MLCA_f2822103_f2822105

August 31, 2022

1 Classifying recyclable and organic waste using MobileNetV2

1.0.1 Authors:

Michail Kalligas, Panagiotis Lolos

This notebook contains our entire implementation from data processing to model training and evaluation for our mini-project in the class Machine Learning & Content Analytics.

As this entire project was developped and implemented in Google colaboratory, we mounted all libraries and our cloud drive here.

For the code, source material and reasoning behind the implementation please refer to our report.

[ ]:
```
import matplotlib.pyplot as plt #used for visualizing useful data
import numpy as np
import os
import tensorflow as tf       #containing all classes used in our architecture
from google.colab import drive #used to mount our cloud drive where we stored␣
 ↪the dataset
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Unzipping data from mounted google drive to the "content" folder in google colab:

[ ]:
```
!unzip -q '/content/drive/MyDrive/MLCA/archive.zip'
```

Loading train and validation datasets using pathlib and Keras' built-in function for rescaling the images to 160x160 and creating batches of 32 at a time.

[ ]:
```
import pathlib

train_dir = pathlib.Path('/content/DATASET/TRAIN')
validation_dir=pathlib.Path('/content/DATASET/TEST')
```

```
BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                            shuffle=True,
                                                           ␣
 ↪batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE)
```

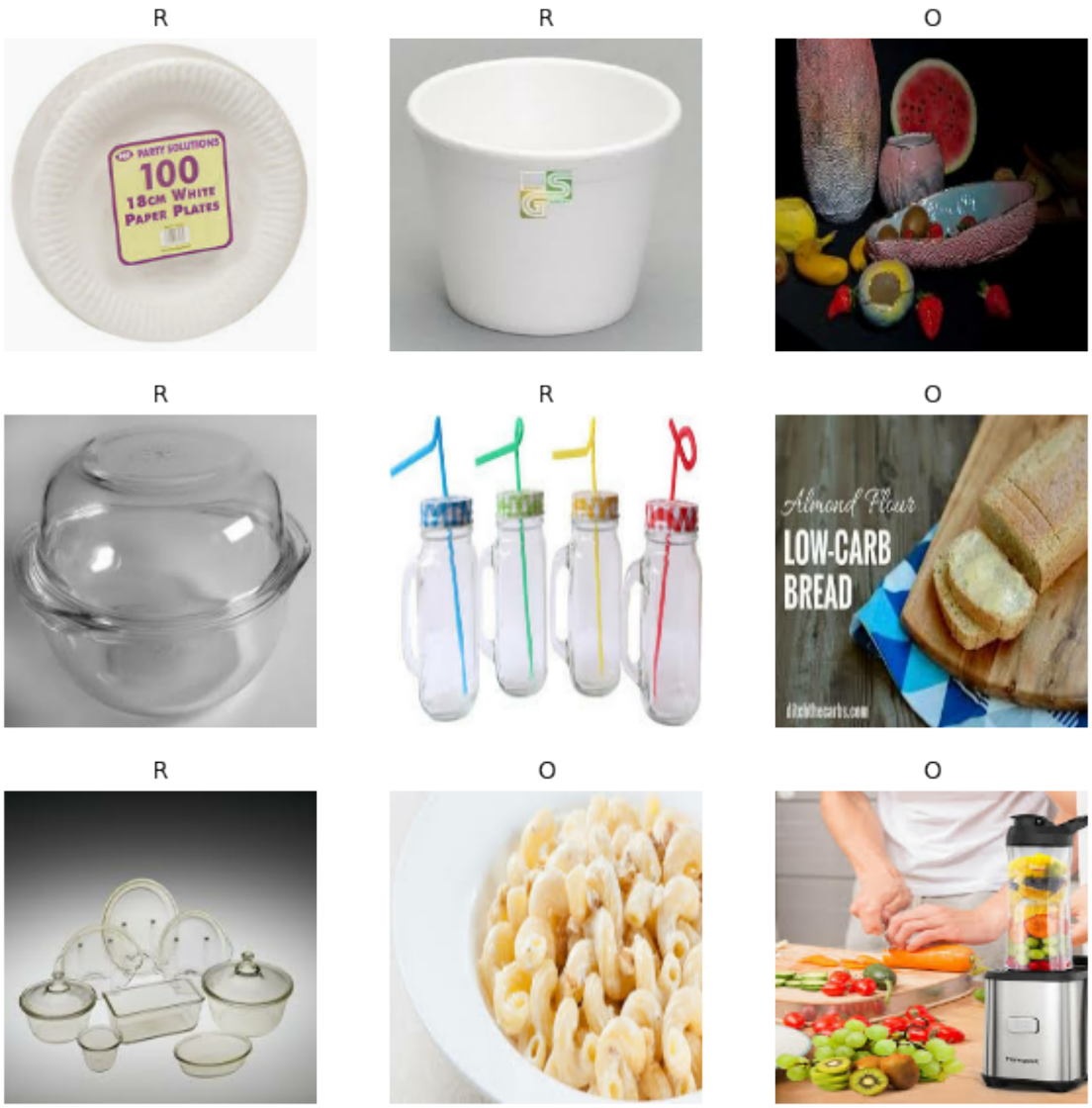
```
Found 22564 files belonging to 2 classes.
```

[ ]:
```
validation_dataset = tf.keras.utils.image_dataset_from_directory(validation_dir,
                                                                 shuffle=True,
                                                                ␣
 ↪batch_size=BATCH_SIZE,
                                                                ␣
 ↪image_size=IMG_SIZE)
```

```
Found 2513 files belonging to 2 classes.
```

Showing the first nine images and labels from the training set to ensure proper loading.

[ ]:
```
class_names = train_dataset.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```



We will use the TRAIN dataset for training (train_dataset), and the TEST dataset for validation during training (validation_ dataset). Since we dont have a dataset for testing after training, we will create our own by splitting the TEST dataset before training. Empirically, a good ratio for this split according to TensorFlow's documentation would be 20-80 for test-validation respectively.

[ ]:
```
val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)
```

[ ]:
```
print('Number of validation batches: %d' % tf.data.experimental.
 ↪cardinality(validation_dataset))
print('Number of test batches: %d' % tf.data.experimental.
 ↪cardinality(test_dataset))
```

```
Number of validation batches: 64
Number of test batches: 15
```

Since we're working in the cloud, we will use TensorFlow's AUTOTUNE and prefetching buffer to reduce I/O blocking.

[ ]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)
```

1.1 Data Augmentation

Our dataset contains a relatively small number of images for training and fine-tuning large convolutional models such as MobileNetV2. That's why we create a part of the pipeline containing 4 different data augmentation techniques: * Random flip horizontally or vertically * Random rotation of the image by 20% * Randomly scaling the contrast up or down by a factor of 0.5 * Randomly zooming at parts of the image by 30%

[ ]:
```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal_and_vertical'),
  tf.keras.layers.RandomRotation(0.2),
  tf.keras.layers.RandomContrast(0.5),
  tf.keras.layers.RandomZoom(0.3),
])
```

1.2 Testing the data augmentation

[ ]:
```
for image, _ in train_dataset.take(1):
  plt.figure(figsize=(10, 10))
  first_image = image[0]
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    augmented_image = data_augmentation(tf.expand_dims(first_image,␣
 ↪0),training=True)
    plt.imshow(augmented_image[0]/255)
    plt.axis('off')
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```

1.3 Rescaling pixels

MobileNetV2's acceptable input for pixels is in the range [-1,1], whereas their current values are in the range [0,255] in the RGB mode. TensorFlow has a builtin function which automatically rescales each pixel turning each image into a 160x160x3 embedded image.

