CASAVA DATASETS with TensorFlow Hub - TFLite

Cassava consists of leaf images for the cassava plant depicting healthy and four (4) disease conditions; Cassava Mosaic Disease (CMD), Cassava Bacterial Blight (CBB), Cassava Greem Mite (CGM) and Cassava Brown Streak Disease (CBSD). Dataset consists of a total of 9430 labelled images. The 9430 labelled images are split into a training set (5656), a test set(1885) and a validation set (1889). The number of images per class are unbalanced with the two disease classes CMD and CBSD having 72% of the images.

Homepage: https://www.kaggle.com/c/cassava-disease/overview

Source code: tfds.image_classification.Cassava

Run in Google Colab View source on GitHub

Setup

```
try:
    %tensorflow_version 2.x
except:
    pass
# Load the TensorBoard notebook extension.
%load_ext tensorboard
```

```
from datetime import datetime
import io
import itertools
from packaging import version
from six.moves import range
import sklearn.metrics


import numpy as np
import matplotlib.pylab as plt

import tensorflow as tf
import tensorflow_hub as hub

from tqdm import tqdm
```

```
print("\u2022 Using TensorFlow Version:", tf.__version__)
print("\u2022 Using TensorFlow Hub Version: ", hub.__version__)
print('\u2022 GPU Device Found.' if tf.test.is_gpu_available() else '\u2022 GPU Device Not
```

```
• Using TensorFlow Version: 2.3.0
• Using TensorFlow Hub Version:  0.10.0
WARNING:tensorflow:From <ipython-input-2-8abb7064da77>:19: is_gpu_available (from te
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
• GPU Device Found.
```

Select the Hub/TF2 Module to Use

Hub modules for TF 1.x won't work here, please use one of the selections provided.

```
module_selection = ("inception_v3", 299, 2048) #@param ["(\"mobilenet_v2\", 224, 1280)", '
handle_base, pixels, FV_SIZE = module_selection
MODULE_HANDLE ="https://tfhub.dev/google/tf2-preview/{}/feature_vector/4".format(handle_ba
IMAGE_SIZE = (pixels, pixels)
print("Using {} with input size {} and output dimension {}".format(MODULE_HANDLE, IMAGE_SI
```

module_selection: ("inception_v3"

```
Using https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector/4 with input
```

Data Preprocessing

Use [TensorFlow Datasets] to load the cats and dogs dataset.

This `tfds` package is the easiest way to load pre-defined data. If you have your own data, and are interested in importing using it with TensorFlow see [loading image data]

```
import tensorflow_datasets as tfds
#tfds.disable_progress_bar()
```

The `tfds.load` method downloads and caches the data, and returns a `tf.data.Dataset` object. These objects provide powerful, efficient methods for manipulating data and piping it into your model.

Since `"cats_vs_dog"` doesn't define standard splits, use the subsplit feature to divide it into (train, validation, test) with 80%, 10%, 10% of the data respectively.

```
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
#splits = tfds.Split.(weighted=(80, 10, 10))

splits, info = tfds.load('cassava', with_info=True, as_supervised=True, split = ['train' ,
```

```
Downloading and preparing dataset cassava/0.1.0 (download: 1.26 GiB, generated: Unkn
```

Dl Completed...: 100% 1/1 [00:36<00:00, 36.79s/ url]

Dl Size...: 100% 1291/1291 [00:36<00:00, 35.12 MiB/s]

Extraction completed...: 100% 1/1 [00:36<00:00, 36.72s/ file]

```
Shuffling and writing examples to /root/tensorflow_datasets/cassava/0.1.0.incomplete
```

100% 5654/5656 [00:30<00:00, 125.12 examples/s]

```
Shuffling and writing examples to /root/tensorflow_datasets/cassava/0.1.0.incomplete
```

99% 1874/1885 [00:04<00:00, 346.01 examples/s]

```
Shuffling and writing examples to /root/tensorflow_datasets/cassava/0.1.0.incomplete
```

98% 1852/1889 [00:07<00:00, 120.98 examples/s]

```
Dataset cassava downloaded and prepared to /root/tensorflow_datasets/cassava/0.1.0.
```

```
(train_examples, validation_examples, test_examples) = splits
```

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




print(num_classes)
```

```
5
```

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

```
clas = np.array(info.features.items)
print(clas)
```

```
<bound method FeaturesDict.items of FeaturesDict({
    'image': Image(shape=(None, None, 3), dtype=tf.uint8),
    'image/filename': Text(shape=(), dtype=tf.string),
    'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=5),
})>
```

```
print(class_names)
```

```
['cbb' 'cbsd' 'cgm' 'cmd' 'healthy']
```

```
from collections import Counter
import pandas  as pd

counts =[]

for _ , train_labels in train_batches.take(178):
  counts.extend(train_labels.numpy())
a = dict(Counter(counts))
```

```
print(a)
```

```
{4: 316, 2: 773, 1: 1443, 3: 2658, 0: 466}
```

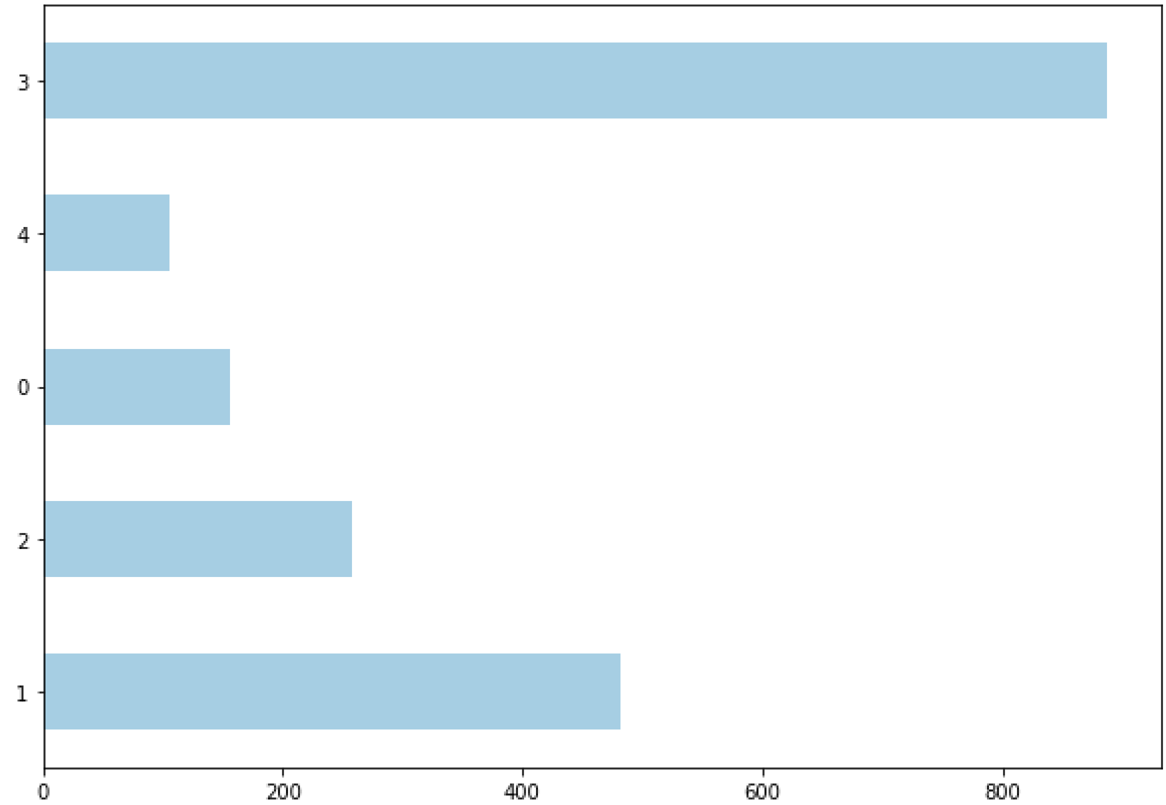
```
class_weight = {0: 5.7,
                1: 1.8,
                2: 3.4,
                3: 1.,
                4: 8.4}
```

