Copyright 2018 The TensorFlow Authors.

```
import numpy as np
import time

import PIL.Image as Image
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

import datetime

%load_ext tensorboard
```

```
mobilenet_v2 ="https://tfhub.dev/google/tf2-preview/mobilenet_v2/cl
inception_v3 = "https://tfhub.dev/google/imagenet/inception_v3/classification/5"

classifier_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_v3"] {type:"raw"}
```

classifier_model: mobilenet_v2

```
IMAGE_SHAPE = (224, 224)

classifier = tf.keras.Sequential([
    hub.KerasLayer(classifier_model, input_shape=IMAGE_SHAPE+(3,))
])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-7db51c8d2d71> in <module>
      1 IMAGE_SHAPE = (224, 224)
      2
----> 3 classifier = tf.keras.Sequential([
      4     hub.KerasLayer(classifier_model, input_shape=IMAGE_SHAPE+(3,))
      5 ])

NameError: name 'tf' is not defined
```

SEARCH STACK OVERFLOW

```
labels_path = tf.keras.utils.get_file('ImageNetLabels.txt','https://storage.googleapis.com
imagenet_labels = np.array(open(labels_path).read().splitlines())
```

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

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
!unzip gdrive/My\ Drive/heart_recorrtadas.zip
```

```
Archive:  gdrive/My Drive/heart_recorrtadas.zip
   creating: heart/
   creating: heart/anormal/
  inflating: heart/anormal/HB(1).jpg
  inflating: heart/anormal/HB(10).jpg
  inflating: heart/anormal/HB(100).jpg
  inflating: heart/anormal/HB(101).jpg
  inflating: heart/anormal/HB(102).jpg
  inflating: heart/anormal/HB(103).jpg
  inflating: heart/anormal/HB(104).jpg
  inflating: heart/anormal/HB(105).jpg
  inflating: heart/anormal/HB(106).jpg
  inflating: heart/anormal/HB(107).jpg
  inflating: heart/anormal/HB(108).jpg
  inflating: heart/anormal/HB(109).jpg
  inflating: heart/anormal/HB(11).jpg
  inflating: heart/anormal/HB(110).jpg
  inflating: heart/anormal/HB(111).jpg
  inflating: heart/anormal/HB(112).jpg
  inflating: heart/anormal/HB(113).jpg
  inflating: heart/anormal/HB(114).jpg
  inflating: heart/anormal/HB(115).jpg
  inflating: heart/anormal/HB(116).jpg
  inflating: heart/anormal/HB(117).jpg
  inflating: heart/anormal/HB(118).jpg
  inflating: heart/anormal/HB(119).jpg
  inflating: heart/anormal/HB(12).jpg
  inflating: heart/anormal/HB(120).jpg
  inflating: heart/anormal/HB(121).jpg
  inflating: heart/anormal/HB(122).jpg
  inflating: heart/anormal/HB(123).jpg
  inflating: heart/anormal/HB(124).jpg
  inflating: heart/anormal/HB(125).jpg
  inflating: heart/anormal/HB(126).jpg
  inflating: heart/anormal/HB(127).jpg
  inflating: heart/anormal/HB(128).jpg
  inflating: heart/anormal/HB(129).jpg
  inflating: heart/anormal/HB(13).jpg
  inflating: heart/anormal/HB(130).jpg
  inflating: heart/anormal/HB(131).jpg
  inflating: heart/anormal/HB(132).jpg
  inflating: heart/anormal/HB(133).jpg
  inflating: heart/anormal/HB(134).jpg
  inflating: heart/anormal/HB(135).jpg
  inflating: heart/anormal/HB(136).jpg
  inflating: heart/anormal/HB(137).jpg
  inflating: heart/anormal/HB(138).jpg
  inflating: heart/anormal/HB(139).jpg
  inflating: heart/anormal/HB(14).jpg
  inflating: heart/anormal/HB(140).jpg
  inflating: heart/anormal/HB(141).jpg
  inflating: heart/anormal/HB(142).jpg
  inflating: heart/anormal/HB(143).jpg
  inflating: heart/anormal/HB(144).jpg
  inflating: heart/anormal/HB(145).jpg
```

```
        inflating: heart/anormal/HB(146).jpg
        inflating: heart/anormal/HB(147).jpg
        inflating: heart/anormal/HB(148).jpg
```

```
batch_size = 170
img_height = 224
img_width = 224
data_root= '/content/heart';
train_ds = tf.keras.utils.image_dataset_from_directory(
  str(data_root),
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)

val_ds = tf.keras.utils.image_dataset_from_directory(
  str(data_root),
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size
)
```

```
    Found 928 files belonging to 4 classes.
    Using 743 files for training.
    Found 928 files belonging to 4 classes.
    Using 185 files for validation.
```

```
#val_ds.classes


class_names = np.array(train_ds.class_names)
print(class_names)
```

```
    ['anormal' 'history' 'miocardio' 'normal']
```

```
normalization_layer = tf.keras.layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x—images, y—labe
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y)) # Where x—images, y—labels.


AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)


print(val_ds)
```

```
    <PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32
```

```
for image_batch, labels_batch in train_ds:
```

```
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(170, 224, 224, 3)
(170,)
```

```
for image_batch_validation, labels_batch_validation in val_ds:
  print(image_batch_validation.shape)
  print(image_batch_validation.shape)
  break
```

```
(170, 224, 224, 3)
(170, 224, 224, 3)
```

```
##print(x_train,y_train )
##print(x_val,y_val )
```

```
#y_train = tf.keras.utils.to_categorical(743)
#y_val = tf.keras.utils.to_categorical(185)
```

```
#y_train = np.expand_dims(x_val,2)
#,y_val = np.expand_dims(y_val,2)
```

Run the classifier on a batch of images

Now, run the classifier on an image batch:

```
#result_batch = classifier.predict(train_ds)
```

```
##predicted_class_names = imagenet_labels[tf.math.argmax(result_batch, axis=-1)]
##predicted_class_names
```

Check how these predictions line up with the images:

```
##print(predicted_class_names)
```

```
##plt.figure(figsize=(10,9))
```

```
##plt.subplots_adjust(hspace=0.5)
##for n in range(30):
  ##plt.subplot(6,5,n+1)
  #plt.imshow(image_batch[n])
 # plt.title(predicted_class_names[n])
 # plt.axis('off')
# = plt.suptitle("ImageNet predictions")
```

Note: all images are licensed CC-BY, creators are listed in the LICENSE.txt file.

The results are far from perfect, but reasonable considering that these are not the classes the model was trained for (except for "daisy").

Download the headless model

TensorFlow Hub also distributes models without the top classification layer. These can be used to easily perform transfer learning.

Select a MobileNetV2 pre-trained model from TensorFlow Hub. Any compatible image feature vector model from TensorFlow Hub will work here, including the examples from the drop-down menu.

```
#mobilenet_v2 = "https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature
mobilenet_v2 = "https://tfhub.dev/google/bit/s-r50x3/ilsvrc2012_classification/1"

feature_extractor_model = mobilenet_v2 #@param ["mobilenet_v2", "inception_v3"] {
```

Create the feature extractor by wrapping the pre-trained model as a Keras layer with `hub.KerasLayer`. Use the `trainable=False` argument to freeze the variables, so that the training only modifies the new classifier layer:

```
feature_extractor_layer = hub.KerasLayer(
    feature_extractor_model,
    input_shape=(224, 224, 3),
    trainable=False)
```

texto del enlaceThe feature extractor returns a 1280-long vector for each image (the image batch size remains at 32 in this example):

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

```
(170, 1000)
```

Attach a classification head

To complete the model, wrap the feature extractor layer in a `tf.keras.Sequential` model and add a fully-connected layer for classification:

```
num_classes = len(class_names)

model = tf.keras.Sequential([
  feature_extractor_layer,
  tf.keras.layers.Dense(num_classes)
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 keras_layer_1 (KerasLayer)  (None, 1000)              217319080 

 dense_1 (Dense)             (None, 4)                 4004      

=================================================================
Total params: 217,323,084
Trainable params: 4,004
Non-trainable params: 217,319,080
_________________________________________________________________
```

```
predictions = model(image_batch)
```

```
predictions.shape
```

```
TensorShape([170, 4])
```

Train the model

Use `Model.compile` to configure the training process and add a `tf.keras.callbacks.TensorBoard` callback to create and store logs:

```
model.compile(
  optimizer=tf.keras.optimizers.Adam(),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['acc'])

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(
    log_dir=log_dir,
    histogram_freq=1) # Enable histogram computation for every epoch.
```

Now use the `Model.fit` method to train the model.

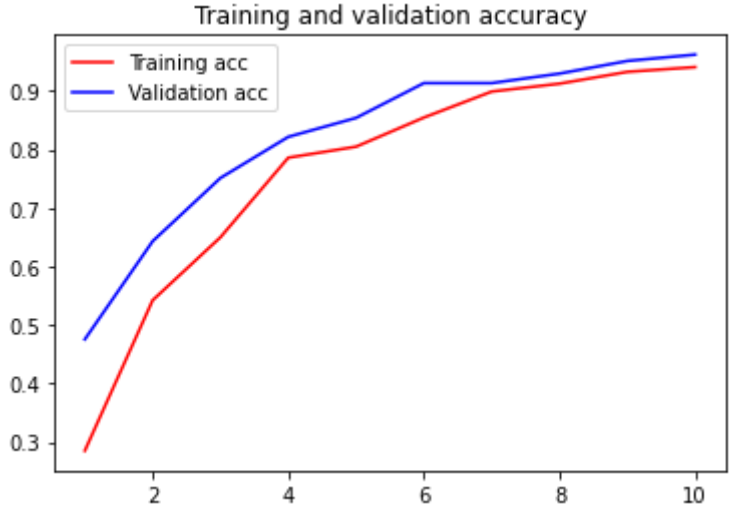
To keep this example short, you'll be training for just 10 epochs. To visualize the training progress in TensorBoard later, create and store logs an a [TensorBoard callback](#).

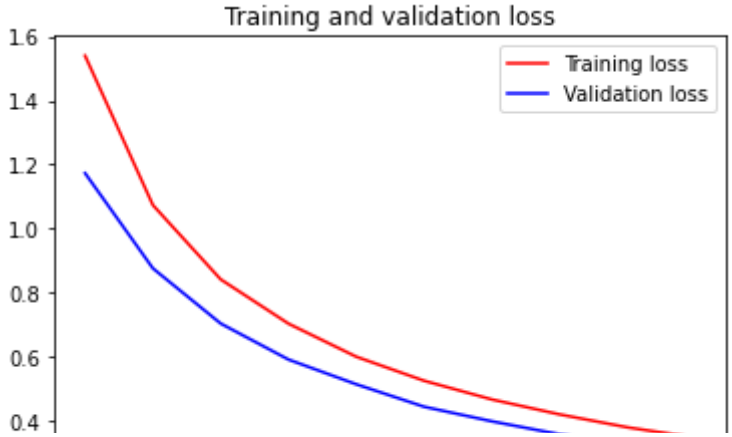
```
NUM_EPOCHS = 10

history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=NUM_EPOCHS,
                    callbacks=tensorboard_callback)
```

