Assignment: Image Classification

- Abigail Smith (ARS190011), Eric Shields (ESS190007)
- CS 4375.004
- Dr. Mazidi
- 04/22/2023

Setting Up Program

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sb
import numpy as np
import PIL
import tensorflow as tf
import os

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

```
## If training using google colab. Otherwise using pandas
from google.colab import drive
drive.mount('/content/drive')
file = "/content/drive/MyDrive/Colab_Notebooks/tom_and_jerry"
```

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

```
## If training from a computer
file = "tom_and_jerry"
```

Describing the Dataset

The dataset used for this assignment is a Tom and Jerry image classification dataset from Kaggle (https://www.kaggle.com/datasets/balabaskar/tom-and-jerry-image-classification). The dataset consists of two types of images -- either images of Tom or images of Jerry. While the original dataset provided a subset of images with both of these cartoon characters in the same image, we decided for this project to use the subsets with only one of the characters in an image. Both the sequential and CNN models should be able to predict if an image has Tom or has Jerry in it.

Dividing Dataset into Test/Train

```
batchSize = 32 # arbitrary
epochs = 10 # how many passes forward and backwards
imageSize = (200, 100)

train, test  = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/Colab_Notebooks/tom_and_jerry",
    validation_split = .2,
    subset = "both",
    seed = 1234,
    image_size = imageSize,
    label_mode = "binary",
    color_mode = "rgb",
    batch_size = batchSize
)
```

```
    Found 3170 files belonging to 2 classes.
    Using 2536 files for training.
    Using 634 files for validation.
```

Exploration

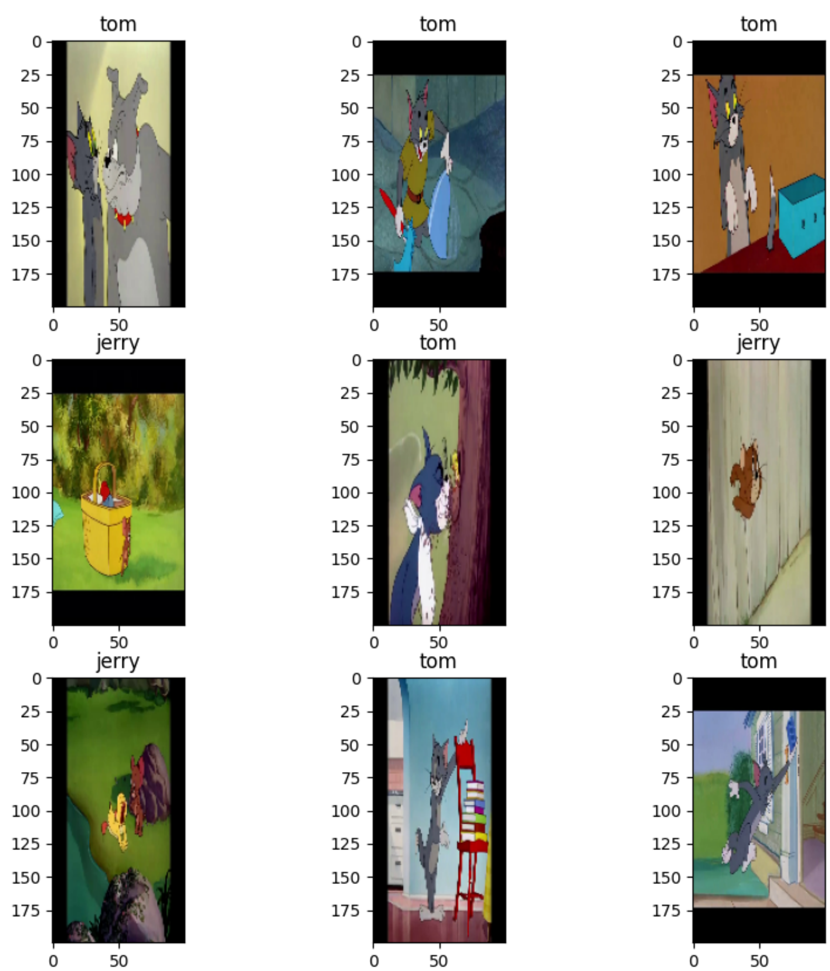
```
## Prints the first image batch
for image_batch, labels_batch in train:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 200, 100, 3)
(32, 1)
```

```
## Prints the class names that will be used for training
class_names = train.class_names
print(class_names)
```

```
['jerry', 'tom']
```

```
## Prints pictures with all of the various classes
plt.figure(figsize = (10, 10))
for images, labels in train.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[int(labels[i])])
```