```
df_dis = pd.DataFrame.from_dict(a , orient='index')
df_dis.plot.barh(figsize = (10,7) , legend =False, colormap='Paired' )

print(class_names)
```

```
counts =[]

for _ , train_labels in validation_batches.take(178):
  counts.extend(train_labels.numpy())
a = dict(Counter(counts))
```

```
df_dis = pd.DataFrame.from_dict(a , orient='index')
df_dis.plot.barh(figsize = (10,7) , legend =False, colormap='Paired' )

print(class_names)
```

    ['cbb' 'cbsd' 'cgm' 'cmd' 'healthy']



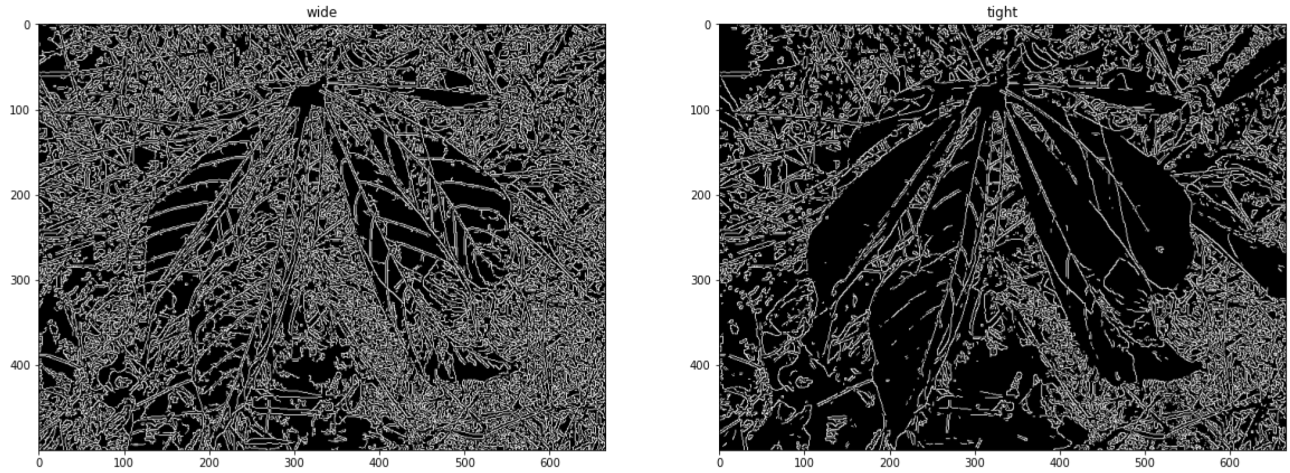

```
from collections import Counter
import pandas  as pd

counts =[]

for _ , train_labels in train_batches.take(178):
  counts.extend(train_labels.numpy())
a = dict(Counter(counts))

df_dis = pd.DataFrame.from_dict(a , orient='index')
df_dis.plot.barh(figsize = (30,10) , legend =False, colormap='Paired' )

print(class_names)
```

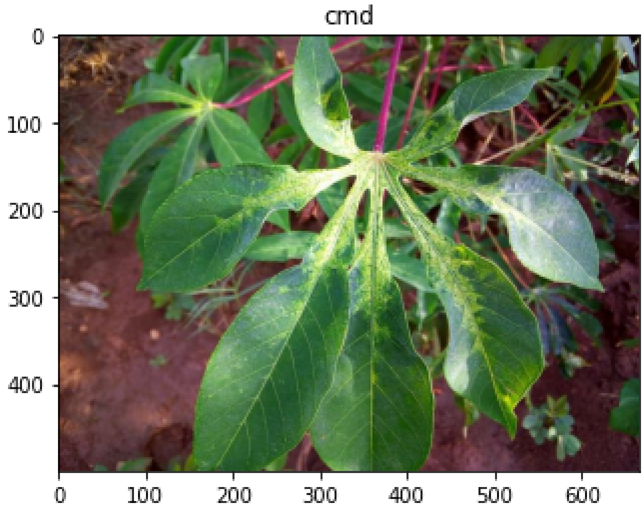
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

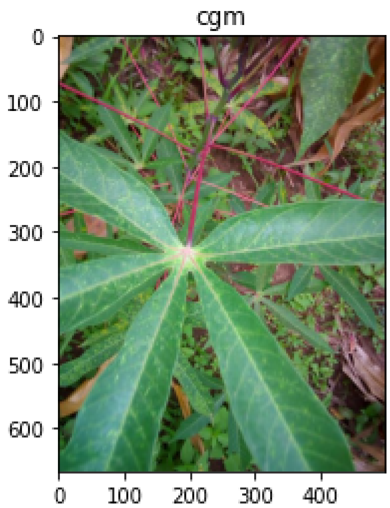
```
%matplotlib inline
```

```
get_label_name = info.features['label'].int2str
#for i in range(num_classes):
 #print(get_label_name(i))

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,10))
for image, label in train_examples.take(4):
  #print(image.shape)
  gray = cv2.cvtColor(image.numpy(), cv2.COLOR_RGB2GRAY)
  # Try Canny using "wide" and "tight" thresholds
  wide = cv2.Canny(gray, 30, 100)
  tight = cv2.Canny(gray, 200, 240)

  ax1.set_title('wide')
  ax1.imshow(wide, cmap='gray')

  ax2.set_title('tight')
  ax2.imshow(tight, cmap='gray')

# Display the images


  plt.figure()
  plt.imshow(image)
  print(label)
  plt.title(get_label_name(label))
```

```
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
```

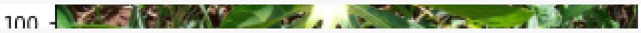






cmd

```
train_label = [ label for image, label in train_examples]
```




```

#print(train_images)

# Clear out prior logging data.
!rm -rf logs/plots

logdir = "logs/plots/" + datetime.now().strftime("%Y%m%d-%H%M%S")
file_writer = tf.summary.create_file_writer(logdir)

def plot_to_image(figure):
  """Converts the matplotlib plot specified by 'figure' to a PNG image and
  returns it. The supplied figure is closed and inaccessible after this call."""
  # Save the plot to a PNG in memory.
  buf = io.BytesIO()
  plt.savefig(buf, format='png')
  # Closing the figure prevents it from being displayed directly inside
  # the notebook.
  plt.close(figure)
  buf.seek(0)
  # Convert PNG buffer to TF image
  image = tf.image.decode_png(buf.getvalue(), channels=4)
  # Add the batch dimension
  image = tf.expand_dims(image, 0)
  return image

def image_grid():
  """Return a 5x5 grid of the MNIST images as a matplotlib figure."""
  # Create a figure to contain the plot.
  figure = plt.figure(figsize=(20,20))
  for i in range(25):
    # Start next subplot.
    plt.subplot(5, 5, i + 1, title=class_names[train_label[i]])
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow([image for image, label in train_examples.take(28)][i] )

  return figure

# Prepare the plot
figure = image_grid()
# Convert to image and log
with file_writer.as_default():
  tf.summary.image("Training data", plot_to_image(figure), step=0)