```
Epoch 1/10
5/5 [==============================] - 86s 17s/step - loss: 1.5393 - acc: 0.2853 - va
Epoch 2/10
5/5 [==============================] - 63s 15s/step - loss: 1.0739 - acc: 0.5424 - va
Epoch 3/10
5/5 [==============================] - 63s 14s/step - loss: 0.8425 - acc: 0.6501 - va
Epoch 4/10
5/5 [==============================] - 62s 14s/step - loss: 0.7045 - acc: 0.7860 - va
Epoch 5/10
5/5 [==============================] - 63s 15s/step - loss: 0.6014 - acc: 0.8048 - va
Epoch 6/10
5/5 [==============================] - 62s 14s/step - loss: 0.5266 - acc: 0.8546 - va
Epoch 7/10
5/5 [==============================] - 64s 15s/step - loss: 0.4682 - acc: 0.8991 - va
Epoch 8/10
5/5 [==============================] - 63s 14s/step - loss: 0.4216 - acc: 0.9125 - va
Epoch 9/10
5/5 [==============================] - 63s 15s/step - loss: 0.3813 - acc: 0.9327 - va
Epoch 10/10
5/5 [==============================] - 63s 15s/step - loss: 0.3494 - acc: 0.9408 - va
```

Start the TensorBoard to view how the metrics change with each epoch and to track other scalar values:

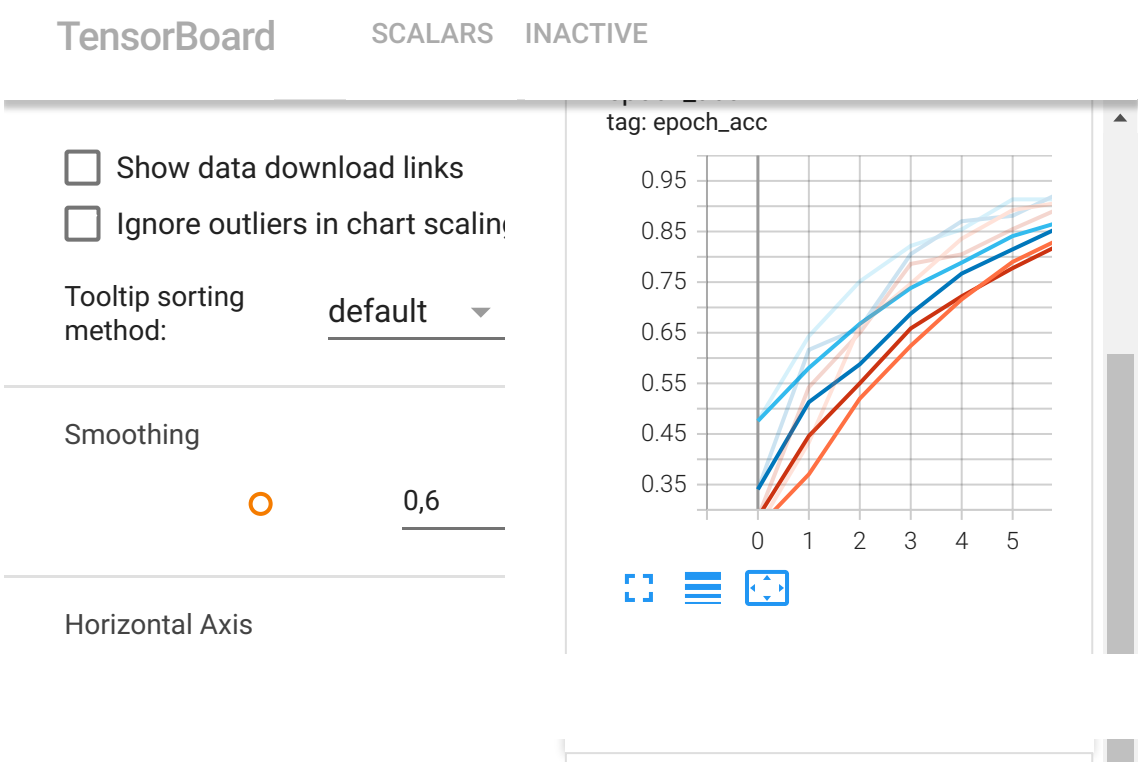
```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```






```
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 579), started 0:16:14 ago. (Use
'!kill 579' to kill it.)
```




```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import f1_score
```