Our goal up until now is to retain the images' colors, textures and core shapes, while also augmenting the dataset and transforming it to a useful input for the model.

[ ]:
```
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
```

1.4 Loading the pre-trained model

The base model used is MobileNet V2 developed at Google. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training

dataset with a wide variety of categories like jackfruit, syringe, bottle etc.

We first load an instance of the model along with its' existing weights and initialize it to accept our input images in a scale of 160x160x3. Since our goal is to create our own custom classifier, we exclude the top layers (classification layers) from the model, so that we will add our own.

[ ]:
```
# Loading MobileNetV2
IMG_SHAPE = IMG_SIZE + (3,) #setting IMG_SHAPE to 160x160x3
base_model = tf.keras.applications.MobileNetV2(input_shape=IMG_SHAPE,
                                               include_top=False,
                                               weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h
5
9412608/9406464 [==============================] - 0s 0us/step
9420800/9406464 [==============================] - 0s 0us/step
```

We use a feature extractor that converts each 160x160x3 image into a 5x5x1280 block of features.

[ ]:
```
image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape) #testing
```

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

2 Feature extraction layers

By freezing the convolutional model and retaining the pretrained weights, we can add a few layers for feature extraction as well as our intended classifier for training.

[ ]:
```
base_model.trainable = False
```

[ ]:
```
#Current model state
base_model.summary()
```

```
Model: "mobilenetv2_1.00_160"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_1 (InputLayer)           [(None, 160, 160, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 80, 80, 32)   864
['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 80, 80, 32)   128         ['Conv1[0][0]']
```

```
 Conv1_relu (ReLU)              (None, 80, 80, 32)   0
['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 80, 80, 32)  288
['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 80, 80, 32)  128
['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 80, 80, 32)  0
['expanded_conv_depthwise_BN[0][0
 ReLU)                                                          ]']

 expanded_conv_project (Conv2D)  (None, 80, 80, 16)  512
['expanded_conv_depthwise_relu[0]
                                                                [0]']

 expanded_conv_project_BN (Batc  (None, 80, 80, 16)  64
['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)        (None, 80, 80, 96)   1536
['expanded_conv_project_BN[0][0]'
                                                               ]

 block_1_expand_BN (BatchNormal  (None, 80, 80, 96)  384
['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)     (None, 80, 80, 96)   0
['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)    (None, 81, 81, 96)   0
['block_1_expand_relu[0][0]']

 block_1_depthwise (DepthwiseCo  (None, 40, 40, 96)  864
['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 40, 40, 96)  384
['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 40, 40, 96)   0
['block_1_depthwise_BN[0][0]']
```

```
 block_1_project (Conv2D)       (None, 40, 40, 24)   2304
['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 40, 40, 24)  96
['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 40, 40, 144)  3456
['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 40, 40, 144)  576
['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 40, 40, 144)  0
['block_2_expand_BN[0][0]']

 block_2_depthwise (DepthwiseCo  (None, 40, 40, 144)  1296
['block_2_expand_relu[0][0]']
 nv2D)

 block_2_depthwise_BN (BatchNor  (None, 40, 40, 144)  576
['block_2_depthwise[0][0]']
 malization)

 block_2_depthwise_relu (ReLU)  (None, 40, 40, 144)  0
['block_2_depthwise_BN[0][0]']

 block_2_project (Conv2D)       (None, 40, 40, 24)   3456
['block_2_depthwise_relu[0][0]']

 block_2_project_BN (BatchNorma  (None, 40, 40, 24)  96
['block_2_project[0][0]']
 lization)

 block_2_add (Add)              (None, 40, 40, 24)   0
['block_1_project_BN[0][0]',
'block_2_project_BN[0][0]']

 block_3_expand (Conv2D)        (None, 40, 40, 144)  3456
['block_2_add[0][0]']

 block_3_expand_BN (BatchNormal  (None, 40, 40, 144)  576
['block_3_expand[0][0]']
 ization)

 block_3_expand_relu (ReLU)     (None, 40, 40, 144)  0
['block_3_expand_BN[0][0]']
```

```
 block_3_pad (ZeroPadding2D)    (None, 41, 41, 144)  0
['block_3_expand_relu[0][0]']

 block_3_depthwise (DepthwiseCo  (None, 20, 20, 144)  1296
['block_3_pad[0][0]']
 nv2D)

 block_3_depthwise_BN (BatchNor  (None, 20, 20, 144)  576
['block_3_depthwise[0][0]']
 malization)

 block_3_depthwise_relu (ReLU)  (None, 20, 20, 144)  0
['block_3_depthwise_BN[0][0]']

 block_3_project (Conv2D)       (None, 20, 20, 32)   4608
['block_3_depthwise_relu[0][0]']

 block_3_project_BN (BatchNorma  (None, 20, 20, 32)  128
['block_3_project[0][0]']
 lization)

 block_4_expand (Conv2D)        (None, 20, 20, 192)  6144
['block_3_project_BN[0][0]']

 block_4_expand_BN (BatchNormal  (None, 20, 20, 192)  768
['block_4_expand[0][0]']
 ization)

 block_4_expand_relu (ReLU)     (None, 20, 20, 192)  0
['block_4_expand_BN[0][0]']

 block_4_depthwise (DepthwiseCo  (None, 20, 20, 192)  1728
['block_4_expand_relu[0][0]']
 nv2D)

 block_4_depthwise_BN (BatchNor  (None, 20, 20, 192)  768
['block_4_depthwise[0][0]']
 malization)

 block_4_depthwise_relu (ReLU)  (None, 20, 20, 192)  0
['block_4_depthwise_BN[0][0]']

 block_4_project (Conv2D)       (None, 20, 20, 32)   6144
['block_4_depthwise_relu[0][0]']

 block_4_project_BN (BatchNorma  (None, 20, 20, 32)  128
['block_4_project[0][0]']
```

```
 lization)

 block_4_add (Add)              (None, 20, 20, 32)   0
['block_3_project_BN[0][0]',
'block_4_project_BN[0][0]']

 block_5_expand (Conv2D)        (None, 20, 20, 192)  6144
['block_4_add[0][0]']

 block_5_expand_BN (BatchNormal  (None, 20, 20, 192)  768
['block_5_expand[0][0]']
 ization)

 block_5_expand_relu (ReLU)     (None, 20, 20, 192)  0
['block_5_expand_BN[0][0]']

 block_5_depthwise (DepthwiseCo  (None, 20, 20, 192)  1728
['block_5_expand_relu[0][0]']
 nv2D)

 block_5_depthwise_BN (BatchNor  (None, 20, 20, 192)  768
['block_5_depthwise[0][0]']
 malization)

 block_5_depthwise_relu (ReLU)  (None, 20, 20, 192)  0
['block_5_depthwise_BN[0][0]']

 block_5_project (Conv2D)       (None, 20, 20, 32)   6144
['block_5_depthwise_relu[0][0]']

 block_5_project_BN (BatchNorma  (None, 20, 20, 32)  128
['block_5_project[0][0]']
 lization)

 block_5_add (Add)              (None, 20, 20, 32)   0
['block_4_add[0][0]',
'block_5_project_BN[0][0]']

 block_6_expand (Conv2D)        (None, 20, 20, 192)  6144
['block_5_add[0][0]']

 block_6_expand_BN (BatchNormal  (None, 20, 20, 192)  768
['block_6_expand[0][0]']
 ization)

 block_6_expand_relu (ReLU)     (None, 20, 20, 192)  0
['block_6_expand_BN[0][0]']
```

```
 block_6_pad (ZeroPadding2D)    (None, 21, 21, 192)  0
['block_6_expand_relu[0][0]']

 block_6_depthwise (DepthwiseCo  (None, 10, 10, 192)  1728
['block_6_pad[0][0]']
 nv2D)

 block_6_depthwise_BN (BatchNor  (None, 10, 10, 192)  768
['block_6_depthwise[0][0]']
 malization)

 block_6_depthwise_relu (ReLU)  (None, 10, 10, 192)  0
['block_6_depthwise_BN[0][0]']

 block_6_project (Conv2D)       (None, 10, 10, 64)   12288
['block_6_depthwise_relu[0][0]']

 block_6_project_BN (BatchNorma  (None, 10, 10, 64)  256
['block_6_project[0][0]']
 lization)

 block_7_expand (Conv2D)        (None, 10, 10, 384)  24576
['block_6_project_BN[0][0]']

 block_7_expand_BN (BatchNormal  (None, 10, 10, 384)  1536
['block_7_expand[0][0]']
 ization)

 block_7_expand_relu (ReLU)     (None, 10, 10, 384)  0
['block_7_expand_BN[0][0]']

 block_7_depthwise (DepthwiseCo  (None, 10, 10, 384)  3456
['block_7_expand_relu[0][0]']
 nv2D)

 block_7_depthwise_BN (BatchNor  (None, 10, 10, 384)  1536
['block_7_depthwise[0][0]']
 malization)

 block_7_depthwise_relu (ReLU)  (None, 10, 10, 384)  0
['block_7_depthwise_BN[0][0]']

 block_7_project (Conv2D)       (None, 10, 10, 64)   24576
['block_7_depthwise_relu[0][0]']