#
```

```
%tensorboard --logdir logs/plots
```

Format the Data

Use the `tf.image` module to format the images for the task.

Resize the images to a fixes input size, and rescale the input channels

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  tf.keras.layers.experimental.preprocessing.RandomRotation(0.2),
  tf.keras.layers.experimental.preprocessing.RandomContrast(),
  tf.keras.layers.experimental.preprocessing.RandomZoom(),
  tf.keras.layers.experimental.preprocessing.RandomTranslation()
])
```

```
def format_image(image, label):
    image = tf.cast(image, tf.float32)
    #image = tf.image.grayscale_to_rgb(image)
    image = tf.image.resize(image, IMAGE_SIZE , preserve_aspect_ratio=False)
    image = tf.image.random_brightness(image, max_delta = 0.3)
    #image = tf.image.random_contrast(image, 0.2, 0.5)
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    print(image.get_shape)
    return  image, label

def format_image_valid(image, label):
    image = tf.cast(image, tf.float32)
    #image = tf.image.grayscale_to_rgb(image)
    image = tf.image.resize(image, IMAGE_SIZE , preserve_aspect_ratio=False)
    print(image.get_shape)
    return  image, label
```

Now shuffle and batch the data

```
BATCH_SIZE =  32#@param {type:"integer"}
```

BATCH_SIZE: 32

```
train_batches = train_examples.shuffle(1000).map(format_image).batch(BATCH_SIZE).prefetch(


validation_batches = validation_examples.map(format_image_valid).batch(BATCH_SIZE).prefetc

cm_validation_batches = validation_examples.map(format_image_valid).batch(2160).prefetch(1


test_batches =       test_examples.map(format_image_valid).batch(32).prefetch(1)
```

```
<bound method Tensor.get_shape of <tf.Tensor 'random_flip_up_down/Identity:0' shape=
```

```
<bound method Tensor.get_shape of <tf.Tensor 'resize/Squeeze:0' shape=(299, 299, 3)
<bound method Tensor.get_shape of <tf.Tensor 'resize/Squeeze:0' shape=(299, 299, 3)
<bound method Tensor.get_shape of <tf.Tensor 'resize/Squeeze:0' shape=(299, 299, 3)
```

```
test_batches =       test_examples.map(format_image).batch(32).prefetch(1)
```

```
<bound method Tensor.get_shape of <tf.Tensor 'truediv:0' shape=(299, 299, 3) dtype=f
```

```
print(0.90*2400)
```

Inspect a batch

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

image_batch.shape
```

```


a,  = cm_validation_batches.take(1)
im , l = a[0], a[1]





print(im.shape)
```

```
from matplotlib.colors import Normalize
import matplotlib.cm  as cm



def class_distribution(train_examples , validation_examples , test_examples):
        train_label_plot = [ label for image, label in train_examples]
        valid_label_plot = [ label for image, label in validation_examples]
        test_label_plot = [ label for image, label in test_examples]
        unique, counts = np.unique(train_label_plot, return_counts=True)
        #print(unique)

        my_cmap = cm.get_cmap('jet')
        plt.figure(figsize=(20,70))
        # Get normalize function (takes data in range [vmin, vmax] -> [0, 1])
        my_norm = Normalize(vmin=0, vmax=196)
        plt.barh(unique, counts ,color=my_cmap(my_norm(unique)))

        plt.yticks(unique, class_names)
        plt.title('Class Frequency')
        plt.xlabel('Class')
        plt.ylabel('Frequency')
        plt.show()
```

```
        fig = plt.gcf

        return fig

class_distribution(train_examples , validation_examples , test_examples)
```

Defining the Model

All it takes is to put a linear classifier on top of the `feature_extractor_layer` with the Hub module.

For speed, we start out with a non-trainable `feature_extractor_layer`, but you can also enable fine-tuning for greater accuracy.

```
do_fine_tuning = True #@param {type:"boolean"}
```

do_fine_tuning: ☑

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

```
print("Building model with", MODULE_HANDLE)

model = tf.keras.Sequential([
        tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape=(299,299
        tf.keras.layers.experimental.preprocessing.RandomRotation(0.4),
        tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
        feature_extractor,
        tf.keras.layers.Dense(num_classes, activation='softmax')
])

model.summary()
```

```
Building model with https://tfhub.dev/google/tf2-preview/inception_v3/feature_vector
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_4 (Rescaling)      (None, 299, 299, 3)       0
_________________________________________________________________
random_rotation_4 (RandomRot (None, 299, 299, 3)       0
_________________________________________________________________
random_zoom_3 (RandomZoom)   (None, 299, 299, 3)       0
_________________________________________________________________
keras_layer_4 (KerasLayer)   (None, 2048)              21802784
```

```
dense_13 (Dense)             (None, 5)                 10245
=================================================================
Total params: 21,813,029
Trainable params: 21,778,597
Non-trainable params: 34,432
_________________________________________________________________
```

```
#@title (Optional) Unfreeze some layers
NUM_LAYERS = 1 #@param {type:"slider", min:1, max:50, step:1}

if do_fine_tuning:
    feature_extractor.trainable = False

    for layer in model.layers[-NUM_LAYERS:]:
        layer.trainable = True

else:
    feature_extractor.trainable = False
```

(Optional) Unfreeze some layers

NUM_LAYERS: 1

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_4 (Rescaling)      (None, 299, 299, 3)       0
_________________________________________________________________
random_rotation_4 (RandomRot (None, 299, 299, 3)       0
_________________________________________________________________
random_zoom_3 (RandomZoom)   (None, 299, 299, 3)       0
_________________________________________________________________
keras_layer_4 (KerasLayer)   (None, 2048)              21802784
_________________________________________________________________
dense_13 (Dense)             (None, 5)                 10245
=================================================================
Total params: 21,813,029
Trainable params: 10,245
Non-trainable params: 21,802,784
_________________________________________________________________
```