Check the predictions

Obtain the ordered list of class names from the model predictions:

```
#predicted_id =  np.argmax(model.predict(image_batch1), axis=-1)
#print(predicted_id)


!unzip gdrive/My\ Drive/Testing.zip
```

```
Archive:  gdrive/My Drive/Testing.zip
   creating: Testing/
   creating: Testing/ANORMALES/
  inflating: Testing/ANORMALES/HB(221).jpg
  inflating: Testing/ANORMALES/HB(222).jpg
  inflating: Testing/ANORMALES/HB(223).jpg
  inflating: Testing/ANORMALES/HB(224).jpg
  inflating: Testing/ANORMALES/HB(225).jpg
  inflating: Testing/ANORMALES/HB(226).jpg
  inflating: Testing/ANORMALES/HB(227).jpg
```

```
        inflating: Testing/ANORMALES/HB(228).jpg
        inflating: Testing/ANORMALES/HB(229).jpg
        inflating: Testing/ANORMALES/HB(230).jpg
        inflating: Testing/ANORMALES/HB(231).jpg
        inflating: Testing/ANORMALES/HB(232).jpg
        inflating: Testing/ANORMALES/HB(233).jpg
         creating: Testing/HISTORIAL/
        inflating: Testing/HISTORIAL/PMI(160).jpg
        inflating: Testing/HISTORIAL/PMI(161).jpg
        inflating: Testing/HISTORIAL/PMI(162).jpg
        inflating: Testing/HISTORIAL/PMI(163).jpg
        inflating: Testing/HISTORIAL/PMI(164).jpg
        inflating: Testing/HISTORIAL/PMI(165).jpg
        inflating: Testing/HISTORIAL/PMI(166).jpg
        inflating: Testing/HISTORIAL/PMI(167).jpg
        inflating: Testing/HISTORIAL/PMI(168).jpg
        inflating: Testing/HISTORIAL/PMI(169).jpg
        inflating: Testing/HISTORIAL/PMI(170).jpg
        inflating: Testing/HISTORIAL/PMI(171).jpg
        inflating: Testing/HISTORIAL/PMI(172).jpg
         creating: Testing/MIOCARDIO/
        inflating: Testing/MIOCARDIO/MI(228).jpg
        inflating: Testing/MIOCARDIO/MI(229).jpg
        inflating: Testing/MIOCARDIO/MI(230).jpg
        inflating: Testing/MIOCARDIO/MI(231).jpg
        inflating: Testing/MIOCARDIO/MI(232).jpg
        inflating: Testing/MIOCARDIO/MI(233).jpg
        inflating: Testing/MIOCARDIO/MI(234).jpg
        inflating: Testing/MIOCARDIO/MI(235).jpg
        inflating: Testing/MIOCARDIO/MI(236).jpg
        inflating: Testing/MIOCARDIO/MI(237).jpg
        inflating: Testing/MIOCARDIO/MI(238).jpg
        inflating: Testing/MIOCARDIO/MI(239).jpg
        inflating: Testing/MIOCARDIO/MI(240).jpg
         creating: Testing/NORMAL/
        inflating: Testing/NORMAL/Normal(272).jpg
        inflating: Testing/NORMAL/Normal(273).jpg
        inflating: Testing/NORMAL/Normal(274).jpg
        inflating: Testing/NORMAL/Normal(275).jpg
        inflating: Testing/NORMAL/Normal(276).jpg
        inflating: Testing/NORMAL/Normal(277).jpg
        inflating: Testing/NORMAL/Normal(278).jpg
        inflating: Testing/NORMAL/Normal(279).jpg
        inflating: Testing/NORMAL/Normal(280).jpg
        inflating: Testing/NORMAL/Normal(281).jpg
        inflating: Testing/NORMAL/Normal(282).jpg
        inflating: Testing/NORMAL/Normal(283).jpg
```

```
data_root_= '/content/heart';


data_test = tf.keras.utils.image_dataset_from_directory(
  str(data_root_),
  image_size=(img_height, img_width)
)
```

```
    Found 928 files belonging to 4 classes.
```

```
normalization_layer = tf.keras.layers.Rescaling(1./255)
data_test = data_test.map(lambda x, y: (normalization_layer(x), y)) # Where x—images, y—la


AUTOTUNE = tf.data.AUTOTUNE
data_test = data_test.cache().prefetch(buffer_size=AUTOTUNE)


test_verd = tf.concat([y for x, y in data_test], axis=-1)
print(test_verd)
```

```
tf.Tensor(
[3 0 3 2 3 0 2 2 1 3 1 2 0 0 3 1 3 1 3 3 2 3 1 0 0 3 0 3 2 1 2 1 0 3 1 2 3
 3 2 1 2 0 3 1 2 1 0 0 0 0 3 3 1 2 0 3 3 0 1 2 2 3 0 1 2 3 1 3 0 2 3 3 3 3
 0 0 0 3 2 3 3 3 3 1 1 3 2 3 1 3 2 2 1 0 0 0 3 0 3 1 1 3 3 2 3 3 2 0 1 2 3
 0 3 2 2 3 1 2 0 3 3 1 0 0 0 3 2 0 1 2 2 2 3 0 3 0 2 3 0 0 3 2 3 3 0 0 1 3