 block_7_project_BN (BatchNorma  (None, 10, 10, 64)  256
['block_7_project[0][0]']
 lization)
```

```
 block_7_add (Add)              (None, 10, 10, 64)   0
['block_6_project_BN[0][0]',
'block_7_project_BN[0][0]']

 block_8_expand (Conv2D)        (None, 10, 10, 384)  24576
['block_7_add[0][0]']

 block_8_expand_BN (BatchNormal  (None, 10, 10, 384)  1536
['block_8_expand[0][0]']
 ization)

 block_8_expand_relu (ReLU)     (None, 10, 10, 384)  0
['block_8_expand_BN[0][0]']

 block_8_depthwise (DepthwiseCo  (None, 10, 10, 384)  3456
['block_8_expand_relu[0][0]']
 nv2D)

 block_8_depthwise_BN (BatchNor  (None, 10, 10, 384)  1536
['block_8_depthwise[0][0]']
 malization)

 block_8_depthwise_relu (ReLU)  (None, 10, 10, 384)  0
['block_8_depthwise_BN[0][0]']

 block_8_project (Conv2D)       (None, 10, 10, 64)   24576
['block_8_depthwise_relu[0][0]']

 block_8_project_BN (BatchNorma  (None, 10, 10, 64)  256
['block_8_project[0][0]']
 lization)

 block_8_add (Add)              (None, 10, 10, 64)   0
['block_7_add[0][0]',
'block_8_project_BN[0][0]']

 block_9_expand (Conv2D)        (None, 10, 10, 384)  24576
['block_8_add[0][0]']

 block_9_expand_BN (BatchNormal  (None, 10, 10, 384)  1536
['block_9_expand[0][0]']
 ization)

 block_9_expand_relu (ReLU)     (None, 10, 10, 384)  0
['block_9_expand_BN[0][0]']

 block_9_depthwise (DepthwiseCo  (None, 10, 10, 384)  3456
```

```
['block_9_expand_relu[0][0]']
 nv2D)

 block_9_depthwise_BN (BatchNor  (None, 10, 10, 384)  1536
['block_9_depthwise[0][0]']
 malization)

 block_9_depthwise_relu (ReLU)  (None, 10, 10, 384)  0
['block_9_depthwise_BN[0][0]']

 block_9_project (Conv2D)       (None, 10, 10, 64)   24576
['block_9_depthwise_relu[0][0]']

 block_9_project_BN (BatchNorma  (None, 10, 10, 64)  256
['block_9_project[0][0]']
 lization)

 block_9_add (Add)              (None, 10, 10, 64)   0
['block_8_add[0][0]',
'block_9_project_BN[0][0]']

 block_10_expand (Conv2D)       (None, 10, 10, 384)  24576
['block_9_add[0][0]']

 block_10_expand_BN (BatchNorma  (None, 10, 10, 384)  1536
['block_10_expand[0][0]']
 lization)

 block_10_expand_relu (ReLU)    (None, 10, 10, 384)  0
['block_10_expand_BN[0][0]']

 block_10_depthwise (DepthwiseC  (None, 10, 10, 384)  3456
['block_10_expand_relu[0][0]']
 onv2D)

 block_10_depthwise_BN (BatchNo  (None, 10, 10, 384)  1536
['block_10_depthwise[0][0]']
 rmalization)

 block_10_depthwise_relu (ReLU)  (None, 10, 10, 384)  0
['block_10_depthwise_BN[0][0]']

 block_10_project (Conv2D)      (None, 10, 10, 96)   36864
['block_10_depthwise_relu[0][0]']

 block_10_project_BN (BatchNorm  (None, 10, 10, 96)  384
['block_10_project[0][0]']
 alization)
```

```
 block_11_expand (Conv2D)       (None, 10, 10, 576)  55296
['block_10_project_BN[0][0]']

 block_11_expand_BN (BatchNorma  (None, 10, 10, 576)  2304
['block_11_expand[0][0]']
 lization)

 block_11_expand_relu (ReLU)    (None, 10, 10, 576)  0
['block_11_expand_BN[0][0]']

 block_11_depthwise (DepthwiseC  (None, 10, 10, 576)  5184
['block_11_expand_relu[0][0]']
 onv2D)

 block_11_depthwise_BN (BatchNo  (None, 10, 10, 576)  2304
['block_11_depthwise[0][0]']
 rmalization)

 block_11_depthwise_relu (ReLU)  (None, 10, 10, 576)  0
['block_11_depthwise_BN[0][0]']

 block_11_project (Conv2D)      (None, 10, 10, 96)   55296
['block_11_depthwise_relu[0][0]']

 block_11_project_BN (BatchNorm  (None, 10, 10, 96)  384
['block_11_project[0][0]']
 alization)

 block_11_add (Add)             (None, 10, 10, 96)   0
['block_10_project_BN[0][0]',
'block_11_project_BN[0][0]']

 block_12_expand (Conv2D)       (None, 10, 10, 576)  55296
['block_11_add[0][0]']

 block_12_expand_BN (BatchNorma  (None, 10, 10, 576)  2304
['block_12_expand[0][0]']
 lization)

 block_12_expand_relu (ReLU)    (None, 10, 10, 576)  0
['block_12_expand_BN[0][0]']

 block_12_depthwise (DepthwiseC  (None, 10, 10, 576)  5184
['block_12_expand_relu[0][0]']
 onv2D)

 block_12_depthwise_BN (BatchNo  (None, 10, 10, 576)  2304
```

```
['block_12_depthwise[0][0]']
 rmalization)

 block_12_depthwise_relu (ReLU)  (None, 10, 10, 576)  0
['block_12_depthwise_BN[0][0]']

 block_12_project (Conv2D)      (None, 10, 10, 96)   55296
['block_12_depthwise_relu[0][0]']

 block_12_project_BN (BatchNorm  (None, 10, 10, 96)  384
['block_12_project[0][0]']
 alization)

 block_12_add (Add)             (None, 10, 10, 96)   0
['block_11_add[0][0]',
'block_12_project_BN[0][0]']

 block_13_expand (Conv2D)       (None, 10, 10, 576)  55296
['block_12_add[0][0]']

 block_13_expand_BN (BatchNorma  (None, 10, 10, 576)  2304
['block_13_expand[0][0]']
 lization)

 block_13_expand_relu (ReLU)    (None, 10, 10, 576)  0
['block_13_expand_BN[0][0]']

 block_13_pad (ZeroPadding2D)   (None, 11, 11, 576)  0
['block_13_expand_relu[0][0]']

 block_13_depthwise (DepthwiseC  (None, 5, 5, 576)   5184
['block_13_pad[0][0]']
 onv2D)

 block_13_depthwise_BN (BatchNo  (None, 5, 5, 576)   2304
['block_13_depthwise[0][0]']
 rmalization)

 block_13_depthwise_relu (ReLU)  (None, 5, 5, 576)   0
['block_13_depthwise_BN[0][0]']

 block_13_project (Conv2D)      (None, 5, 5, 160)    92160
['block_13_depthwise_relu[0][0]']

 block_13_project_BN (BatchNorm  (None, 5, 5, 160)   640
['block_13_project[0][0]']
 alization)
```

```
 block_14_expand (Conv2D)       (None, 5, 5, 960)    153600
['block_13_project_BN[0][0]']

 block_14_expand_BN (BatchNorma  (None, 5, 5, 960)   3840
['block_14_expand[0][0]']
 lization)

 block_14_expand_relu (ReLU)    (None, 5, 5, 960)    0
['block_14_expand_BN[0][0]']

 block_14_depthwise (DepthwiseC  (None, 5, 5, 960)   8640
['block_14_expand_relu[0][0]']
 onv2D)

 block_14_depthwise_BN (BatchNo  (None, 5, 5, 960)   3840
['block_14_depthwise[0][0]']
 rmalization)

 block_14_depthwise_relu (ReLU)  (None, 5, 5, 960)   0
['block_14_depthwise_BN[0][0]']

 block_14_project (Conv2D)      (None, 5, 5, 160)    153600
['block_14_depthwise_relu[0][0]']

 block_14_project_BN (BatchNorm  (None, 5, 5, 160)   640
['block_14_project[0][0]']
 alization)

 block_14_add (Add)             (None, 5, 5, 160)    0
['block_13_project_BN[0][0]',
'block_14_project_BN[0][0]']

 block_15_expand (Conv2D)       (None, 5, 5, 960)    153600
['block_14_add[0][0]']

 block_15_expand_BN (BatchNorma  (None, 5, 5, 960)   3840
['block_15_expand[0][0]']
 lization)

 block_15_expand_relu (ReLU)    (None, 5, 5, 960)    0
['block_15_expand_BN[0][0]']

 block_15_depthwise (DepthwiseC  (None, 5, 5, 960)   8640
['block_15_expand_relu[0][0]']
 onv2D)

 block_15_depthwise_BN (BatchNo  (None, 5, 5, 960)   3840
['block_15_depthwise[0][0]']
```

```
 rmalization)

 block_15_depthwise_relu (ReLU)  (None, 5, 5, 960)   0
['block_15_depthwise_BN[0][0]']

 block_15_project (Conv2D)      (None, 5, 5, 160)    153600
['block_15_depthwise_relu[0][0]']

 block_15_project_BN (BatchNorm  (None, 5, 5, 160)   640
['block_15_project[0][0]']
 alization)

 block_15_add (Add)             (None, 5, 5, 160)    0
['block_14_add[0][0]',
'block_15_project_BN[0][0]']

 block_16_expand (Conv2D)       (None, 5, 5, 960)    153600
['block_15_add[0][0]']

 block_16_expand_BN (BatchNorma  (None, 5, 5, 960)   3840
['block_16_expand[0][0]']
 lization)

 block_16_expand_relu (ReLU)    (None, 5, 5, 960)    0
['block_16_expand_BN[0][0]']

 block_16_depthwise (DepthwiseC  (None, 5, 5, 960)   8640
['block_16_expand_relu[0][0]']
 onv2D)

 block_16_depthwise_BN (BatchNo  (None, 5, 5, 960)   3840
['block_16_depthwise[0][0]']
 rmalization)

 block_16_depthwise_relu (ReLU)  (None, 5, 5, 960)   0
['block_16_depthwise_BN[0][0]']

 block_16_project (Conv2D)      (None, 5, 5, 320)    307200
['block_16_depthwise_relu[0][0]']

 block_16_project_BN (BatchNorm  (None, 5, 5, 320)   1280
['block_16_project[0][0]']
 alization)

 Conv_1 (Conv2D)                (None, 5, 5, 1280)   409600
['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalization)  (None, 5, 5, 1280)  5120
```

```
['Conv_1[0][0]']

 out_relu (ReLU)                (None, 5, 5, 1280)   0
['Conv_1_bn[0][0]']

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

We will add a GlobalAveragePooling layer for expansion and use that as the input for classifications/predictions. This will convert the current output from 5x5 to a vector of 1280 elements for each image.

[ ]:
```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)
```