Training the Model

```
if do_fine_tuning:
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.002),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])
else:
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.002),
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=
early_stop_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5 , mo
```

```
early_stop_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, mo
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_4 (Rescaling)      (None, 299, 299, 3)       0
_________________________________________________________________
random_rotation_4 (RandomRot (None, 299, 299, 3)       0
_________________________________________________________________
random_zoom_3 (RandomZoom)   (None, 299, 299, 3)       0
_________________________________________________________________
keras_layer_4 (KerasLayer)   (None, 2048)              21802784
_________________________________________________________________
dense_13 (Dense)             (None, 5)                 10245
=================================================================
Total params: 21,813,029
Trainable params: 10,245
Non-trainable params: 21,802,784
_________________________________________________________________
```

```
!rm -rf logs/image

logdir = "logs/image/" + datetime.now().strftime("%Y%m%d-%H%M%S")
# Define the basic TensorBoard callback.
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
file_writer_cm = tf.summary.create_file_writer(logdir + '/cm')
file_writer_roc = tf.summary.create_file_writer(logdir + '/roc')
```

```
def plot_confusion_matrix(cm, class_names):
  """
  Returns a matplotlib figure containing the plotted confusion matrix.

  Args:
    cm (array, shape = [n, n]): a confusion matrix of integer classes
    class_names (array, shape = [n]): String names of the integer classes
  """
  figure = plt.figure(figsize=(8, 8))
  plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
  plt.title("Confusion matrix")
  plt.colorbar()
  tick_marks = np.arange(len(class_names))
  plt.xticks(tick_marks, class_names, rotation=45)
  plt.yticks(tick_marks, class_names)

  # Normalize the confusion matrix.
  cm = np.around(cm.astype('float') / cm.sum(axis=1)[:, np.newaxis], decimals=2)

  # Use white text if squares are dark; otherwise black.
  threshold = cm.max() / 2.
  for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    color = "white" if cm[i, j] > threshold else "black"
    plt.text(j, i, cm[i, j], horizontalalignment="center", color=color)

  plt.tight_layout()
```

```
  plt.ylabel('True label')
  plt.xlabel('Predicted label')
  return figure
```

```


def log_confusion_matrix(epoch, logs):
  # Use the model to predict the values from the validation dataset.
  test_pred_raw = model.predict(im)
  test_pred = np.argmax(test_pred_raw, axis=1)

      # Calculate the confusion matrix.
  cm = sklearn.metrics.confusion_matrix(l, test_pred)
  # Log the confusion matrix as an image summary.
  figure = plot_confusion_matrix(cm, class_names=class_names)
  cm_image = plot_to_image(figure)

  # Log the confusion matrix as an image summary.
  with file_writer_cm.as_default():
    tf.summary.image("Confusion Matrix", cm_image, step=epoch)

# Define the per-epoch callback.
cm_callback = tf.keras.callbacks.LambdaCallback(on_epoch_end=log_confusion_matrix)
```

```
!pip install scikit-plot
```

```
import scikitplot as skplt

def plot_roc(y_true, y_probas):
  #figure = plt.figure(figsize=(8, 8))
  figure, axes = plt.subplots(1,1, figsize = (8,8))

  skplt.metrics.plot_roc_curve(y_true, y_probas , ax =axes ,text_fontsize ='small', figsiz
  plt.title('ROC')
  #plt.colorbar()
  plt.ylabel('True Label')
  plt.xlabel('Predicated Label')
  fig = plt.gcf()
  #plt.show()
  fig.savefig("test_rasterization.png", dpi=150)
  #print(fig)


  return fig
```

```
def log_roc(epoch, logs):
  test_pred_raw = model.predict(im)
  test_pred = np.argmax(test_pred_raw, axis=1)
  #print(test_pred.shape)
  #print(l.shape)
  figure_roc = plot_roc(l, test_pred_raw)
  roc_image = plot_to_image(figure_roc)
```

```
  # Log the roc  as an image summary.
  with file_writer_roc.as_default():
    tf.summary.image("ROC", roc_image, step=epoch)

roc_callback = tf.keras.callbacks.LambdaCallback( on_epoch_end=log_roc)
```

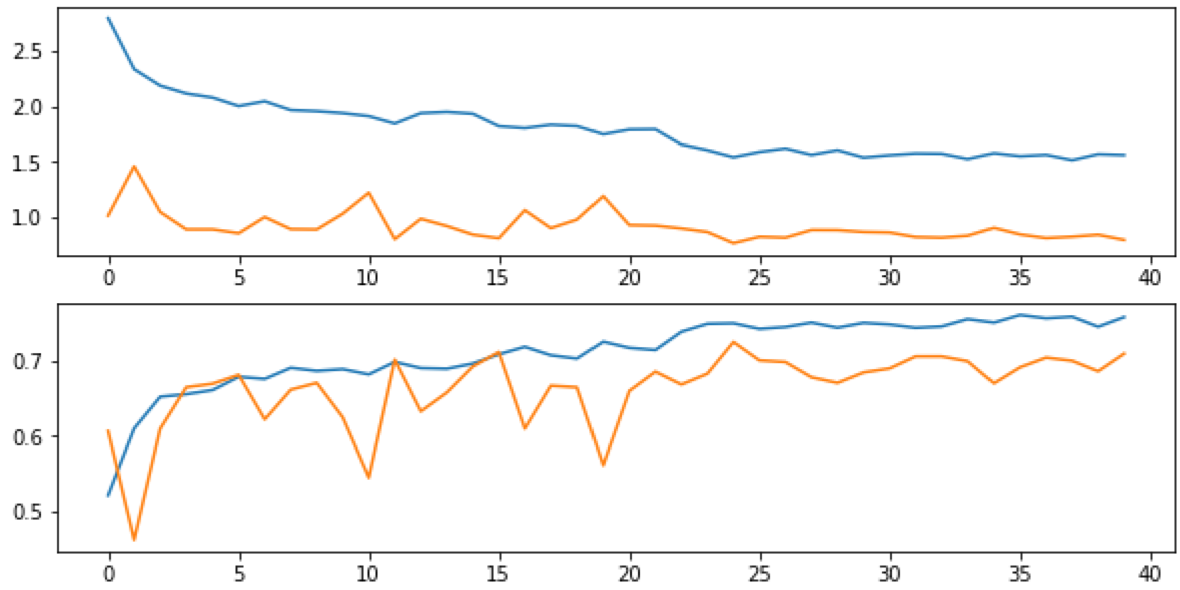
Double-click (or enter) to edit

```
# learning rate dense nodes accuacy
#  0.0002           101      16%
#  0.002            1280     15%
```

```
EPOCHS = 40
# Start TensorBoard.


%tensorboard --logdir logs/




hist = model.fit(train_batches,
                 epochs=EPOCHS,
                 callbacks =[tensorboard_callback,reduce_lr],
                 validation_data=validation_batches,
                 class_weight= class_weight)
                 #callbacks=[tensorboard_callback])
```

```
from matplotlib import pyplot as plt




fig, axs = plt.subplots(2, 1 , figsize=(10,5))
#axs[0, 0].plot(x, y)
##axs[0, 0].set_title('Axis [0, 0]')
#axs[0, 1].plot(x, y, 'tab:orange')
#axs[0, 1].set_title('Axis [0, 1]')
#axs[1, 0].plot(x, -y, 'tab:green')
#axs[1, 0].set_title('Axis [1, 0]')
#axs[1, 1].plot(x, -y, 'tab:red')
#axs[1, 1].set_title('Axis [1, 1]')



#plt.yscale('log')
axs[0].plot(hist.history['loss'])
axs[0].plot(hist.history['val_loss'])
axs[1].plot(hist.history['accuracy'])
axs[1].plot(hist.history['val_accuracy'])
```

```
[<matplotlib.lines.Line2D at 0x7f0e85dd90b8>]
```




```
model.evaluate(validation_batches.take(100))
```

```
60/60 [==============================] - 8s 131ms/step - loss: 0.7271 - accuracy: 0.
[0.7271420359611511, 0.7554261684417725]
```