 3 0 0 1 2 3 3 2 0 3 1 0 3 0 0 0 0 1 2 1 0 2 2 2 3 2 0 2 0 3 2 3 0 2 0 0 0
 0 2 2 3 0 2 2 3 2 3 1 3 2 2 0 2 1 3 0 0 3 1 1 2 0 3 0 0 0 0 3 2 0 3 3 2 3
 1 0 1 1 3 1 0 0 1 1 1 1 2 2 0 1 1 3 3 3 0 3 0 1 1 0 3 0 2 2 2 3 3 2 0 3 0
 0 2 3 3 3 2 1 2 3 3 0 2 3 0 2 1 0 1 3 0 0 0 2 3 1 2 0 1 2 3 3 3 2 2 0 2 2
 0 0 1 3 0 0 0 3 3 2 3 2 2 0 0 3 1 1 2 1 0 2 0 3 1 2 3 0 0 1 0 2 3 0 0 2 0
 3 3 1 3 1 3 2 1 1 3 3 3 2 3 2 3 0 2 3 3 0 0 1 3 2 3 0 2 2 1 3 1 2 0 0 3 2
 1 2 1 0 0 0 0 1 2 1 2 2 0 0 0 0 2 0 0 3 3 2 3 1 1 0 3 0 0 2 1 0 2 1 1 1 2
 3 0 2 1 3 3 0 3 1 3 3 0 3 3 1 0 2 3 2 2 1 3 0 3 3 0 0 1 1 3 2 1 0 0 0 2 0
 0 3 3 0 3 0 3 3 3 3 3 3 3 3 2 3 3 0 1 3 2 1 1 2 2 0 2 2 1 3 0 0 0 2 2 1 0
 1 2 2 3 3 2 3 0 3 1 3 3 0 3 1 2 1 3 0 3 0 0 2 2 2 3 2 2 1 2 3 2 1 2 0 3 3
 0 2 0 2 0 3 2 3 0 0 2 3 2 0 0 3 3 2 3 3 3 2 1 1 1 2 3 3 2 3 0 1 3 0 3 2 2
 2 3 2 3 3 0 3 1 3 3 0 1 3 3 0 0 3 1 1 2 1 3 3 2 2 1 1 3 3 2 0 1 0 0 3 0 3
 2 0 1 0 1 3 3 2 3 0 3 2 0 0 3 0 3 1 3 2 0 2 0 1 3 1 3 0 1 0 2 2 1 3 3 1 1
 2 2 2 1 3 0 3 1 1 0 3 3 0 1 0 3 0 0 1 1 3 0 0 3 1 0 2 2 3 3 2 0 3 1 0 2 3
 0 3 0 0 0 2 1 3 3 2 3 2 3 2 1 0 1 1 2 2 2 3 3 2 0 3 3 0 3 3 3 0 3 1 0 3 2
 3 3 1 2 1 1 1 3 3 1 1 2 2 0 2 2 2 1 1 0 1 0 3 3 2 2 2 3 0 3 0 3 1 2 3 1 3
 0 0 1 1 1 3 1 2 1 1 3 2 2 2 3 1 1 2 2 2 0 0 2 3 0 3 0 0 1 1 2 2 0 0 2 3 2
 2 3 2 3 3 3 3 2 1 0 0 0 2 0 2 3 2 2 2 3 1 1 0 2 2 0 3 2 1 0 3 2 3 3 2 3 2
 1 3 3 2 3 3 3 1 3 0 1 2 1 2 3 2 0 3 2 2 0 3 3 3 1 2 3 2 2 0 2 3 0 0 0 2 1
 2 2 3 1 2 2 1 2 0 3 3 2 1 1 0 0 3 3 3 3 2 3 2 2 1 2 2 1 2 2 0 0 3 0 1 2 2
 0 2 3 1 1 2 1 3 1 1 3 0 3 0 1 3 2 2 2 0 2 2 0 2 0 3 2 1 2 2 0 2 0 2 3 1 2
 3 3 0], shape=(928,), dtype=int32)
```

```
test_pred = np.argmax( model.predict(data_test), axis=-1)
print( test_pred )
print( len(test_pred) )
```

```
[3 0 3 2 3 0 2 2 1 3 1 2 0 0 3 1 3 1 3 3 2 3 1 0 0 3 0 3 2 1 2 0 0 3 1 2 3
 3 2 1 2 0 3 1 2 0 0 0 0 0 3 3 1 2 0 3 3 0 1 3 2 3 0 1 2 3 1 3 0 2 3 3 3 3
 0 0 0 3 2 3 3 3 3 1 1 3 2 3 0 3 2 2 1 0 0 0 3 0 3 1 1 3 3 2 3 3 2 0 1 2 3
 0 3 3 2 3 1 2 0 3 3 1 0 0 0 3 2 3 1 2 2 2 3 0 3 0 2 3 0 0 3 2 3 3 0 0 1 3
 3 3 0 1 2 3 3 2 0 3 1 0 3 0 0 0 0 1 2 1 0 2 2 2 3 2 0 2 0 3 2 3 0 2 0 2 0
 0 2 2 3 0 2 2 3 2 3 1 3 2 2 0 2 2 3 0 0 3 2 2 2 0 3 0 0 0 1 3 2 0 3 3 2 3
 1 0 1 1 3 1 0 0 1 1 1 1 2 2 0 1 1 3 3 3 0 3 0 1 1 0 3 0 2 2 2 3 3 2 0 3 0
 0 2 3 3 3 2 1 2 3 3 0 2 2 0 2 1 0 1 3 0 0 0 2 3 1 2 0 1 2 3 3 3 2 2 0 2 2
 0 0 1 3 0 0 0 3 3 3 3 2 2 0 0 3 1 1 2 1 0 2 0 3 1 2 3 0 0 1 0 2 3 0 0 2 0
 3 3 1 3 1 3 2 1 3 3 3 3 2 3 2 3 0 2 3 3 0 0 1 3 2 3 0 2 2 2 3 1 2 0 0 3 2
 1 2 1 0 0 0 0 1 2 1 2 2 0 0 0 0 2 0 0 3 3 2 3 1 1 0 3 0 0 2 1 0 2 1 1 2 2
 3 0 2 1 3 3 0 3 1 3 3 0 3 3 0 0 2 3 2 2 2 3 0 3 3 0 0 1 1 3 2 1 0 0 0 2 0
 1 3 3 0 3 0 3 3 3 3 3 3 3 3 2 3 3 0 1 3 2 1 1 2 2 0 2 2 1 3 0 0 0 2 2 1 0
 1 2 2 3 2 2 3 3 3 1 3 3 0 3 1 2 1 3 0 3 0 0 2 2 2 3 2 2 1 2 3 2 1 2 0 3 3
 0 2 0 2 0 3 2 3 0 0 2 3 2 0 0 3 3 2 3 3 3 2 1 1 1 2 3 3 2 3 0 1 3 0 3 2 2
```

```
 2 3 2 3 3 1 3 1 3 3 0 3 3 3 0 0 3 1 1 2 3 3 3 2 2 1 1 3 3 2 3 1 0 0 3 0 3
 2 0 1 0 1 3 3 3 3 0 3 2 0 0 3 0 3 1 3 2 0 2 0 1 3 2 3 0 1 0 3 2 1 3 3 1 1
 2 2 3 1 3 0 3 1 1 0 3 3 0 3 0 3 0 0 0 1 3 0 0 3 2 0 2 2 3 3 2 0 3 1 0 2 3
 0 3 0 0 0 2 0 3 3 2 3 2 3 2 1 0 1 1 2 2 2 3 3 2 0 3 3 0 3 3 3 0 3 1 0 3 2
 3 3 2 2 1 1 2 3 3 1 1 2 2 0 2 2 3 1 1 0 3 0 3 3 2 2 2 3 2 3 0 3 1 2 3 1 3
 0 0 1 1 1 3 1 2 1 1 3 2 2 2 3 1 1 2 2 2 0 0 2 3 0 3 0 0 2 1 2 2 0 0 2 3 2
 2 3 2 3 3 3 3 2 1 0 0 0 2 0 2 3 2 2 2 3 1 3 0 2 2 0 3 2 3 0 3 2 3 3 2 3 2
 1 3 3 2 3 3 3 1 3 0 3 2 2 2 3 2 0 3 2 2 0 3 3 3 1 2 3 2 2 0 2 3 0 0 0 2 1
 2 2 3 1 3 2 1 2 0 3 3 2 2 1 0 0 3 3 3 3 2 3 2 2 1 2 2 1 2 2 0 0 3 0 1 2 2
 0 2 3 3 1 2 2 3 1 1 3 0 3 0 3 3 2 2 2 0 2 2 0 2 0 3 2 1 2 2 0 2 0 2 3 1 2
 3 3 0]
928
```

```
print(class_names )
```

```
['anormal' 'history' 'miocardio' 'normal']
```