```
(32, 1280)
```

Next we use a tf.keras.layers.Dense layer to convert the above output into a single prediction for each image. The prediction we want will be binary, using logit where positive values will be classified as class R and negative values will be classified as class O.

[ ]:
```
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)
```

```
(32, 1)
```

2.1 Compiling the completed model

We create our pipeline using Keras' functional approach and adding all of the previously mentioned steps. We also add a steady dropout rate of 0.2 after the global average layer and right before the prediction layer to reduce overfitting risks.

Before compiling we use the Adam optimizer for the base model and a learning rate of 0.0001. Finally, our loss calculation function, as is generally used with classifiers will be binary cross-entropy, which is the most robust evaluator in our case.

[ ]:
```
inputs = tf.keras.Input(shape=(160, 160, 3)) #inputing images and rescaling␣
 ↪accordingly
x = data_augmentation(inputs)        # data augmentation
x = preprocess_input(x)              # automatic preprocessing for the color␣
 ↪channels
x = base_model(x, training=False)    # inputing the model and freezing the weights
x = global_average_layer(x)          # average pooling expansion layer
```

```
x = tf.keras.layers.Dropout(0.2)(x) # dropout rate for overfitting counter
outputs = prediction_layer(x)       # logit prediction
model = tf.keras.Model(inputs, outputs)
```

[ ]:
```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.
 ↪Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

[ ]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambda  (None, 160, 160, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 160, 160, 3)      0
 a)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

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

The 2.5 million parameters in MobileNet are frozen, but there are 1.2 thousand trainable parameters in the Dense layer. We will train only those.

[ ]:
```
len(model.trainable_variables)
```

[ ]: 2

3 Training the model

After many iterations, we found that a good number of epochs for training is between 15 and 20, so we will keep 20 and save the weights using callbacks to prevent overfitting.

For our own evaluations, we first try to predict the validation dataset without any training to see how the basic model performs.