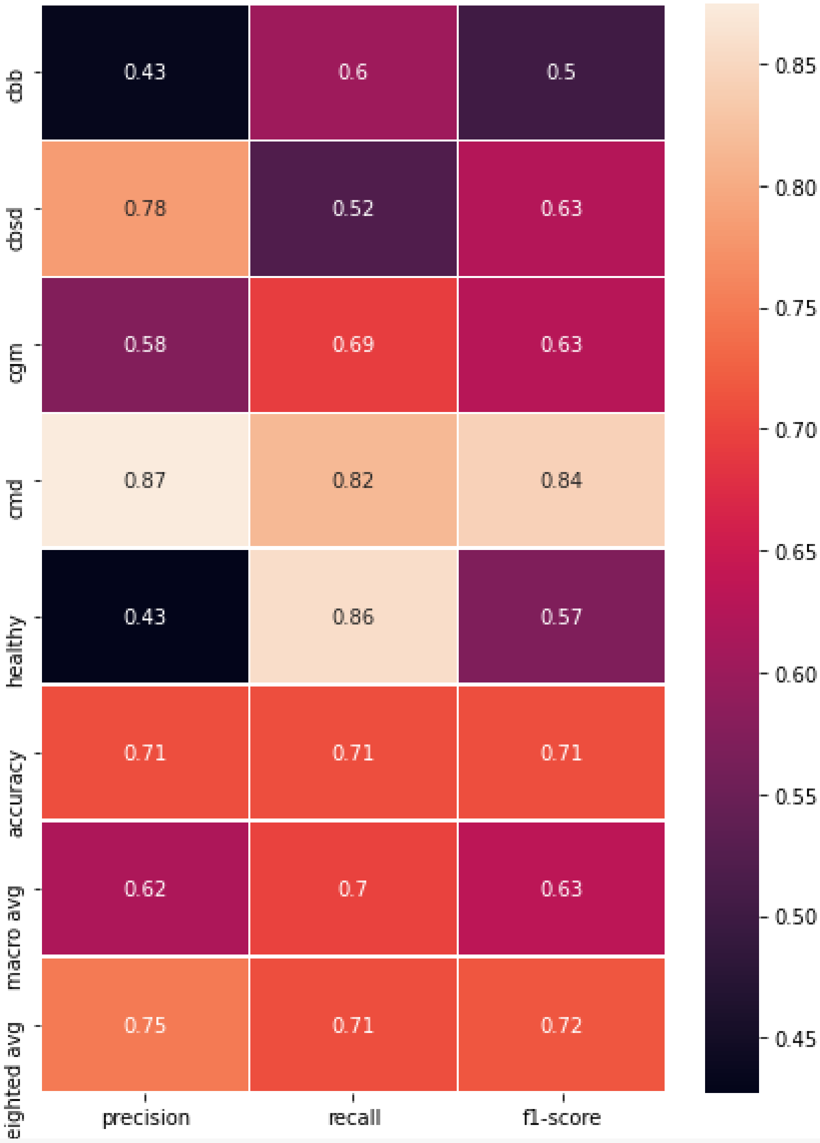
```
model.evaluate(test_batches.take(10000))
```

```
59/59 [==============================] - 8s 139ms/step - loss: 0.7836 - accuracy: 0.
[0.7835850119590759, 0.7092838287353516]
```

```
predictions = []
real_label  = []
for image_batch , labels_batch in validation_batches.take(99):

  predictions.extend(np.argmax(model.predict(image_batch), axis =-1))
  real_label.extend(labels_batch.numpy())
```

```
import seaborn as sns
import pandas as pd
from sklearn.metrics import classification_report
plt.figure(figsize = (7,10))
#print("the classification report : \n" , cl_report)
cl_report = classification_report(real_label,predictions, target_names = class_names, outp
sns.heatmap(pd.DataFrame(cl_report).iloc[:-1, :].T, annot= True, linewidths=.5)
print(class_names)
print(a)
```

```
['cbb' 'cbsd' 'cgm' 'cmd' 'healthy']
{4: 316, 2: 773, 1: 1443, 3: 2658, 0: 466}
```




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



cm = confusion_matrix(real_label, predictions)
print(cm)
```

```
[[ 94  20  15  12  15]
 [ 90 251  31  50  60]
 [ 12  15 179  37  15]
 [ 16  29  86 724  32]
 [  5   5   0   5  91]]
```

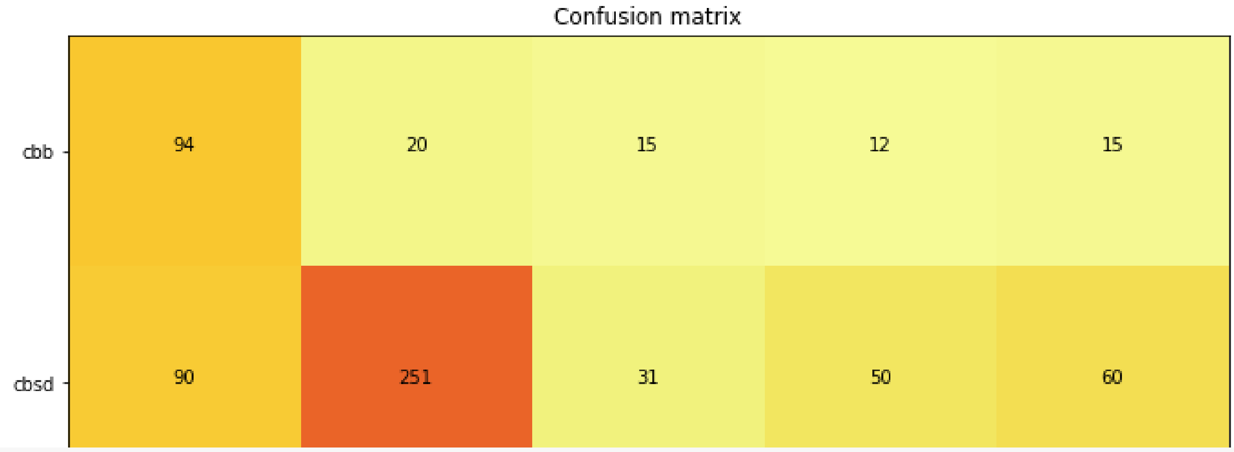
```
import numpy as np


def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
```

```
"""
given a sklearn confusion matrix (cm), make a nice plot

Arguments
---------
cm:           confusion matrix from sklearn.metrics.confusion_matrix

target_names: given classification classes such as [0, 1, 2]
              the class names, for example: ['high', 'medium', 'low']

title:        the text to display at the top of the matrix

cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
              see http://matplotlib.org/examples/color/colormaps_reference.html
              plt.get_cmap('jet') or plt.cm.Blues

normalize:    If False, plot the raw numbers
              If True, plot the proportions

Usage
-----
plot_confusion_matrix(cm           = cm,                  # confusion matrix created b
                                                          # sklearn.metrics.confusion_
                      normalize    = True,                # show proportions
                      target_names = y_labels_vals,       # list of names of the class
                      title        = best_estimator_name) # title of graph

Citiation
---------
http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.htm

"""
import matplotlib.pyplot as plt
import numpy as np
import itertools

accuracy = np.trace(cm) / np.sum(cm).astype('float')
misclass = 1 - accuracy

if cmap is None:
    cmap = plt.get_cmap('Blues')

plt.figure(figsize=(10, 10))
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title(title)
#plt.colorbar()

if target_names is not None:
    tick_marks = np.arange(len(target_names))
    plt.xticks(tick_marks, target_names, rotation=90)
    plt.yticks(tick_marks, target_names)

if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misc
    plt.show()
```

Double-click (or enter) to edit