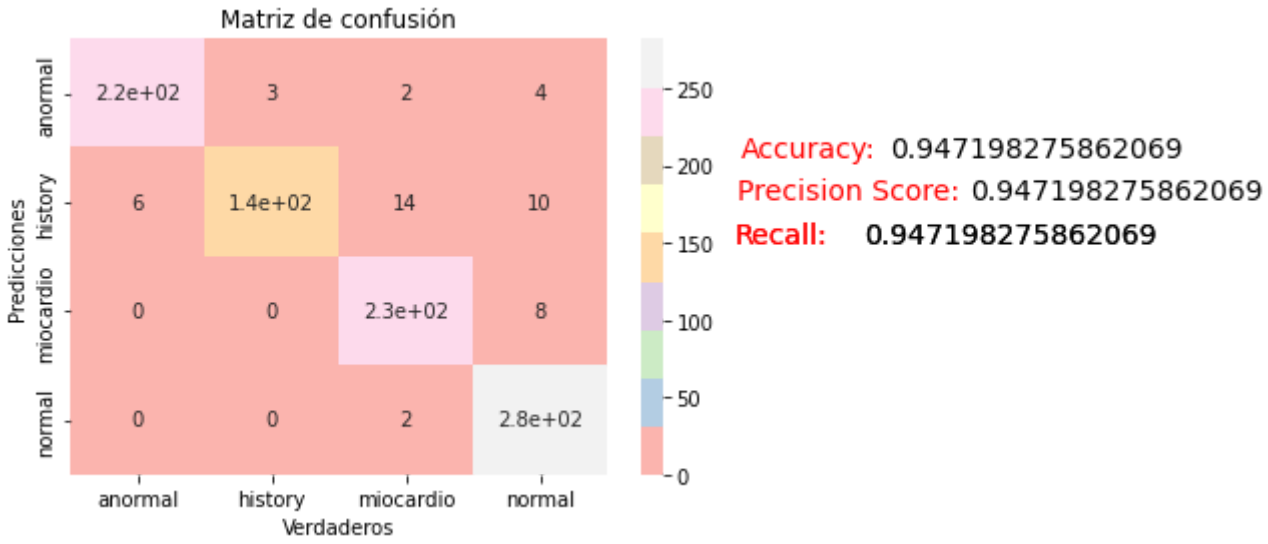
```
cm = confusion_matrix(test_verd,  test_pred )
print(cm)
```

```
[[224   3   2   4]
 [  6 142  14  10]
 [  0   0 231   8]
 [  0   0   2 282]]
```

```
from string import ascii_uppercase
import pandas as pd
import seaborn as sns
```

```
columnas=class_names
df_cm = pd.DataFrame( cm,index= columnas, columns = columnas )
grafica = sns.heatmap( df_cm, cmap= 'Pastel1',annot = True )

plt.title('Matriz de confusión')
plt.ylabel('Valores verdaderos')
plt.xlabel('Predicciones')
grafica.set(xlabel = 'Verdaderos',ylabel='Predicciones')
## Mostramos Accuracy
accuracy = accuracy_score( test_verd,  test_pred )
plt.text(5.5, 1, 'Accuracy:', fontsize=14, c="red", ha="center", va="center")
plt.text(7.2,1, accuracy , fontsize=14, ha="center", va="center")
## Mostramos Precision Score
precision = precision_score( test_verd,  test_pred,average='micro' )
plt.text(5.8, 1.4, 'Precision Score:', fontsize=14, c="red", ha="center", va="center")
plt.text(7.8,1.4, precision , fontsize=14, ha="center", va="center")
## Mostramos Recall
recall = recall_score( test_verd,  test_pred, average='micro' )
plt.text(5.3, 1.79, 'Recall:', fontsize=14, c="red", ha="center", va="center")
plt.text(7.0,1.79, recall , fontsize=14, ha="center", va="center")
## Mostramos Recall
f1 = f1_score( test_verd,  test_pred, average='micro' )
plt.text(5.3, 1.79, 'Recall:', fontsize=14, c="red", ha="center", va="center")
plt.text(7.0,1.79, recall , fontsize=14, ha="center", va="center")
```

```
plt.show()
```




```
accuracy = accuracy_score( test_verd,  test_pred )
```

```
precision = precision_score( test_verd,  test_pred,average='micro' )
```

```
recall = recall_score( test_verd,  test_pred, average='micro' )
```

```
f1 = f1_score( test_verd,  test_pred, average='micro' )
```