[ ]:
```
initial_epochs = 20

loss0, accuracy0 = model.evaluate(validation_dataset)
```

```
64/64 [==============================] - 34s 496ms/step - loss: 1.0121 -
accuracy: 0.4063
```

[ ]:
```
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
initial loss: 1.01
initial accuracy: 0.41
```

[ ]:
```
checkpoint_path = "/content/drive/MyDrive/MLCA/checkpoints/
 ↪feature_extraction_cp_1"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1,
                                                 save_best_only=True)
history = model.fit(train_dataset,
                    epochs=initial_epochs,
                    validation_data=validation_dataset,
                    callbacks=[cp_callback]
                    )
```

```
Epoch 1/20
706/706 [==============================] - ETA: 0s - loss: 0.5195 - accuracy:
0.7258
Epoch 1: val_loss improved from inf to 0.36677, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 494s 693ms/step - loss: 0.5195 -
accuracy: 0.7258 - val_loss: 0.3668 - val_accuracy: 0.7787
Epoch 2/20
706/706 [==============================] - ETA: 0s - loss: 0.3269 - accuracy:
0.8563
Epoch 2: val_loss improved from 0.36677 to 0.30594, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 502s 710ms/step - loss: 0.3269 -
accuracy: 0.8563 - val_loss: 0.3059 - val_accuracy: 0.8151
```

```
Epoch 3/20
706/706 [==============================] - ETA: 0s - loss: 0.2890 - accuracy:
0.8778
Epoch 3: val_loss improved from 0.30594 to 0.28773, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 496s 702ms/step - loss: 0.2890 -
accuracy: 0.8778 - val_loss: 0.2877 - val_accuracy: 0.8288
Epoch 4/20
706/706 [==============================] - ETA: 0s - loss: 0.2714 - accuracy:
0.8861
Epoch 4: val_loss improved from 0.28773 to 0.25757, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 486s 689ms/step - loss: 0.2714 -
accuracy: 0.8861 - val_loss: 0.2576 - val_accuracy: 0.8500
Epoch 5/20
706/706 [==============================] - ETA: 0s - loss: 0.2601 - accuracy:
0.8930
Epoch 5: val_loss did not improve from 0.25757
706/706 [==============================] - 489s 692ms/step - loss: 0.2601 -
accuracy: 0.8930 - val_loss: 0.2667 - val_accuracy: 0.8431
Epoch 6/20
706/706 [==============================] - ETA: 0s - loss: 0.2545 - accuracy:
0.8967
Epoch 6: val_loss improved from 0.25757 to 0.24069, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 487s 689ms/step - loss: 0.2545 -
accuracy: 0.8967 - val_loss: 0.2407 - val_accuracy: 0.8628
Epoch 7/20
706/706 [==============================] - ETA: 0s - loss: 0.2479 - accuracy:
0.8989
Epoch 7: val_loss improved from 0.24069 to 0.23733, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 490s 694ms/step - loss: 0.2479 -
accuracy: 0.8989 - val_loss: 0.2373 - val_accuracy: 0.8662
Epoch 8/20
706/706 [==============================] - ETA: 0s - loss: 0.2455 - accuracy:
0.8982
Epoch 8: val_loss improved from 0.23733 to 0.23302, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 490s 694ms/step - loss: 0.2455 -
accuracy: 0.8982 - val_loss: 0.2330 - val_accuracy: 0.8706
Epoch 9/20
706/706 [==============================] - ETA: 0s - loss: 0.2423 - accuracy:
0.9029
Epoch 9: val_loss improved from 0.23302 to 0.21961, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 494s 700ms/step - loss: 0.2423 -
accuracy: 0.9029 - val_loss: 0.2196 - val_accuracy: 0.8775
```

```
Epoch 10/20
706/706 [==============================] - ETA: 0s - loss: 0.2400 - accuracy:
0.9049
Epoch 10: val_loss did not improve from 0.21961
706/706 [==============================] - 495s 700ms/step - loss: 0.2400 -
accuracy: 0.9049 - val_loss: 0.2275 - val_accuracy: 0.8736
Epoch 11/20
706/706 [==============================] - ETA: 0s - loss: 0.2350 - accuracy:
0.9039
Epoch 11: val_loss improved from 0.21961 to 0.21835, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 501s 709ms/step - loss: 0.2350 -
accuracy: 0.9039 - val_loss: 0.2184 - val_accuracy: 0.8800
Epoch 12/20
706/706 [==============================] - ETA: 0s - loss: 0.2350 - accuracy:
0.9048
Epoch 12: val_loss did not improve from 0.21835
706/706 [==============================] - 497s 703ms/step - loss: 0.2350 -
accuracy: 0.9048 - val_loss: 0.2197 - val_accuracy: 0.8800
Epoch 13/20
706/706 [==============================] - ETA: 0s - loss: 0.2276 - accuracy:
0.9091
Epoch 13: val_loss improved from 0.21835 to 0.21744, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 497s 703ms/step - loss: 0.2276 -
accuracy: 0.9091 - val_loss: 0.2174 - val_accuracy: 0.8829
Epoch 14/20
706/706 [==============================] - ETA: 0s - loss: 0.2297 - accuracy:
0.9065
Epoch 14: val_loss improved from 0.21744 to 0.21581, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 501s 709ms/step - loss: 0.2297 -
accuracy: 0.9065 - val_loss: 0.2158 - val_accuracy: 0.8824
Epoch 15/20
706/706 [==============================] - ETA: 0s - loss: 0.2295 - accuracy:
0.9066
Epoch 15: val_loss improved from 0.21581 to 0.21286, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 504s 714ms/step - loss: 0.2295 -
accuracy: 0.9066 - val_loss: 0.2129 - val_accuracy: 0.8849
Epoch 16/20
706/706 [==============================] - ETA: 0s - loss: 0.2284 - accuracy:
0.9075
Epoch 16: val_loss improved from 0.21286 to 0.20466, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 502s 711ms/step - loss: 0.2284 -
accuracy: 0.9075 - val_loss: 0.2047 - val_accuracy: 0.8923
Epoch 17/20
```

```
706/706 [==============================] - ETA: 0s - loss: 0.2270 - accuracy:
0.9085
Epoch 17: val_loss improved from 0.20466 to 0.20253, saving model to
/content/drive/MyDrive/MLCA/checkpoints/feature_extraction_cp_1
706/706 [==============================] - 507s 718ms/step - loss: 0.2270 -
accuracy: 0.9085 - val_loss: 0.2025 - val_accuracy: 0.8967
Epoch 18/20
706/706 [==============================] - ETA: 0s - loss: 0.2264 - accuracy:
0.9089
Epoch 18: val_loss did not improve from 0.20253
706/706 [==============================] - 499s 706ms/step - loss: 0.2264 -
accuracy: 0.9089 - val_loss: 0.2042 - val_accuracy: 0.8947
Epoch 19/20
706/706 [==============================] - ETA: 0s - loss: 0.2258 - accuracy:
0.9072
Epoch 19: val_loss did not improve from 0.20253
706/706 [==============================] - 497s 703ms/step - loss: 0.2258 -
accuracy: 0.9072 - val_loss: 0.2030 - val_accuracy: 0.8908
Epoch 20/20
706/706 [==============================] - ETA: 0s - loss: 0.2207 - accuracy:
0.9097
Epoch 20: val_loss did not improve from 0.20253
706/706 [==============================] - 500s 707ms/step - loss: 0.2207 -
accuracy: 0.9097 - val_loss: 0.2061 - val_accuracy: 0.8923
```

4 Saving and reloading

[ ]:
```
# saving the weights only
# model.save_weights('/content/drive/MyDrive/MLCA/checkpoints/my_checkpoint')
```

