TensorFlow_Tutorials / TensorFlow_2.x / Chapter5_TransferLearning.ipynb

hy-23 Created using Colaboratory History

1 contributor

1.62 MB ···

Open in Colab

In [1]:

```
import os
import time
import tensorflow as tf
import numpy as np
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt
```

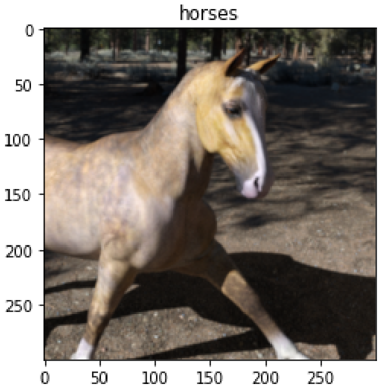
In [2]:

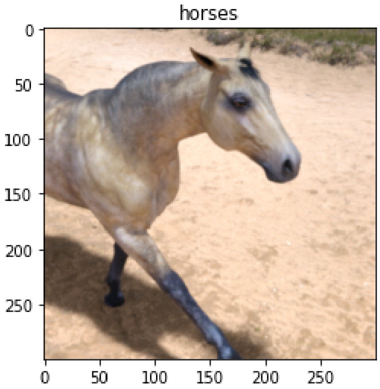
```
(train_ds, val_ds, test_ds), metadata = tfds.load('horses_or_humans',
                                                  split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
                                                  with_info=True,
                                                  as_supervised=True)
```

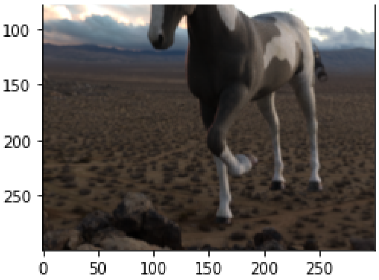
```
Downloading and preparing dataset horses_or_humans/3.0.0 (download: 153.59 MiB, generated: Unknown size,
 total: 153.59 MiB) to /root/tensorflow_datasets/horses_or_humans/3.0.0...
Dl Completed...: 0 url [00:00, ? url/s]
Dl Size...: 0 MiB [00:00, ? MiB/s]

0 examples [00:00, ? examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/horses_or_humans/3.0.0.incomplete07JYZI/horse
s_or_humans-train.tfrecord
  0%|          | 0/1027 [00:00<?, ? examples/s]
0 examples [00:00, ? examples/s]
Shuffling and writing examples to /root/tensorflow_datasets/horses_or_humans/3.0.0.incomplete07JYZI/horse
s_or_humans-test.tfrecord
  0%|          | 0/256 [00:00<?, ? examples/s]
Dataset horses_or_humans downloaded and prepared to /root/tensorflow_datasets/horses_or_humans/3.0.0. Sub
sequent calls will reuse this data.
```

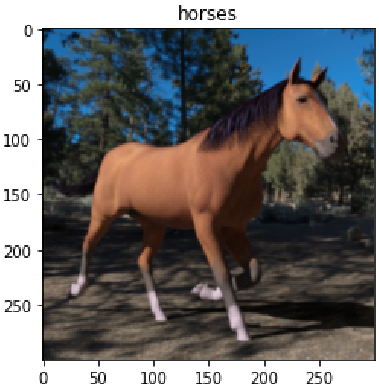
In [3]:

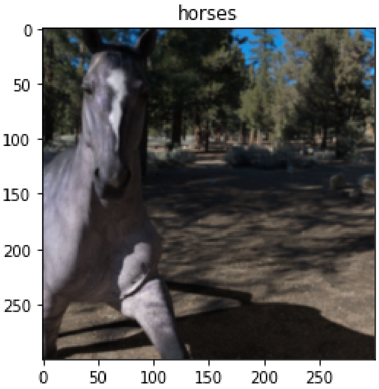
```
# inspect some images.

get_label_name = metadata.features['label'].int2str

def show_images(dataset):
  for image, label in dataset.take(10):
    plt.figure()
    plt.imshow(image)
    plt.title(get_label_name(label))

show_images(train_ds)
```