```
plot_confusion_matrix(cm = cm,
                          target_names= class_names,
                          title='Confusion matrix',
                          cmap='inferno_r',
                          normalize=False)
```



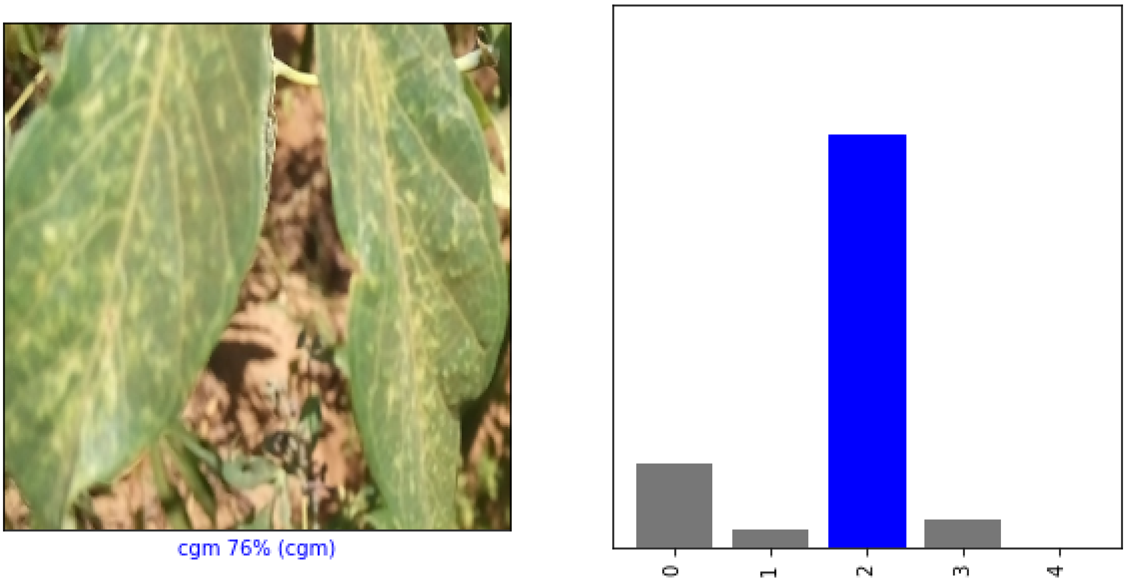

```
CLS = class_names
```

```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array, true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img.numpy().astype("uint8"))

  predicted_label = np.argmax(predictions_array)
  #print(predicted_label)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(CLS[predicted_label],
                                100*np.max(predictions_array),
                                CLS[true_label]),
                                color=color)







def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array, true_label[i]
  plt.grid(False)
  plt.xticks(range(5) , rotation =90)
  plt.yticks([])
  thisplot = plt.bar(range(5), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)
  #print(predicted_label)
  #print(true_label)
  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

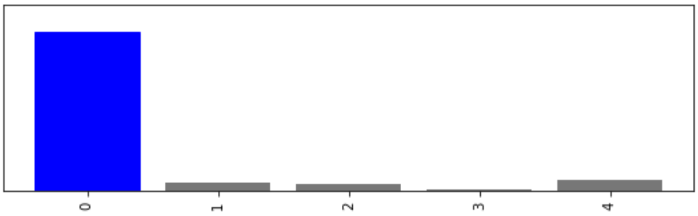
```
  thisplot[true_label].set_color('blue')
```

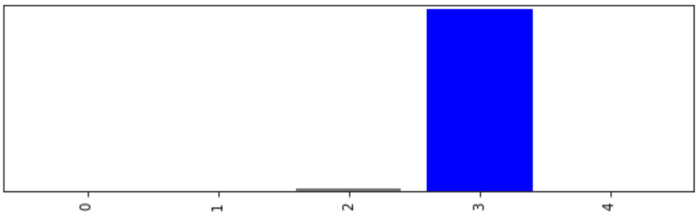
```
for imgs , lbs in test_batches.take(1):

  pred = model.predict(imgs)
```

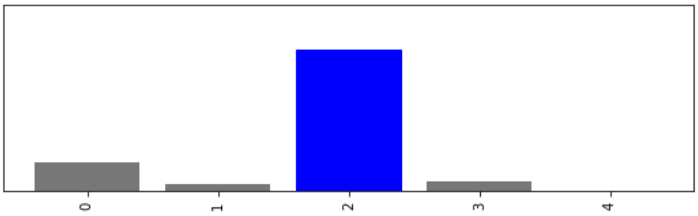
```
i = 2
plt.figure(figsize=(10,5))
plt.subplot(1,2,1)



plot_image(i, pred[i], lbs, imgs)
plt.subplot(1,2,2)
plot_value_array(i, pred[i],  lbs)
plt.show()
```

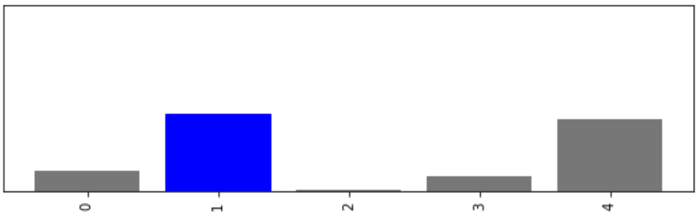


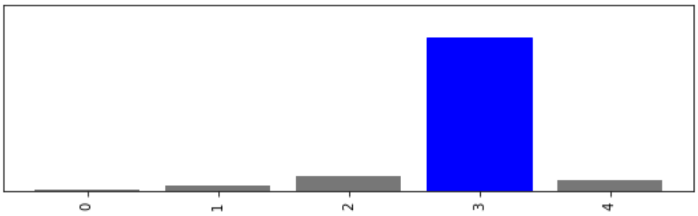

```
# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 30
num_cols = 1
num_images = num_rows*num_cols
plt.figure(figsize=(10*2*num_cols, 3*num_rows))

for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, pred[i], lbs, imgs)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, pred[i], lbs)

plt.show()
```

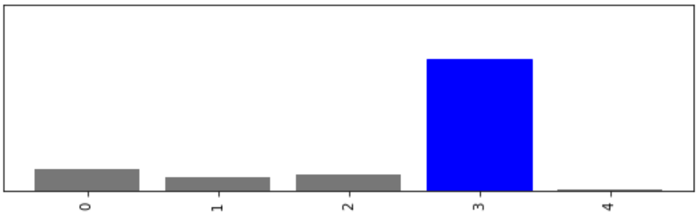

cbb 85% (cbb)


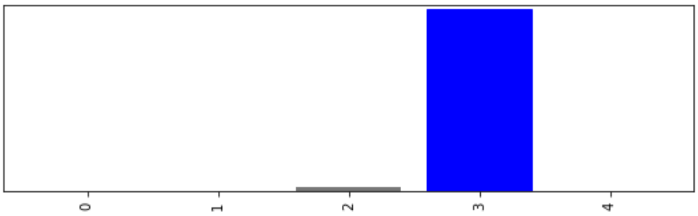

cmd 98% (cmd)


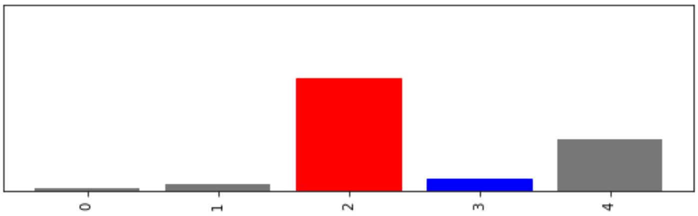

cgm 76% (cgm)


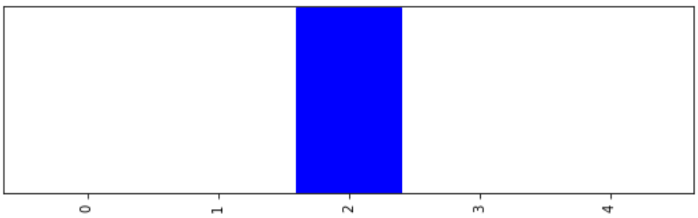

cbsd 42% (cbsd)