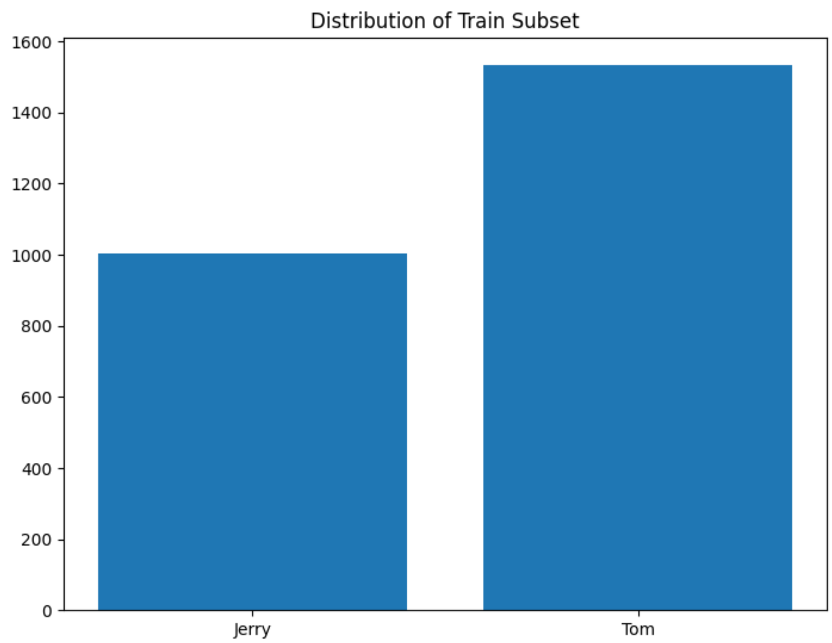



```
## Finding distribution in train dataset
labels = []
train_unbatched = tuple(train.unbatch())
for (image, label) in train_unbatched:
  labels.append(label.numpy())
labels = pd.Series(labels)
```

```
count = labels.value_counts().sort_index()
count.index = class_names

print(count)
```

```
jerry    1003
tom      1533
dtype: int64
```

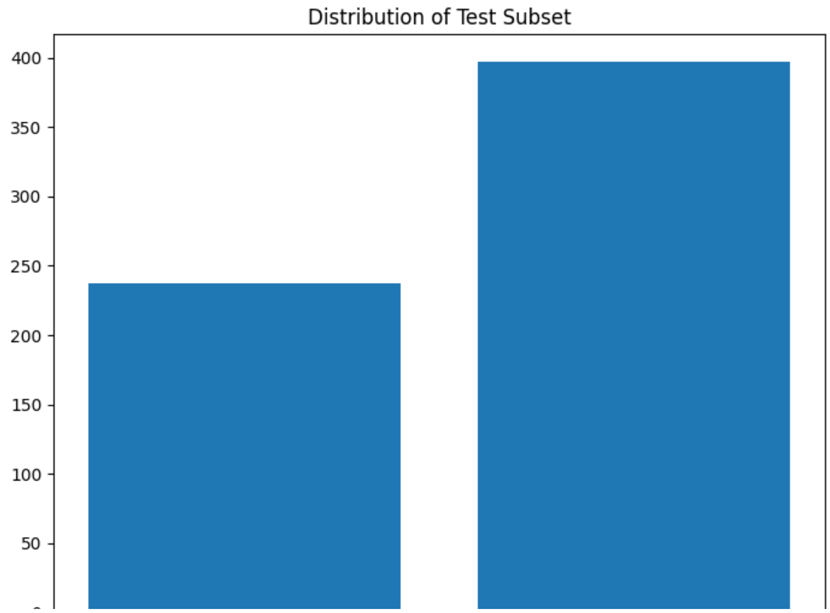
```
figure = plt.figure()
axes = figure.add_axes([1, 1, 1, 1])
x_axis = ['Jerry', 'Tom']
y_axis = [count[0], count[1]]
axes.set_title("Distribution of Train Subset")
axes.bar(x_axis, y_axis)
plt.show()
```




```
## Finding distribution in test dataset
labels = []
test_unbatched = tuple(test.unbatch())
for (image, label) in test_unbatched:
  labels.append(label.numpy())
labels = pd.Series(labels)
count = labels.value_counts().sort_index()
count.index = class_names

print(count)
```

```
jerry    237
tom      397
dtype: int64
```

```
figure = plt.figure()
axes = figure.add_axes([1, 1, 1, 1])
x_axis = ['Jerry', 'Tom']
y_axis = [count[0], count[1]]
axes.set_title("Distribution of Test Subset")
axes.bar(x_axis, y_axis)
plt.show()
```