[ ]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f23cde80190>
```

[ ]:
```
# saving the entire model
# model.save('/content/drive/MyDrive/MLCA/models/model2')
```

4.1 Reloading

[ ]:
```
# restoring the weights after creating a model with the exact same architecture.␣
↪Doesn't work properly if the model isn't identical.
model.load_weights('/content/drive/MyDrive/MLCA/checkpoints/finetuning_cp_1')
# model.load_weights('/content/drive/MyDrive/MLCA/checkpoints/
↪feature_extraction_cp_3')
```

[ ]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7ff0e7effd90>
```

[ ]:
```
# reloading a fresh model from the previously saved one, along with the weights
# new_model = tf.keras.models.load_model('/content/drive/MyDrive/MLCA/models/
 ↪model1')

# # Check its architecture
# new_model.summary()
```

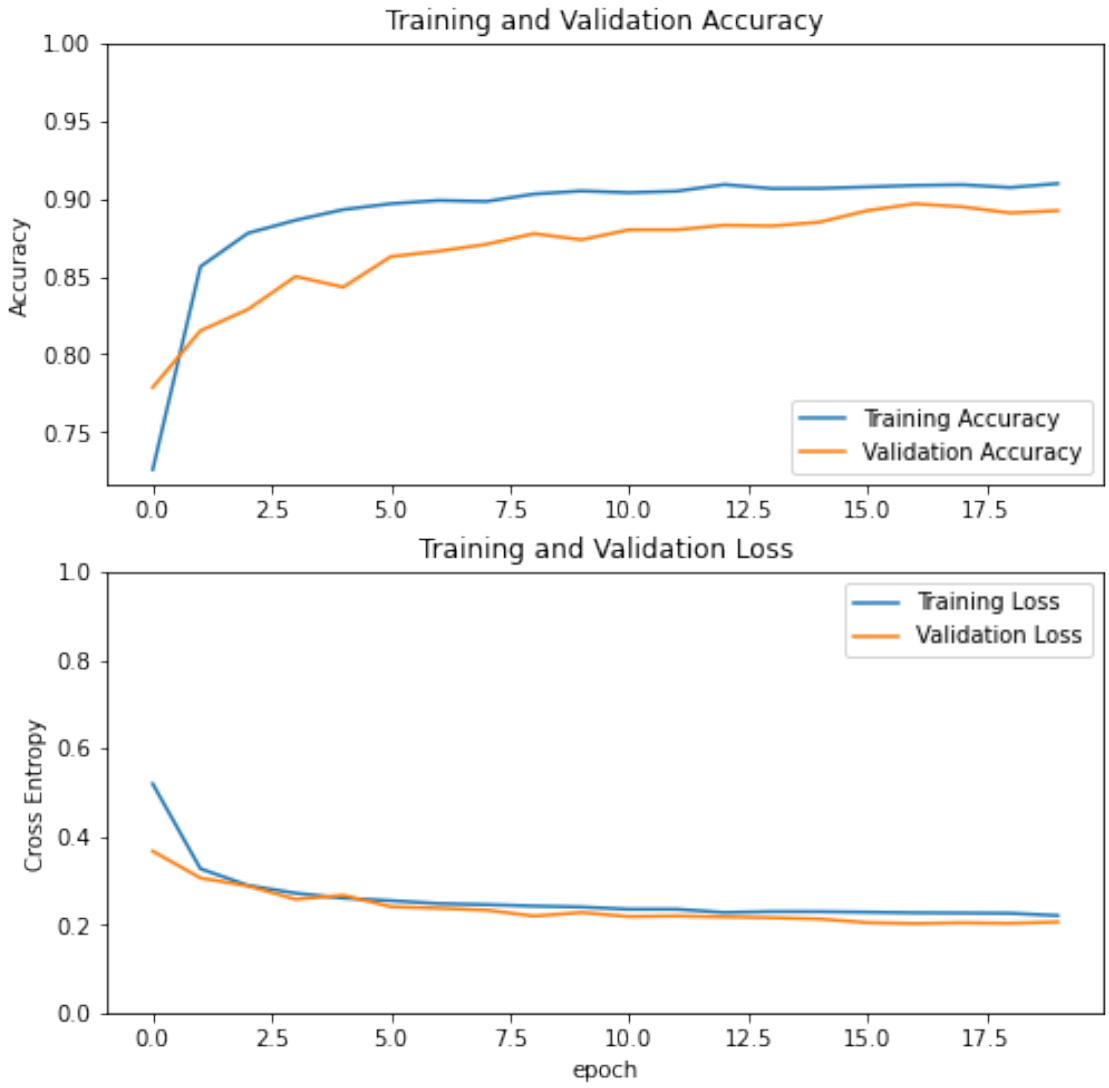
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambda  (None, 160, 160, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 160, 160, 3)      0
 a)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

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

5 Learning curves

We will use the next blocks to visualize our model's evolution in terms of accuracy and loss, to find insights on how to fine tune it.

[ ]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
plt.figure(figsize=(8, 8)) #Plotting training and validation accuracy evolution
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2) #Plotting training and validation loss evolution
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```



6 Fine Tuning

Our model has reached satisfactory thresholds in both evaluations, although there is potential for improvement, which we will build upon. For starters, we will unfreeze 30% of the layers of the pre-trained model and finetune them to our needs. The model is lightweight enough that we can train more than 1.5M parameters without significant cost or stronger equipment than the one we have at our disposal.

[ ]:
```
base_model.trainable = True
```

[ ]:
```
# finding the number of layers in the base model
print("Number of layers in the base model: ", len(base_model.layers))
```

```
Number of layers in the base model:  154
```

[ ]:
```
# Fine-tune from this layer onwards
fine_tune_at = 100

# Freezing only the first 100 layers
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

6.1 Recompiling the model

We will now recompile the model before retraining it. This time, we will use RMSprop (which ignores momentum and optimizes at a lower rate than Adam) for optimization and a much slower learning rate of 0.00001 to prevent overfitting.

The number of trainable parameters this time will be over 1.8M.

[ ]:
```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
              optimizer = tf.keras.optimizers.
 ↪RMSprop(learning_rate=base_learning_rate/10),
              metrics=['accuracy'])
```

[ ]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 160, 160, 3)]     0

 sequential (Sequential)     (None, 160, 160, 3)       0

 tf.math.truediv (TFOpLambda  (None, 160, 160, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 160, 160, 3)      0
 a)

 mobilenetv2_1.00_160 (Funct  (None, 5, 5, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 1)                 1281

=================================================================
Total params: 2,259,265
Trainable params: 1,862,721
```

```
Non-trainable params: 396,544
_________________________________________________________________
```

[ ]:
```
len(model.trainable_variables)
```

[ ]:
```
56
```

6.2 Re-training the model

Through re-training we expect an improvement of at least a couple of percentage points in accuracy and a significant reduction of loss value.