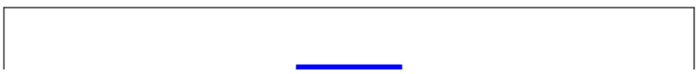

cmd 83% (cmd)



cmd 71% (cmd)


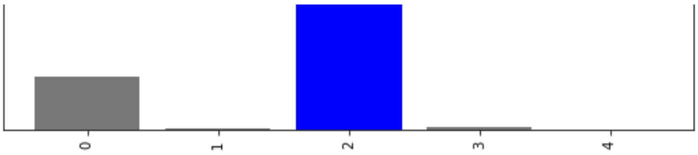

cmd 98% (cmd)



cgm 61% (cmd)


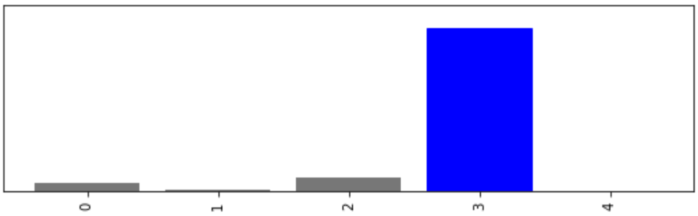

cgm 99% (cgm)


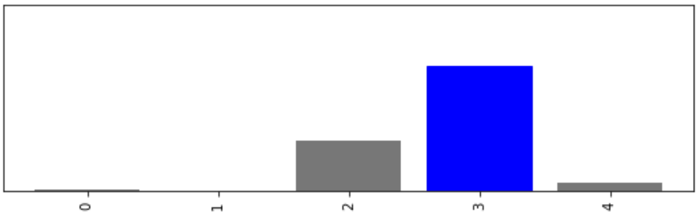
cgm 70% (cgm)



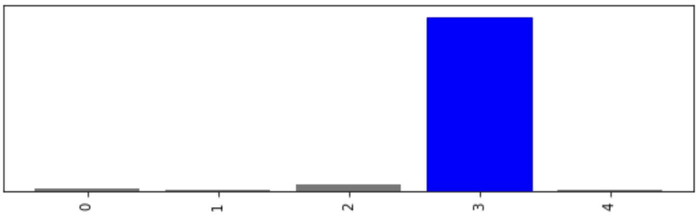
cmd 88% (cmd)



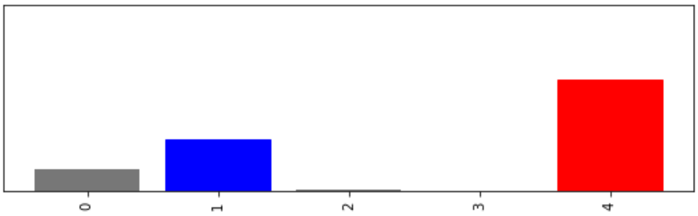
cmd 67% (cmd)



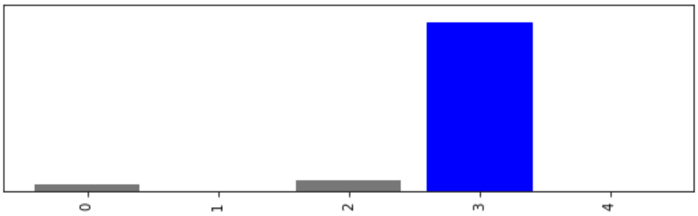
cmd 93% (cmd)



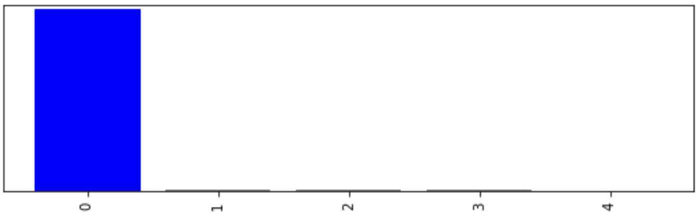
healthy 60% (cbsd)



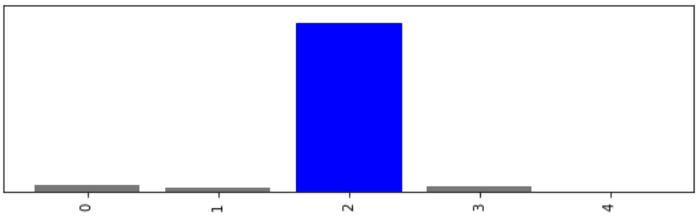
cmd 91% (cmd)



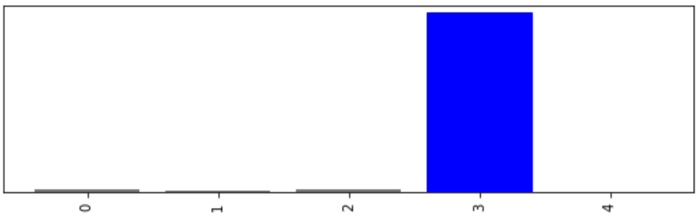
cbb 98% (cbb)



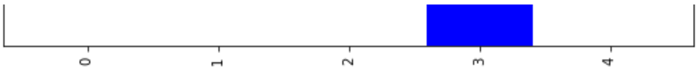
cgm 91% (cgm)



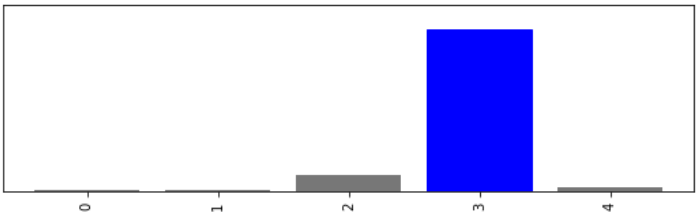
cmd 97% (cmd)



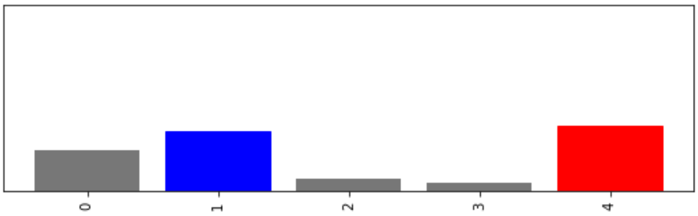

cmd 100% (cmd)



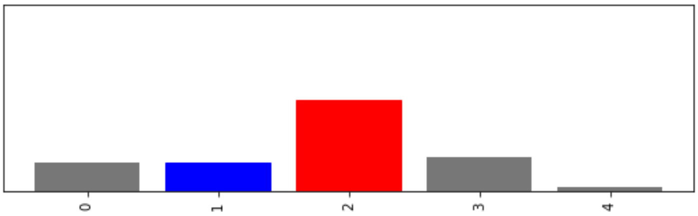

cmd 87% (cmd)



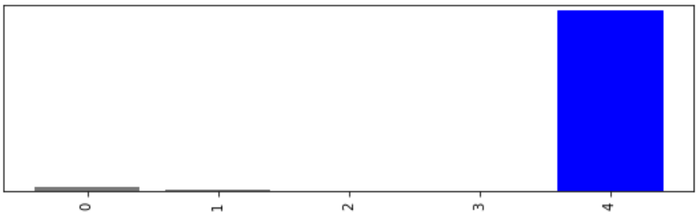

healthy 35% (cbsd)