Sequential Model

Creating Model

```
#Getting number of classes
num_classes = len(class_names)

## Model creation but without the convalution level for a plain sequential model
model = Sequential([
  layers.Flatten(input_shape = (200, 100, 3)),
  layers.Dense(512, activation='relu'),
  layers.Dropout(0.2),
  layers.Dense(256, activation = 'relu'),
  layers.Dropout(0.2),
  layers.Dense(128, activation = 'relu'),
  layers.Dropout(0.2),
  layers.Dense(2, activation = 'softmax')
])
```

Showing Summary

```
## Prints the summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 60000)             0

 dense (Dense)               (None, 512)               30720512

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dropout_2 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 2)                 258

=================================================================
Total params: 30,884,994
Trainable params: 30,884,994
Non-trainable params: 0
_________________________________________________________________
```

Compiling Model

```
## Compiles the sequantial model
model.compile(optimizer='rmsprop',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```

Fitting Model

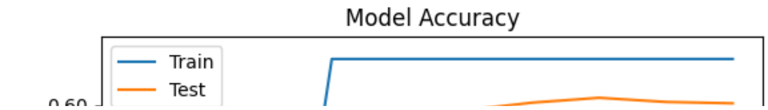
```
## Fits the model to the data
epochs = 10
history = model.fit(
  train,
  validation_data=test,
  epochs=epochs,
  batch_size = batchSize,
  verbose = 1
)
```

```
Epoch 1/10
80/80 [==============================] - 56s 673ms/step - loss: 1530.1299 - accuracy: 0.5426 - val_loss: 299.1588 - val_accuracy: 0.3738
Epoch 2/10
80/80 [==============================] - 53s 639ms/step - loss: 81.9818 - accuracy: 0.5229 - val_loss: 1.0233 - val_accuracy: 0.3722
Epoch 3/10
80/80 [==============================] - 53s 643ms/step - loss: 6.6325 - accuracy: 0.4803 - val_loss: 0.6935 - val_accuracy: 0.3738
Epoch 4/10
80/80 [==============================] - 54s 661ms/step - loss: 1.6129 - accuracy: 0.5793 - val_loss: 0.6815 - val_accuracy: 0.6262
Epoch 5/10
80/80 [==============================] - 50s 616ms/step - loss: 1.3808 - accuracy: 0.5946 - val_loss: 0.6748 - val_accuracy: 0.6262
Epoch 6/10
80/80 [==============================] - 54s 655ms/step - loss: 1.0403 - accuracy: 0.5970 - val_loss: 0.6693 - val_accuracy: 0.6262
Epoch 7/10
80/80 [==============================] - 52s 641ms/step - loss: 1.1712 - accuracy: 0.6013 - val_loss: 0.6659 - val_accuracy: 0.6262
Epoch 8/10
80/80 [==============================] - 64s 791ms/step - loss: 1.5193 - accuracy: 0.6041 - val_loss: 0.6646 - val_accuracy: 0.6262
Epoch 9/10
80/80 [==============================] - 53s 647ms/step - loss: 1.1659 - accuracy: 0.6017 - val_loss: 0.6635 - val_accuracy: 0.6262
Epoch 10/10
80/80 [==============================] - 52s 635ms/step - loss: 1.1565 - accuracy: 0.6009 - val_loss: 0.6638 - val_accuracy: 0.6262
```

Plotting Accuracy to Epoch

```
#Plots the accuarcy
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



The sequential model is a little different in that the increases we get without the convulation layer are significantly less. Additionally unlike the CNN architechure the model doesn't perform as well on the test data. In fact with a 60 percent sucess rate we are only slightly more likely to get the right answer than flipping a coin. This was after trying multiple neural net set-ups as well. Tried single Dense layer to 3 layer of Dense with multiple types of nodes at each layer.