Haz doble clic (o pulsa Intro) para editar

Haz doble clic (o pulsa Intro) para editar

Plot the model predictions:

Export and reload your model

Now that you've trained the model, export it as a SavedModel for reusing it later.

```
t = time.time()

export_path = "/tmp/saved_models/{}".format(int(t))
model.save('model_h5_one_line.h5')

export_path
```

```
'/tmp/saved_models/1662733280'
```

Confirm that you can reload the SavedModel and that the model is able to output the same results:

```
#/tmp/saved_models/1659915398
reloaded = tf.keras.models.load_model(export_path)


result_batch = model.predict(image_batch)
reloaded_result_batch = reloaded.predict(image_batch)


abs(reloaded_result_batch - result_batch).max()
```

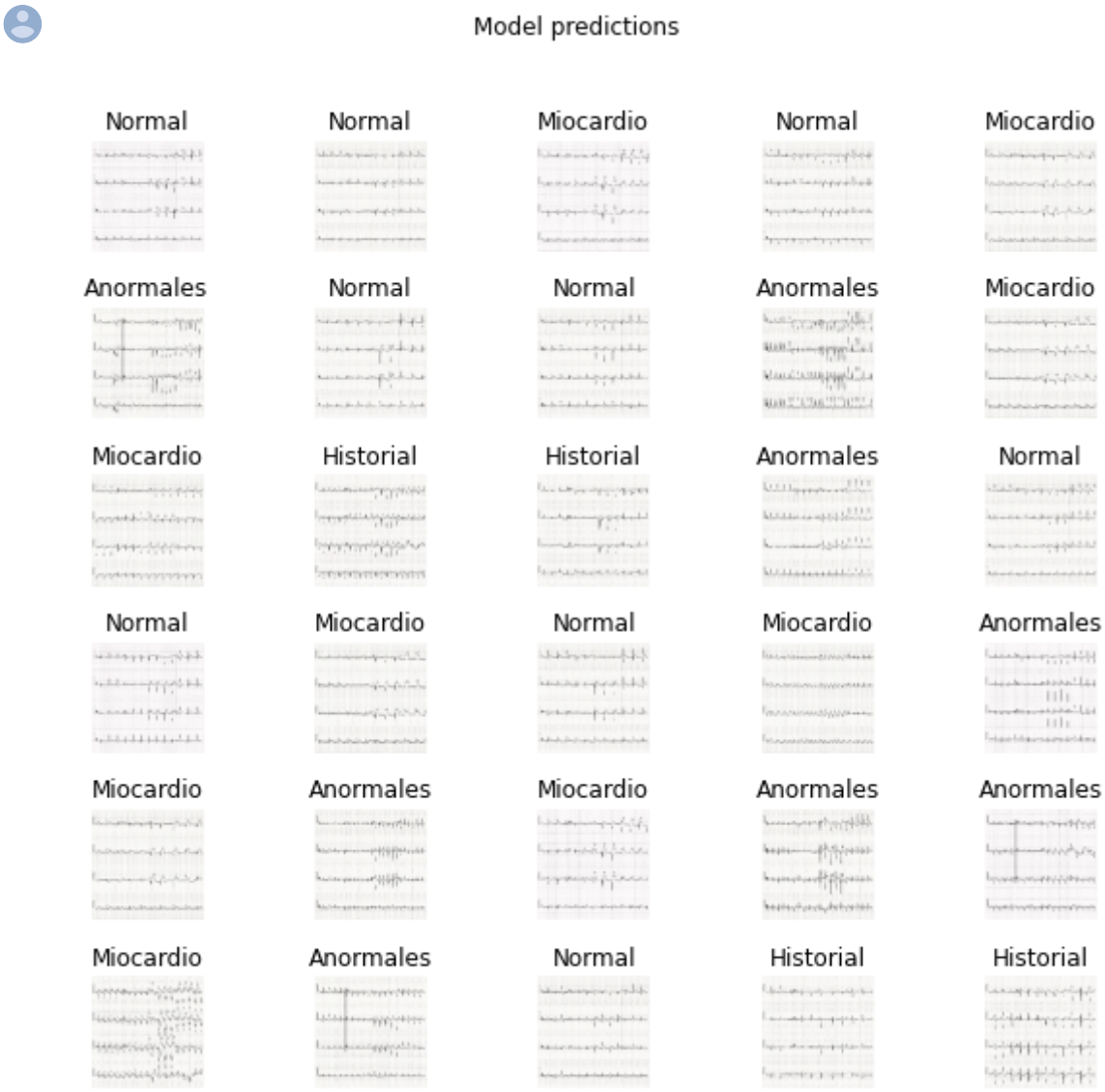
```
0.0
```

```
reloaded_predicted_id = tf.math.argmax(reloaded_result_batch, axis=-1)
reloaded_predicted_label_batch = class_names[reloaded_predicted_id]
print(reloaded_predicted_label_batch)
```

```
['anormal' 'history' 'anormal' 'anormal' 'anormal' 'miocardio' 'anormal'
 'normal' 'miocardio' 'normal' 'normal' 'history' 'anormal' 'normal'
 'normal' 'history' 'miocardio' 'miocardio' 'history' 'anormal' 'history'
 'normal' 'anormal' 'anormal' 'miocardio' 'history' 'anormal' 'miocardio'
 'history' 'miocardio' 'miocardio' 'normal' 'normal' 'anormal' 'anormal'
 'anormal' 'normal' 'miocardio' 'anormal' 'miocardio' 'miocardio'
 'anormal' 'normal' 'history' 'normal' 'miocardio' 'anormal' 'normal'
 'normal' 'anormal' 'miocardio' 'anormal' 'normal' 'miocardio' 'normal'
 'normal' 'miocardio' 'miocardio' 'anormal' 'normal' 'miocardio' 'anormal'
 'anormal' 'history' 'anormal' 'anormal' 'normal' 'anormal' 'normal'
 'normal' 'normal' 'history' 'anormal' 'history' 'miocardio' 'history'
 'miocardio' 'miocardio' 'anormal' 'normal' 'normal' 'normal' 'miocardio'
 'normal' 'anormal' 'miocardio' 'miocardio' 'normal' 'miocardio' 'normal'
 'anormal' 'miocardio' 'history' 'normal' 'miocardio' 'miocardio' 'normal'
 'history' 'anormal' 'normal' 'anormal' 'anormal' 'miocardio' 'anormal'
 'normal' 'history' 'history' 'history' 'anormal' 'miocardio' 'miocardio'
 'anormal' 'history' 'normal' 'anormal' 'normal' 'anormal' 'history'
 'miocardio' 'history' 'normal' 'anormal' 'miocardio' 'normal' 'history'
 'miocardio' 'anormal' 'anormal' 'history' 'miocardio' 'miocardio'
 'history' 'miocardio' 'miocardio' 'anormal' 'normal' 'normal' 'normal'
```

```
       'normal' 'anormal' 'miocardio' 'anormal' 'miocardio' 'normal' 'miocardio'
       'anormal' 'normal' 'history' 'anormal' 'history' 'miocardio' 'miocardio'
       'history' 'anormal' 'normal' 'normal' 'miocardio' 'miocardio' 'anormal'
       'normal' 'history' 'anormal' 'anormal' 'normal' 'miocardio' 'normal'
       'miocardio' 'miocardio' 'normal' 'anormal']
```

```
plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(30):
  plt.subplot(6,5,n+1)
  plt.imshow(image_batch[n])
  plt.title(reloaded_predicted_label_batch[n].title())
  plt.axis('off')
_ = plt.suptitle("Model predictions")
```




```
#Categorizar una imagen de internet
from PIL import Image
import requests
from io import BytesIO
import cv2

def categorizar():
  #respuesta = requests.get(url)
```

```
  img = Image.open('/content/ELECTRO 2.JPG')
  img = np.array(img).astype(float)/255

  img = cv2.resize(img, (224,224))
  prediccion = reloaded.predict(img.reshape(-1, 224, 224, 3))
  return np.argmax(prediccion[0], axis=-1)


#0 = History, 1 = Myocardial, 2 = Normal 3 = abnormal
prediccion = categorizar ()
print(class_names[prediccion])
```

```
normal
```

```
print(class_names )
```

```
['anormal' 'history' 'miocardio' 'normal']
```

```
from matriz_confusion import graficar_matriz_de_confusion
```

```
---------------------------------------------------------------------------
--
ModuleNotFoundError                       Traceback (most recent call
last)
<ipython-input-40-44704fe37b65> in <module>()
----> 1 from matriz_confusion import graficar_matriz_de_confusion

ModuleNotFoundError: No module named 'matriz_confusion'

---------------------------------------------------------------------------
--
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
```