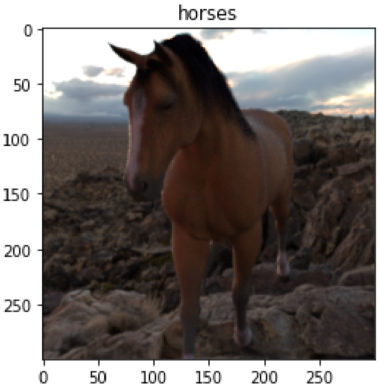


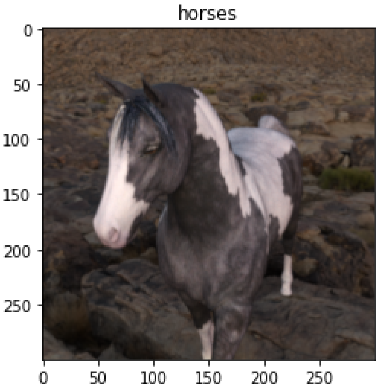



100
150
200
250
0
50
100
150
200
250

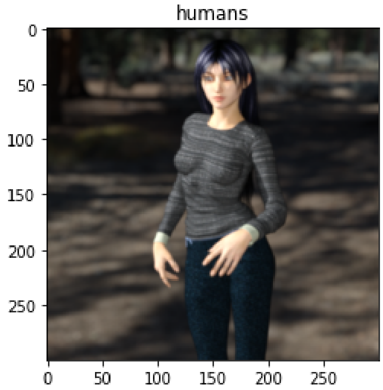

horses
0
50
100
150
200
250
0
50
100
150
200
250

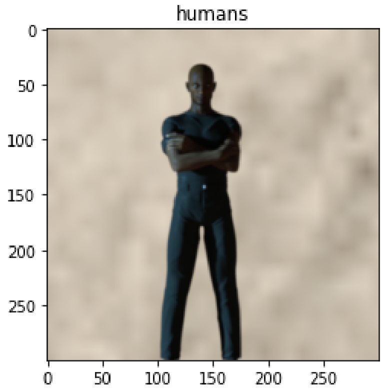

horses
0
50
100
150
200
250
0
50
100
150
200
250


horses
0
50
100
150
200
250
0
50
100
150
200
250


horses
0
50
100
150
200
250
0
50
100
150
200
250

humans
0
50
100





In [4]:

```
# format the images as per the requirement for MobileNetV2

IMG_SIZE = 160

def format_example(image, label):
  print("shape of incoming image is {}".format(image.shape))
  image = tf.cast(image, tf.float32)
  image = (image / 127.5) - 1
  image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
  return image, label

# This transformation applies map_func to each element of this dataset, and
# returns a new dataset containing the transformed elements, in the same
# order as they appeared in the input.
# map_func can be used to change both the values and the structure of a
# dataset's elements. Supported structure constructs are documented

train = train_ds.map(format_example)
val   = val_ds.map(format_example)
test  = test_ds.map(format_example)
```

```
shape of incoming image is (300, 300, 3)
shape of incoming image is (300, 300, 3)
shape of incoming image is (300, 300, 3)
```

In [5]:

```
BATCH_SIZE = 32
BUFFER_SIZE = 2000

AUTOTUNE = tf.data.AUTOTUNE

def configure_for_performance(ds):
  ds = ds.cache()
  ds = ds.shuffle(buffer_size=BUFFER_SIZE)
  ds = ds.batch(BATCH_SIZE)
  ds = ds.prefetch(buffer_size=AUTOTUNE)
  return ds
```

In [6]:

```
train_batch = configure_for_performance(train)
val_batch   = configure_for_performance(val)
```

```
test_batch  = test.batch(BATCH_SIZE)
```

In [7]:

```
IMG_SHAPE = (IMG_SIZE, IMG_SIZE, 3)

base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet
_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h5
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

In [8]:

```
# base_model.summary()

# Total params: 2,257,984
# Trainable params: 2,223,872
# Non-trainable params: 34,112
```

In [9]:

```
# base_model.summary()

# Total params: 2,257,984
# Trainable params: 0
# Non-trainable params: 2,257,984

base_model.trainable = False
```

In [10]:

```
for image_batch, label_batch in train_batch.take(1):
  pass

# each batch has set of 32 images in them.
print (image_batch.shape)
```

```
(32, 160, 160, 3)
```

In [11]:

```
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

```
(32, 5, 5, 1280)
```

In [12]:

```
# global average pooling. (32,5,5,1280) -> (32,1,1,1280) -> (32,1280)
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()

# feature_batch_average = tf.keras.layers.GlobalAveragePooling2D()(feature_batch)
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

In [39]:

```
prediction_layer = tf.keras.layers.Dense(1, activation='sigmoid')
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

In [40]:

```
# let us combine all of them.
model = tf.keras.models.Sequential([base_model,
                                   global_average_layer,
                                   prediction_layer])
```

In [41]:

```
# compile the model
lr = 0.0001
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate = lr),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

In [42]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense_1 (Dense)             (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,281
Non-trainable params: 2,257,984
_________________________________________________________________
```

In [43]:

```python
# evaluate the current model

loss, accuracy = model.evaluate(val_batch, steps=4)
```

```
4/4 [==============================] - 3s 307ms/step - loss: 1.3381 - accuracy: 0.2843
```

In [44]:

```python
total_no_images = metadata.splits['train'].num_examples

# The set contains 500 rendered images of various species of horse in various
# poses in various locations. It also contains 527 rendered images of humans in
# various poses and locations.

print("Total number of images {}".format(total_no_images))

num_train = total_no_images * 0.8
num_val   = total_no_images * 0.1
num_test  = total_no_images * 0.1
print("num_train is {}".format(num_train))
print("num_val is {}".format(num_val))
print("num_test is {}".format(num_test))
```

```
Total number of images 1027
num_train is 821.6
num_val is 102.7
num_test is 102.7
```

In [45]:

```python
# training settings.
initial_epoch = 10

# // : Floor division - division that results into whole number adjusted to the
# left in the number line
steps_per_epoch = round(num_train)//BATCH_SIZE
print("steps_per_epoch is {}".format(steps_per_epoch))
print("round(num_train) is {}".format(round(num_train)))
print("BATCH_SIZE is {}".format(BATCH_SIZE))
```

```
steps_per_epoch is 25
round(num_train) is 822
BATCH_SIZE is 32
```

In [46]:

```python
history = model.fit(train_batch, batch_size=BATCH_SIZE, epochs=initial_epoch,
                    validation_data=val_batch)
```

```
Epoch 1/10
26/26 [==============================] - 15s 458ms/step - loss: 0.9338 - accuracy: 0.4221 - val_loss: 0.9
255 - val_accuracy: 0.3431
Epoch 2/10
26/26 [==============================] - 12s 458ms/step - loss: 0.6966 - accuracy: 0.5608 - val_loss: 0.6
979 - val_accuracy: 0.5588
Epoch 3/10
26/26 [==============================] - 12s 453ms/step - loss: 0.5239 - accuracy: 0.7421 - val_loss: 0.5
199 - val_accuracy: 0.7549
Epoch 4/10
26/26 [==============================] - 12s 454ms/step - loss: 0.3909 - accuracy: 0.8869 - val_loss: 0.3
885 - val_accuracy: 0.9020
Epoch 5/10
26/26 [==============================] - 12s 458ms/step - loss: 0.2887 - accuracy: 0.9696 - val_loss: 0.2
894 - val_accuracy: 0.9510
Epoch 6/10
26/26 [==============================] - 12s 459ms/step - loss: 0.2123 - accuracy: 0.9866 - val_loss: 0.2
181 - val_accuracy: 0.9804
Epoch 7/10
26/26 [==============================] - 12s 461ms/step - loss: 0.1576 - accuracy: 0.9939 - val_loss: 0.1
656 - val_accuracy: 0.9902
Epoch 8/10
26/26 [==============================] - 12s 456ms/step - loss: 0.1171 - accuracy: 0.9951 - val_loss: 0.1
272 - val_accuracy: 0.9902
Epoch 9/10
26/26 [==============================] - 12s 461ms/step - loss: 0.0873 - accuracy: 0.9976 - val_loss: 0.0
886 - val_accuracy: 0.9902
```

```
996 - val_accuracy: 0.9902
Epoch 10/10
26/26 [==============================] - 12s 452ms/step - loss: 0.0655 - accuracy: 0.9976 - val_loss: 0.0
776 - val_accuracy: 0.9902
```