CNN Model

Creating Model

```
# Prints the number of classes
num_classes = len(class_names)

## Makes a sequential model using the keras api for a CNN architechure. Has several layers that ends in a softmax for binary classification
model = Sequential([
  layers.Rescaling(1./255, input_shape=(200, 100, 3)),
  layers.Conv2D(100, (3, 3), activation = "relu"),
  layers.MaxPooling2D(pool_size = (2, 2)),
  layers.Conv2D(100, (3, 3), activation = 'relu'),
  layers.MaxPooling2D(pool_size = (2, 2)),
  layers.Flatten(),
  layers.Dropout(0.5),
  layers.Dense(2, activation = "softmax")
])
```

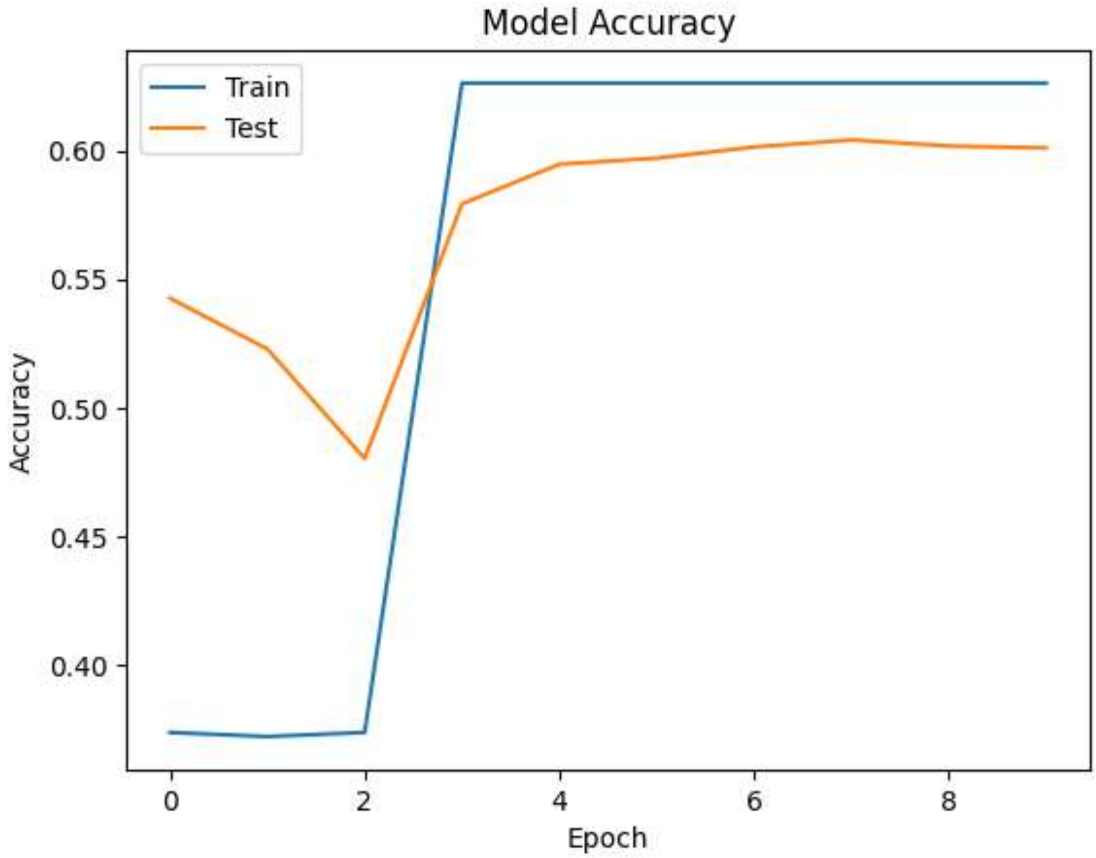
Showing Summary

```
## The summary of a model
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 200, 100, 3)       0

 conv2d (Conv2D)             (None, 198, 98, 100)      2800

 max_pooling2d (MaxPooling2D  (None, 99, 49, 100)      0
 )

 conv2d_1 (Conv2D)           (None, 97, 47, 100)       90100

 max_pooling2d_1 (MaxPooling  (None, 48, 23, 100)      0
 2D)

 flatten_1 (Flatten)         (None, 110400)            0

 dropout_3 (Dropout)         (None, 110400)            0

 dense_4 (Dense)             (None, 2)                 220802

=================================================================
Total params: 313,702
Trainable params: 313,702
Non-trainable params: 0
_________________________________________________________________
```