Note: we ran training several times, the weights of the best model are saved to finetuning_cp_1

[ ]:
```
checkpoint_path = "/content/drive/MyDrive/MLCA/checkpoints/finetuning_cp_3"
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create a callback that saves the model's weights
cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                 save_weights_only=True,
                                                 verbose=1,
                                                 save_best_only=True)
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_dataset,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=validation_dataset,
                         callbacks=[cp_callback])
```

```
Epoch 20/30
706/706 [==============================] - ETA: 0s - loss: 0.2143 - accuracy:
0.9159
Epoch 20: val_loss improved from inf to 0.21431, saving model to
/content/drive/MyDrive/MLCA/checkpoints/finetuning_cp_3
706/706 [==============================] - 800s 1s/step - loss: 0.2143 -
accuracy: 0.9159 - val_loss: 0.2143 - val_accuracy: 0.9006
Epoch 21/30
706/706 [==============================] - ETA: 0s - loss: 0.1894 - accuracy:
0.9258
Epoch 21: val_loss did not improve from 0.21431
706/706 [==============================] - 784s 1s/step - loss: 0.1894 -
accuracy: 0.9258 - val_loss: 0.2234 - val_accuracy: 0.8933
Epoch 22/30
706/706 [==============================] - ETA: 0s - loss: 0.1758 - accuracy:
0.9283
Epoch 22: val_loss improved from 0.21431 to 0.16434, saving model to
/content/drive/MyDrive/MLCA/checkpoints/finetuning_cp_3
```

```
706/706 [==============================] - 796s 1s/step - loss: 0.1758 -
accuracy: 0.9283 - val_loss: 0.1643 - val_accuracy: 0.9233
Epoch 23/30
706/706 [==============================] - ETA: 0s - loss: 0.1674 - accuracy:
0.9328
Epoch 23: val_loss improved from 0.16434 to 0.14795, saving model to
/content/drive/MyDrive/MLCA/checkpoints/finetuning_cp_3
706/706 [==============================] - 804s 1s/step - loss: 0.1674 -
accuracy: 0.9328 - val_loss: 0.1480 - val_accuracy: 0.9415
Epoch 24/30
706/706 [==============================] - ETA: 0s - loss: 0.1595 - accuracy:
0.9373
Epoch 24: val_loss did not improve from 0.14795
706/706 [==============================] - 802s 1s/step - loss: 0.1595 -
accuracy: 0.9373 - val_loss: 0.1799 - val_accuracy: 0.9262
Epoch 25/30
706/706 [==============================] - ETA: 0s - loss: 0.1503 - accuracy:
0.9427
Epoch 25: val_loss did not improve from 0.14795
706/706 [==============================] - 799s 1s/step - loss: 0.1503 -
accuracy: 0.9427 - val_loss: 0.1980 - val_accuracy: 0.9149
Epoch 26/30
706/706 [==============================] - ETA: 0s - loss: 0.1472 - accuracy:
0.9410
Epoch 26: val_loss did not improve from 0.14795
706/706 [==============================] - 806s 1s/step - loss: 0.1472 -
accuracy: 0.9410 - val_loss: 0.1643 - val_accuracy: 0.9385
Epoch 27/30
706/706 [==============================] - ETA: 0s - loss: 0.1403 - accuracy:
0.9452
Epoch 27: val_loss did not improve from 0.14795
706/706 [==============================] - 808s 1s/step - loss: 0.1403 -
accuracy: 0.9452 - val_loss: 0.1629 - val_accuracy: 0.9365
Epoch 28/30
706/706 [==============================] - ETA: 0s - loss: 0.1372 - accuracy:
0.9469
Epoch 28: val_loss did not improve from 0.14795
706/706 [==============================] - 805s 1s/step - loss: 0.1372 -
accuracy: 0.9469 - val_loss: 0.1835 - val_accuracy: 0.9233
Epoch 29/30
706/706 [==============================] - ETA: 0s - loss: 0.1325 - accuracy:
0.9471
Epoch 29: val_loss did not improve from 0.14795
706/706 [==============================] - 802s 1s/step - loss: 0.1325 -
accuracy: 0.9471 - val_loss: 0.1841 - val_accuracy: 0.9223
Epoch 30/30
706/706 [==============================] - ETA: 0s - loss: 0.1289 - accuracy:
0.9485
```

```
Epoch 30: val_loss did not improve from 0.14795
706/706 [==============================] - 798s 1s/step - loss: 0.1289 -
accuracy: 0.9485 - val_loss: 0.2472 - val_accuracy: 0.8982
```

7 Evaluating the model

We will first visualize the model's evolution as we did before, to confirm which weights are the better ones. We notice a drastic improvement after finetuning, with an increase in accuracy by 5 percentage points and a decrease in loss by more than 50%. This exceeds expectations.

[ ]:
```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

loss += history_fine.history['loss']
val_loss += history_fine.history['val_loss']
```

[ ]:
```
plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0.8, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.ylim([0, 1.0])
plt.plot([initial_epochs-1,initial_epochs-1],
         plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

[ ]:
```
loss, accuracy = model.evaluate(test_dataset)
print('Test accuracy :', accuracy)
```

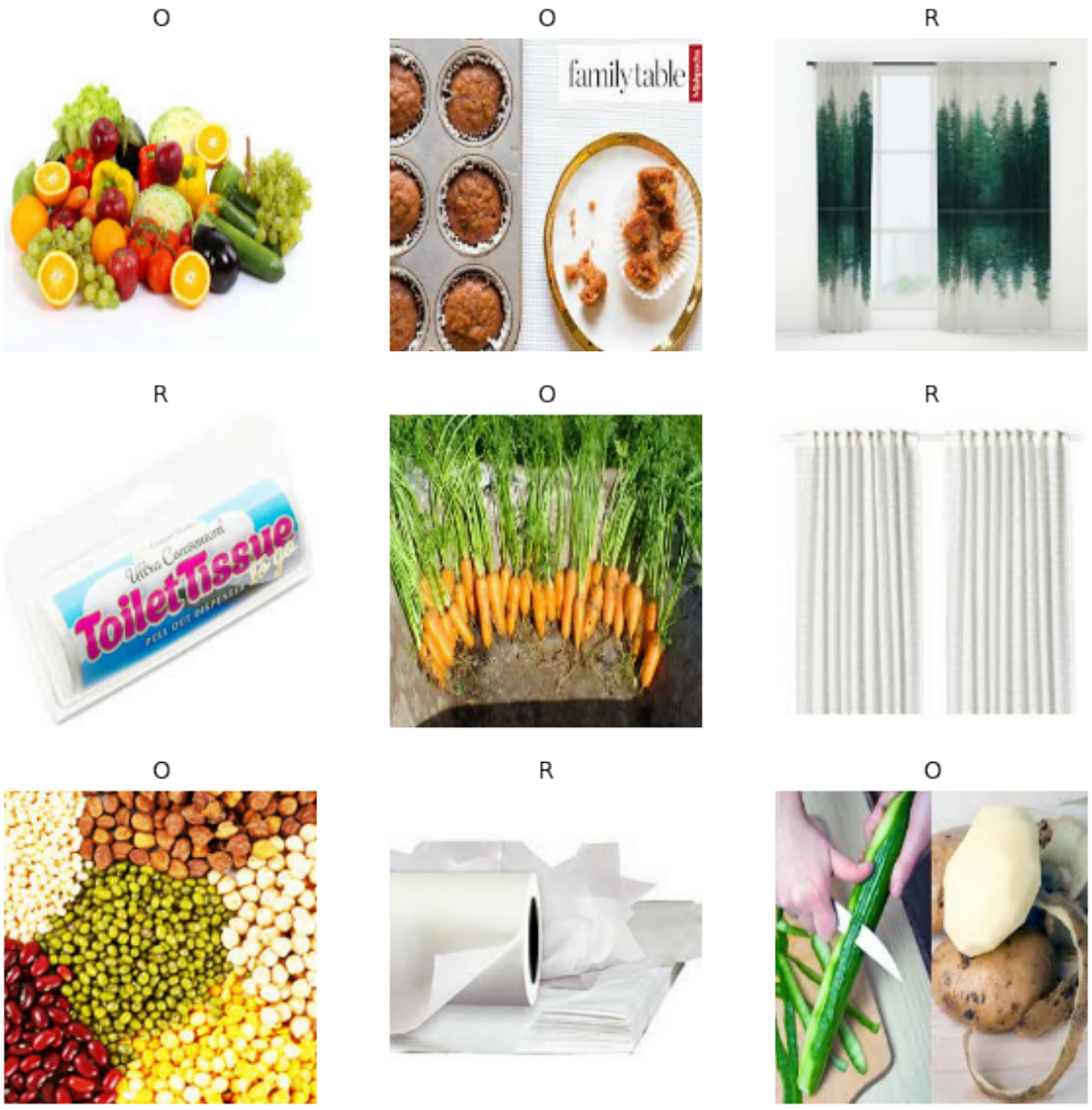
```
15/15 [==============================] - 13s 626ms/step - loss: 0.1390 -
accuracy: 0.9521
Test accuracy : 0.9520833492279053
```

Example predictions

[ ]:
```
# Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
```