Next steps

You can use the SavedModel to load for inference or convert it to a TensorFlow Lite model (for on-device machine learning) or a TensorFlow.js model (for machine learning in JavaScript).

Discover more tutorials to learn how to use pre-trained models from TensorFlow Hub on image, text, audio, and video tasks.

```
print(train_ds)
```

```
<PrefetchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32
```

```
c_m = confusion_matrix(y_true, y_predicted)

# Showing Confusion Matrix in form of 2D Numpy array
print(c_m)
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call
last)
<ipython-input-59-7ca8cdea42da> in <module>()
----> 1 c_m = confusion_matrix(y_true, y_predicted)
      2
      3 # Showing Confusion Matrix in form of 2D Numpy array
      4 print(c_m)

NameError: name 'y_true' is not defined
```

```
# Magic function that renders the figure in a jupyter notebook
# instead of displaying a figure object
%matplotlib inline


# Setting default size of the plot
# Setting default fontsize used in the plot
plt.rcParams['figure.figsize'] = (10.0, 9.0)
plt.rcParams['font.size'] = 20


# Implementing visualization of Confusion Matrix
display_c_m = ConfusionMatrixDisplay(c_m, display_labels=['ANORMALES' 'HISTORIAL' 'MIOCARD


# Plotting Confusion Matrix
# Setting colour map to be used
display_c_m.plot(cmap='OrRd', xticks_rotation=25)
# Other possible options for colour map are:
# 'autumn_r', 'Blues', 'cool', 'Greens', 'Greys', 'PuRd', 'copper_r'


# Setting fontsize for xticks and yticks
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)


# Giving name to the plot
plt.title('Confusion Matrix', fontsize=24)


# Saving plot
plt.savefig('confusion_matrix.png', transparent=True, dpi=500)
```

```
# Showing the plot
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-60-06d14bda1cf2> in <module>()
     11
     12 # Implementing visualization of Confusion Matrix
---> 13 display_c_m = ConfusionMatrixDisplay(c_m, display_labels=['ANORMALES'
'HISTORIAL' 'MIOCARDIO' 'NORMAL'])
     14
     15

NameError: name 'c_m' is not defined
```

SEARCH STACK OVERFLOW

++

```
modelp = tensorflow.keras.models.load_model('/content/drive/MyDrive/Test
_GAN/Saved models/CNN/x64/modelo_Covid_DCGAN_nadamx64.h5', compile=False
)
```

```
validation_generator = test_datagen.flow_from_directory(
 validation_dir,
 target_size=IMAGE_SIZE,
 batch_size=32,
 color_mode='grayscale',
 class_mode='binary',
 shuffle=False)
```

FORMA WILSON

```
from sklearn.metrics import confusion_matrix
test_predictions = modelp.predict_generator(validation_generator, 189.31 )
test_predictions = (test_predictions > 0.5)
print(test_predictions)
print(test_predictions.size)
validation_generator.classes
validation_generator.classes.size
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(validation_generator.classes, test_predictions)
print('Confusion Matrix')
print(cm)
import matplotlib.pyplot as plt
import seaborn as sns

fig, ax = plt.subplots(figsize=(5, 5))
ax.matshow(cm, cmap=plt.cm.cividis_r, alpha=0.5)
for i in range(cm.shape[0]):
 for j in range(cm.shape[1]):
  ax.text(x=j, y=i, s=cm[i, j], va='center', ha='center')

plt.title("Matriz de confusión", fontsize=15)
plt.xlabel('Predicciones')
plt.ylabel('Valores verdaderos o etiquetas')
plt.tight_layout()
plt.show()
```

```
  File "<ipython-input-64-62c5faf9f72a>", line 3
    test predictions = (test predictions > 0.5)
                     ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW