
      dtype=float32)>
```

Create the model

```
model = tf.keras.Sequential([
                hub_layer,
                tf.keras.layers.Dense(16, activation='relu'),
                tf.keras.layers.Dense(1, activation='sigmoid')])
```

Train the model

```
# Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Start training
model.fit(train_data.shuffle(10000).batch(512),
          epochs=20,
          validation_data=validation_data.batch(512),
          verbose=1)
```

Time to evaluate our model

```
results = model.evaluate(test_data.batch(512), verbose=2)

>>> for name, value in zip(model.metrics_names, results):
      print("%s: %.3f" % (name, value))
loss: 0.318
accuracy: 0.865
```

Classify Cats and Dogs

```
splits = tfds.Split.ALL.subsplit(weighted=(80, 10, 10))

splits, info = tfds.load('cats_vs_dogs', 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
```

Prepare datasets

```
def format_image(image, label):
  # Resizes and normalizes the image
  image = tf.image.resize(image, IMAGE_SIZE) / 255.0
  return image, label

# Crate train and validation datasets
train_batches = train_examples.shuffle(SHUFFLE_SIZE)
                              .map(format_image)
                              .batch(BATCH_SIZE)
                              .prefetch(tf.data.experimental.AUTOTUNE)
validation_batches = validation_examples.map(format_image)
                                        .batch(BATCH_SIZE)
                                        .prefetch(tf.data.experimental.AUTOTUNE)
```

Select feature vector module

```
# Pick a feature vector module. e.g., InceptionV3, MobileNetV2, ...
handle_base = "mobilenet_v2"
MODULE_HANDLE = "https://tfhub.dev/google/tf2-preview/{}/feature_vector/4"
                .format(handle_base)

# This is size of the feature vector
FV_SIZE = 1280 # For MobileNetV2

IMAGE_SIZE = (224, 224)
```

https://tfhub.dev/google/imagenet/mobilenet_v2_050_96/feature_vector/4

Feature vector

Publisher

Google

Architecture

V2

Data set

Ilsvrc 2012 cls

Fine tunable: No

Last updated: 10/17/2019

V4

TF2 SavedModel

This is a SavedModel in TensorFlow 2 format. Using it requires TensorFlow 2 (or 1.15) and TensorFlow Hub 0.5.0 or newer.

Overview

MobileNet V2 is a family of neural network architectures for efficient on-device image classification and related tasks, originally published by

- Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen: "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation", 2018.

Mobilenets come in various sizes controlled by a multiplier for the depth (number of features) in the convolutional layers. They can also be trained for various sizes of input images to control inference speed.

This TF Hub model uses the TF-Slim implementation of `mobilenet_v2` with a depth multiplier of 0.5 and an input size of 96x96 pixels. This implementation of Mobilenet V2 rounds feature depths to multiples of 8 (an optimization *not* described in the paper). Depth multipliers less than 1.0 are not applied to the last convolutional layer (from which the module takes the image feature vector).

The model contains a trained instance of the network, packaged to get feature vectors from images. If you want the full model including the classification it was originally trained for, use `google/imagenet/mobilenet_v2_050_96/classification/4` instead.

Training

The checkpoint exported into this model was `mobilenet_v2_0.5_96/mobilenet_v2_0.5_96.ckpt` downloaded from MobileNet V2 pre-trained models. Its weights were originally obtained by training on the ILSVRC-2012-CLS dataset for image classification ("Imagenet").

Usage

This model can be used with the `hub.KerasLayer` as follows. It *cannot* be used with the `hub.Module` API for TensorFlow 1.

```
m = tf.keras.Sequential([
    hub.KerasLayer("https://tfhub.dev/google/imagenet/mobilenet_v2_050_96/feature_vector/4",
                   trainable=False),  # Can be True, see below.
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
m.build([None, 96, 96, 3])  # Batch input shape.
```

Transfer Learning with TF Hub

```
MODULE_HANDLE = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'

feature_extractor = hub.KerasLayer(MODULE_HANDLE,
                                   input_shape=IMAGE_SIZE + (3,),
                                   output_shape=[FV_SIZE],
                                   trainable=True)


>>> len(feature_extractor.variables)
260
>>> len(feature_extractor.trainable_variables)
156
>>> len(feature_extractor.non_trainable_variables)
104
```

Build model with TF Hub module

```
# Fine-tuning switch
do_fine_tuning = True

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

# Build model with the chosen module handle
model = tf.keras.Sequential([
    feature_extractor,
    tf.keras.layers.Dense(2, activation='softmax') # (cat, dog)
])
```

Model summary

```
>>> model.summary()

Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dense (Dense)                (None, 2)                 2562
=================================================================
Total params: 2,260,546
Trainable params: 2,226,434
Non-trainable params: 34,112
_________________________________________________________________
```

Choose the right training configuration

```python
# Define loss and metrics
loss = 'sparse_categorical_crossentropy'
metrics = ['accuracy']

# Choose an appropriate optimizer when fine-tuning
if do_fine_tuning:
  optimizer = tf.keras.optimizers.SGD(lr=0.002, momentum=0.9)
else:
  optimizer = 'adam'
```

Train the model

```
# Compile the model and fit
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.fit(train_batches, epochs=5,
          validation_data=validation_batches)
```