```
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

```
Predictions:
 [0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0]
Labels:
 [0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 0]
```



7.1 Single image predictions

[ ]:
```
import cv2

def predict_img(dir,name,ext='.png'):
    img=cv2.imread(dir+name+ext)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis("off")
    img = cv2.resize(img, (160, 160))
    img = tf.reshape(img, [-1,160,160,3])
    result = tf.where(model.predict(img) < 0.5, 0, 1)
    if result == 0: plt.title('Organic')
```

```
    elif result ==1: plt.title('Recyclable')
```

[ ]:
```
dir='/content/drive/MyDrive/MLCA/test images/'
name='Screenshot_3'
predict_img(dir,name)
```

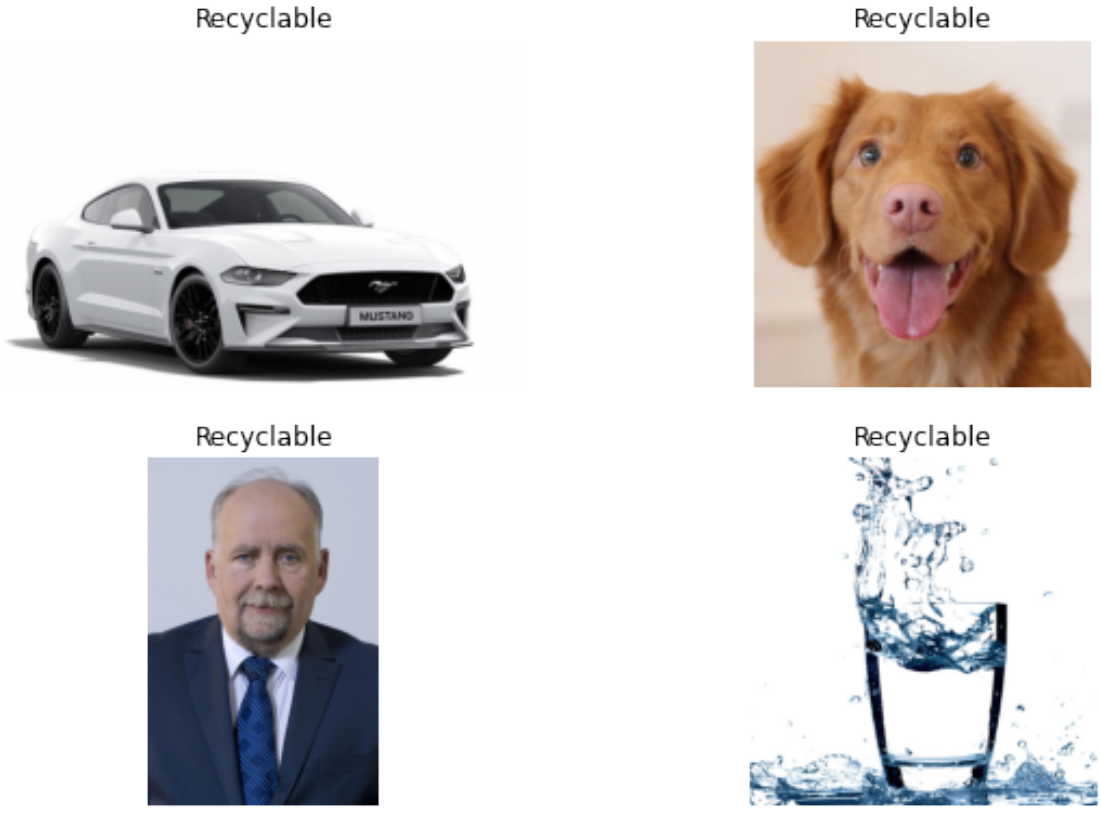
7.2 Crash testing for biases and out-of-sample capabilities

As we try to identify the model's shortcomings and the features it has incorporated, we feed it with specific images with the potential to confuse it. We discuss our hypotheses and conclusions on the matter in our report.

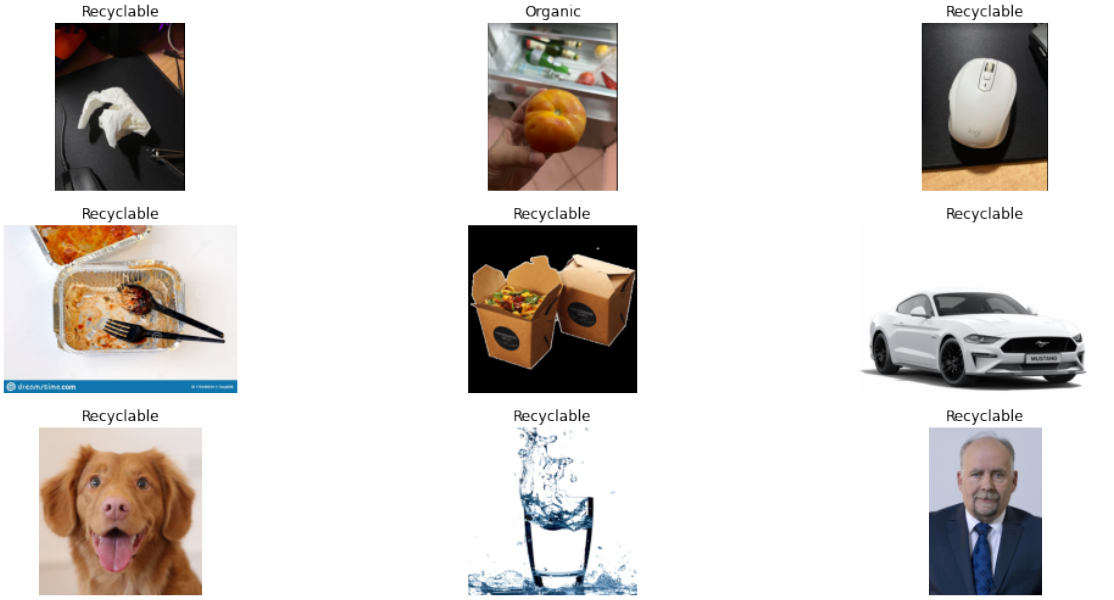
[ ]:
```
import math

def predict_dir(dir,n=3):
  '''
  Predicts a folder of images only, given its directory (dir).
  The user can also set the output's number of photos in the vertical dimension␣
 ↪of the
  grid (default n=3)
  '''
  imlist=os.listdir(dir)
  num=len(imlist)

  plt.figure(figsize=(num*2,num*2))
```

```
  for i in range(num):
    ax = plt.subplot(n,math.ceil(num/n) , i + 1)
    plt.subplots_adjust(top = 0.99, bottom=0.5)
    predict_img(dir,imlist[i],ext='')
```

[ ]:
```
predict_dir(dir,2)
```



[ ]:
```
predict_dir(dir,3)
```



7.3 This marks the end of this notebook.