Compiling Model

```
## Compiles the model
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])
```


Model Accuracy
Train
Test
0.60
0.55
0.50
0.45
0.40
Accuracy
0
2
4
6
8
Epoch

Fitting Model

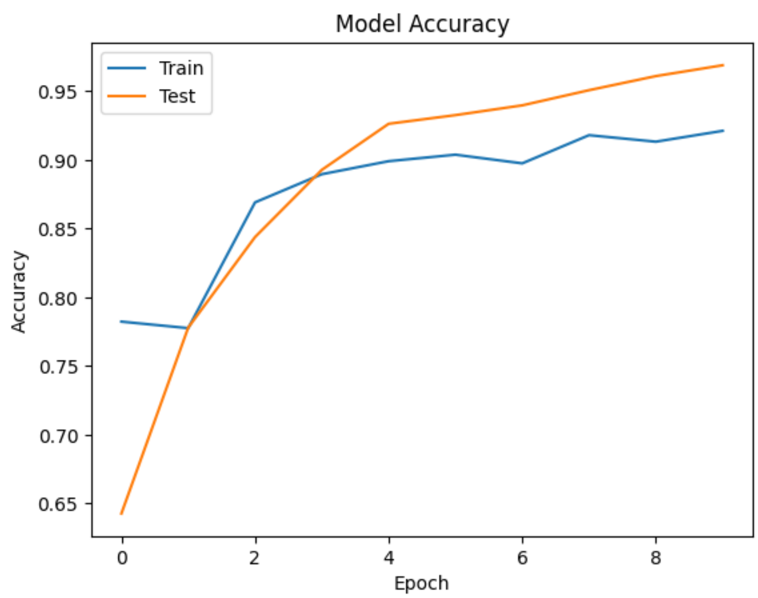
```
## Fits the model
epochs = 10
history = model.fit(
  train,
  validation_data=test,
  epochs=epochs,
  batch_size = batchSize,
  verbose = 1
)
```

```
Epoch 1/10
80/80 [==============================] - 382s 4s/step - loss: 0.6440 - accuracy: 0.6427 - val_loss: 0.4838 - val_accuracy: 0.7823
Epoch 2/10
80/80 [==============================] - 333s 4s/step - loss: 0.4826 - accuracy: 0.7780 - val_loss: 0.4874 - val_accuracy: 0.7776
Epoch 3/10
80/80 [==============================] - 337s 4s/step - loss: 0.3641 - accuracy: 0.8438 - val_loss: 0.3321 - val_accuracy: 0.8691
Epoch 4/10
80/80 [==============================] - 317s 4s/step - loss: 0.2795 - accuracy: 0.8927 - val_loss: 0.2644 - val_accuracy: 0.8896
Epoch 5/10
80/80 [==============================] - 344s 4s/step - loss: 0.2055 - accuracy: 0.9263 - val_loss: 0.2279 - val_accuracy: 0.8991
Epoch 6/10
80/80 [==============================] - 335s 4s/step - loss: 0.1991 - accuracy: 0.9326 - val_loss: 0.2120 - val_accuracy: 0.9038
Epoch 7/10
80/80 [==============================] - 337s 4s/step - loss: 0.1482 - accuracy: 0.9397 - val_loss: 0.2631 - val_accuracy: 0.8975
Epoch 8/10
80/80 [==============================] - 315s 4s/step - loss: 0.1350 - accuracy: 0.9507 - val_loss: 0.1932 - val_accuracy: 0.9180
Epoch 9/10
80/80 [==============================] - 332s 4s/step - loss: 0.1080 - accuracy: 0.9610 - val_loss: 0.1908 - val_accuracy: 0.9132
Epoch 10/10
80/80 [==============================] - 335s 4s/step - loss: 0.0848 - accuracy: 0.9688 - val_loss: 0.1819 - val_accuracy: 0.9211
```

Plotting Accuracy to Epoch

```
## Makes a plot with the accuracy of the model
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



When it comes to training this model with the CNN architechure we can see that even the starting accuracy is pretty high at around 70 percent. There are pretty sharp increases as well for the first 2 epochs. After epoch 3 we can see that the model switches to fine tuning the with some up and down in the train data. What is interesting is that the model definetly isn't overfitted to the dataset, because our test data performs better than the original training dataset.

I think a big reason for this is the convalution layer, but I would be interested to see if changing the image sizes back to their original size would help the model work a little better. Unfortunately I can't test this because I don't have enough ram to keep the pictures in memory.

Using Pretrained Model

by following the provided tutorial: https://www.tensorflow.org/tutorials/images/transfer_learning

Downloading Data

```
# Gets all of the data fom a given url and save it as a file
database_url = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin = database_url, extract = True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

# Getting train and test directories
train_directory = os.path.join(PATH, 'train')
test_directory = os.path.join(PATH, 'validation')

# Setting up batch and image sizes
batchSize = 32
imageSize = (200, 100)
```

```
Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 [==============================] - 2s 0us/step
```

Creating Train Subset

```
#Loads the data to memory
train = tf.keras.utils.image_dataset_from_directory(
    train_directory,
    shuffle = True,
    batch_size = batchSize,
    image_size = imageSize
)
```

```
Found 2000 files belonging to 2 classes.
```

Creating Test Subset

```
#Loads a seperate directory as the validation data set
test = tf.keras.utils.image_dataset_from_directory(
    test_directory,
    shuffle = True,
    batch_size = 32,
    image_size = imageSize
)
```

```
Found 1000 files belonging to 2 classes.
```

Data Exploration

```
# Gets all of the class names
class_names = train.class_names

## Prints pictures with all of the various classes
plt.figure(figsize = (10, 10))
for images, labels in train.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]])
```

Preparing to Create Model

```
## Splits the validation dataset and the test set into their batches for training efficency
test_batches = tf.data.experimental.cardinality(test)
test_dataset = test.take(test_batches // 5)
test = test.skip(test_batches // 5)

print('Count of test batches: %d' % tf.data.experimental.cardinality(test))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