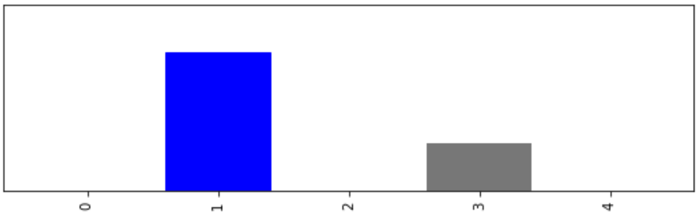

cgm 49% (cbsd)



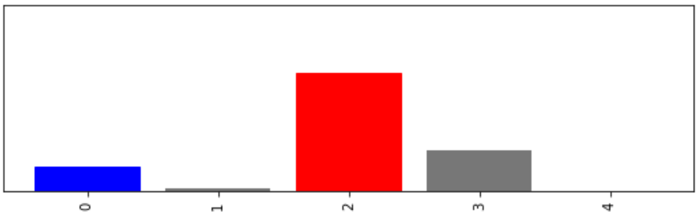

healthy 97% (healthy)



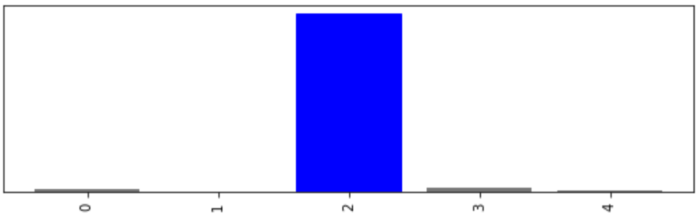

cbsd 74% (cbsd)



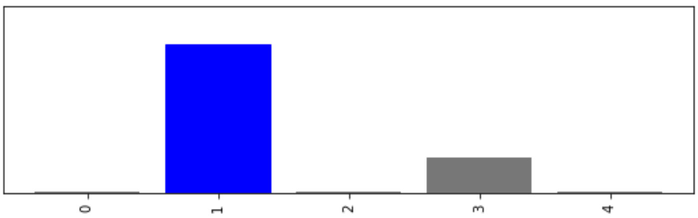

cgm 64% (cbb)



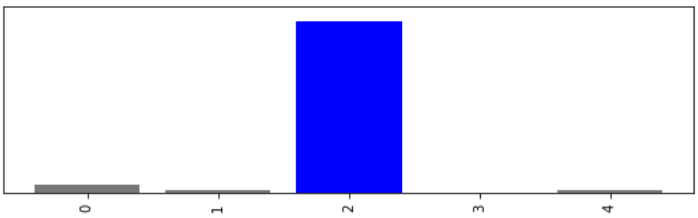

cgm 96% (cgm)



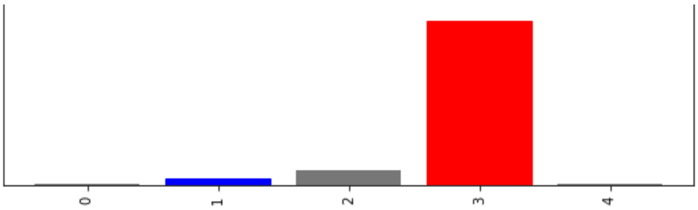

cbsd 80% (cbsd)



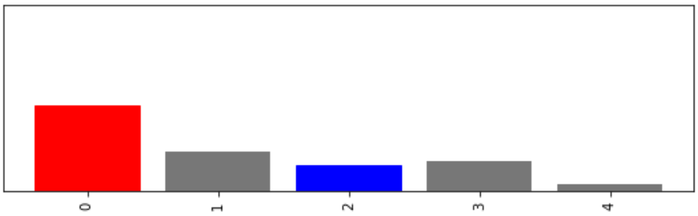

cgm 92% (cgm)



cmd 88% (cbsd)



cbb 46% (cgm)



Export the Model

```
RPS_SAVED_MODEL = "rps_saved_model"
```

Export the SavedModel

```
tf.saved_model.save(model, RPS_SAVED_MODEL)
```

```
%%bash -s $RPS_SAVED_MODEL
saved_model_cli show --dir $1 --tag_set serve --signature_def serving_default
```

```
loaded = tf.saved_model.load(RPS_SAVED_MODEL)
```

```
print(list(loaded.signatures.keys()))
infer = loaded.signatures["serving_default"]
print(infer.structured_input_signature)
print(infer.structured_outputs)
```

Convert Using TFLite's Converter

```
converter = tf.lite.TFLiteConverter.from_saved_model(RPS_SAVED_MODEL)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()
```

```
tflite_model_file = 'converted_model.tflite'

with open(tflite_model_file, "wb") as f:
    f.write(tflite_model)
```

Test the TFLite Model Using the Python Interpreter

```
# Load TFLite model and allocate tensors.
with open(tflite_model_file, 'rb') as fid:
    tflite_model = fid.read()

interpreter = tf.lite.Interpreter(model_content=tflite_model)
interpreter.allocate_tensors()

input_index = interpreter.get_input_details()[0]["index"]
output_index = interpreter.get_output_details()[0]["index"]
```

```
# Gather results for the randomly sampled test images
predictions = []

test_labels, test_imgs = [], []
for img, label in tqdm(test_batches.take(100)):
    interpreter.set_tensor(input_index, img)
    interpreter.invoke()
    predictions.append(interpreter.get_tensor(output_index))

    test_labels.append(label.numpy()[0])
    test_imgs.append(img)
```

```
#@title Utility functions for plotting
# Utilities for plotting

#class_names = ['dandelion',
```

Utility functions for plotting

```
#class_names = ['dandelion',
#'daisy',
#'tulips',
#'sunflowers',
#'roses']

def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    img = np.squeeze(img)

    plt.imshow(img[: , : , 0], cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)

    print(type(predicted_label), type(true_label))

    if predicted_label == true_label:
        color = 'green'
    else:
        color = 'red'

    plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                         100*np.max(predictions_array),
                                         class_names[true_label]), color=color)
```

```
#@title Visualize the outputs { run: "auto" }
index = 81 #@param {type:"slider", min:0, max:100, step:1}
plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plot_image(index, predictions, test_labels, test_imgs)
plt.show()
```

Visualize the outputs

index: 81

Create a file to save the labels.

```
with open('labels.txt', 'w') as f:
    f.write('\n'.join(class_names))
```

If you are running this notebook in a Colab, you can run the cell below to download the model and labels to your local disk.

Note: If the files do not download when you run the cell, try running the cell a second time. Your browser might prompt you to allow multiple files to be downloaded.

```
try:
    from google.colab import files
    files.download('converted_model.tflite')
    files.download('labels.txt')
except:
```

```
except:
    pass
```

Prepare the Test Images for Download (Optional)

This part involves downloading additional test images for the Mobile Apps only in case you need to try out more samples

```
!mkdir -p test_images
```

```
from PIL import Image

for index, (image, label) in enumerate(test_batches.take(50)):
    image = tf.cast(image * 255.0, tf.uint8)
    image = tf.squeeze(image).numpy()
    pil_image = Image.fromarray(image)
    pil_image.save('test_images/{}_{}.jpg'.format(class_names[label[0]], index))
```

```
!ls test_images
```

```
!zip -qq rps_test_images.zip -r test_images/
```

If you are running this notebook in a Colab, you can run the cell below to download the Zip file with the images to your local disk.

Note: If the Zip file does not download when you run the cell, try running the cell a second time.

```
try:
    files.download('rps_test_images.zip')
except:
    pass
```

```

```