In [47]:

```
import cv2
```

In [54]:

```
im1 = cv2.imread('/content/drive/MyDrive/ResourceFiles/Shreyas.jpg')
im2 = cv2.imread('/content/drive/MyDrive/ResourceFiles/horse.jpg')
im3 = cv2.imread('/content/drive/MyDrive/ResourceFiles/Amruthesha.jpg')
im4 = cv2.imread('/content/drive/MyDrive/ResourceFiles/Dog.jpg')
im5 = cv2.imread('/content/drive/MyDrive/ResourceFiles/horse_2.jpg')
```

In [55]:

```
img1 = cv2.resize(im1, (160, 160)).astype(np.float32)
img1 = np.expand_dims(img1, axis=0)

img2 = cv2.resize(im2, (160, 160)).astype(np.float32)
img2 = np.expand_dims(img2, axis=0)

img3 = cv2.resize(im3, (160, 160)).astype(np.float32)
img3 = np.expand_dims(img3, axis=0)

img4 = cv2.resize(im4, (160, 160)).astype(np.float32)
img4 = np.expand_dims(img4, axis=0)

img5 = cv2.resize(im5, (160, 160)).astype(np.float32)
img5 = np.expand_dims(img5, axis=0)

im = np.concatenate((img1, img2, img3, img4, img5), axis=0)
```

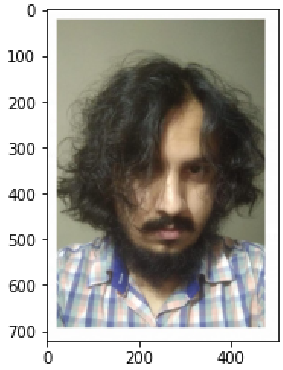
In [56]:

```
img, label = format_example(im, '')
out = model.predict(img)
```

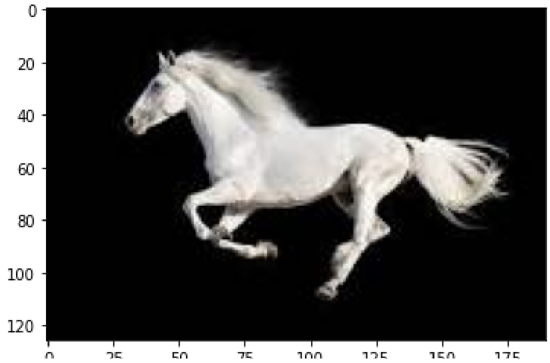
```
shape of incoming image is (5, 160, 160, 3)
```

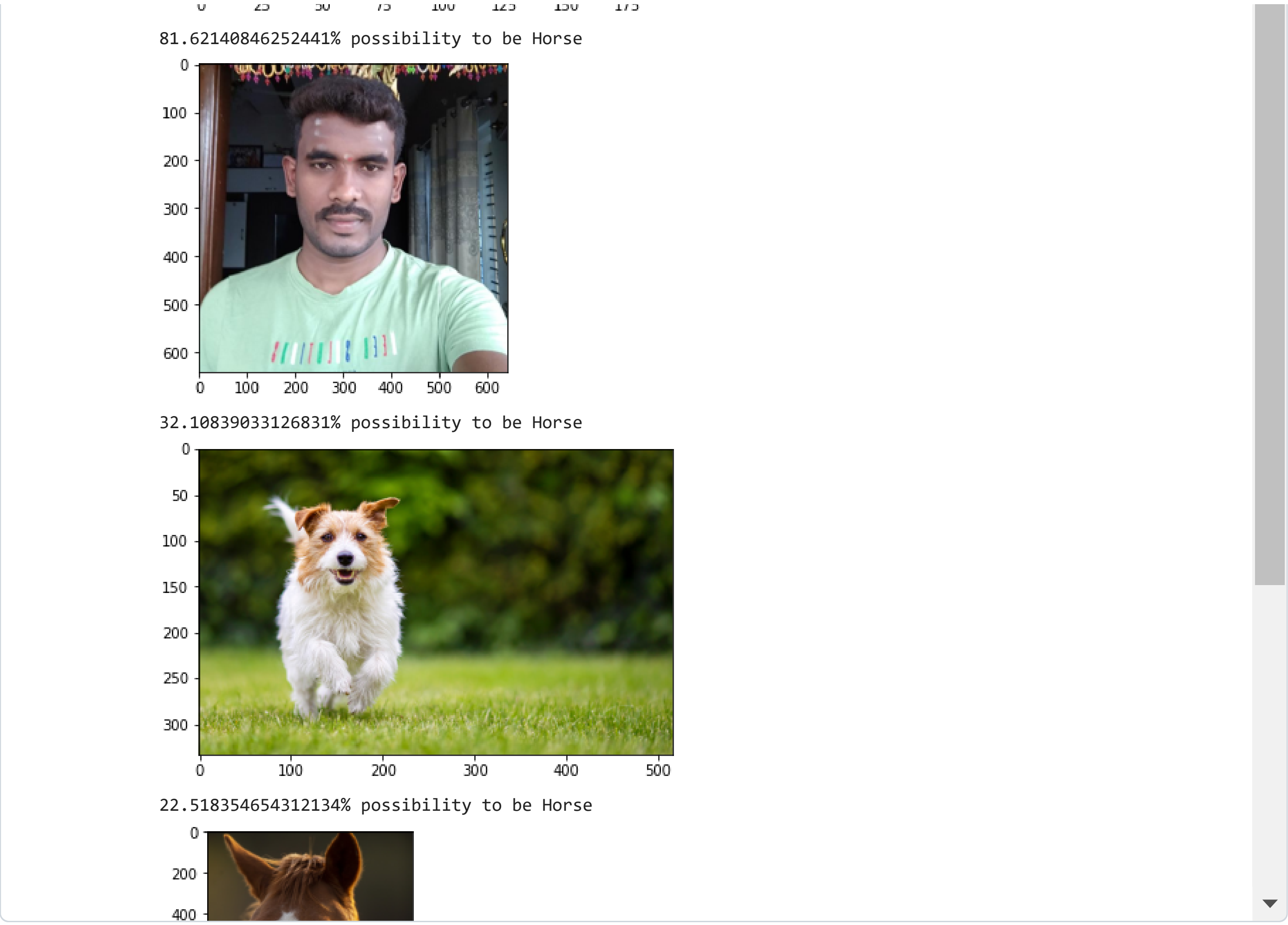
In [57]:

```
def h_plot(prediction, *args):
  for count, im in enumerate(args):
    plt.imshow(cv2.cvtColor(im, cv2.COLOR_BGR2RGB))
    plt.show()

    # category = 'horse' if (prediction[count] < 0.5) else 'not horse'
    confidence = (1-prediction[count][0])

    print("{}% possibility to be Horse".format(confidence*100))


h_plot(out, im1, im2, im3, im4, im5)
```



```
24.644547700881958% possibility to be Horse
```





```
81.62140846252441% possibility to be Horse
```

```
32.10839033126831% possibility to be Horse
```

```
22.518354654312134% possibility to be Horse
```