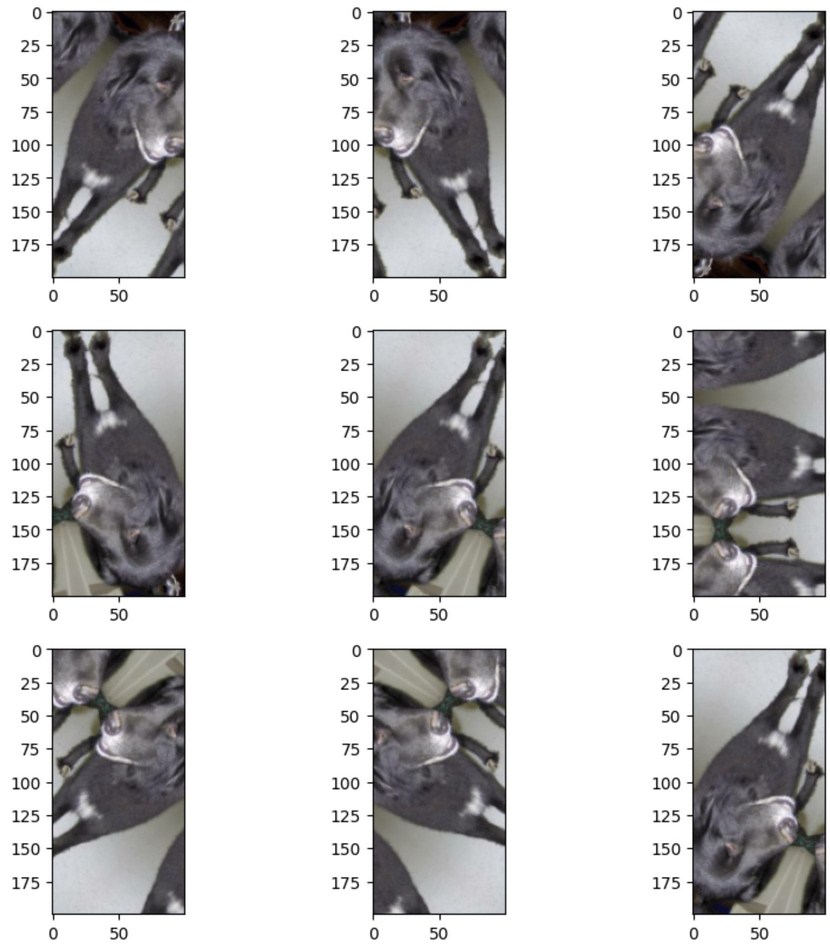
```
Count of test batches: 26
Number of test batches: 6
```

```
## Will prefetch the data
AUTOTUNE = tf.data.AUTOTUNE

train = train.prefetch(buffer_size = AUTOTUNE)
test = test.prefetch(buffer_size = AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size = AUTOTUNE)
```

```
## Augements the data by spinning the images to a certain extend
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('vertical'),
  tf.keras.layers.RandomRotation(0.6),
])
```

```
## Prints an augemented picture
for image, _ in train.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))
    plt.imshow(augmented_image[0] / 255)
```



```
## Makes a preprocessing unit that can be used to scale the images
preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input


## Rescales the images to be 1/127th the size they were originally
rescale = tf.keras.layers.Rescaling(1./127.5, offset = -1)
```

Creating Model

```
# Creating base model
image_shape = imageSize + (3,)

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

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224,
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_
9406464/9406464 [==============================] - 2s 0us/step
```

```
## Shows that batches are 32 images in a 5 by 5
image_batch, label_batch = next(iter(train))
```

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

```
(32, 7, 4, 1280)
```

Showing Summary

```
# Setting base model to trainable and printing summary
base_model.trainable = False
base_model.summary()
```

```
Model: "mobilenetv2_1.00_224"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 200, 100, 3  0           []
                                )]

 Conv1 (Conv2D)                 (None, 100, 50, 32)  864         ['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 100, 50, 32)  128         ['Conv1[0][0]']

 Conv1_relu (ReLU)              (None, 100, 50, 32)  0           ['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 100, 50, 32)  288        ['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 100, 50, 32)  128        ['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 100, 50, 32)  0          ['expanded_conv_depthwise_BN[0][0
 ReLU)                                                           ]']

 expanded_conv_project (Conv2D)  (None, 100, 50, 16)  512        ['expanded_conv_depthwise_relu[0]
                                                                 [0]']

 expanded_conv_project_BN (Batc  (None, 100, 50, 16)  64         ['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)        (None, 100, 50, 96)  1536        ['expanded_conv_project_BN[0][0]'
                                                                 ]

 block_1_expand_BN (BatchNormal  (None, 100, 50, 96)  384        ['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)     (None, 100, 50, 96)  0           ['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)    (None, 101, 51, 96)  0           ['block_1_expand_relu[0][0]']

 block_1_depthwise (DepthwiseCo  (None, 50, 25, 96)  864         ['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 50, 25, 96)  384         ['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 50, 25, 96)   0           ['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)       (None, 50, 25, 24)   2304        ['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 50, 25, 24)  96          ['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 50, 25, 144)  3456        ['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 50, 25, 144)  576        ['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 50, 25, 144)  0           ['block_2_expand_BN[0][0]']
```

```
# Adding a classification head
globalAverageLayer = tf.keras.layers.GlobalAveragePooling2D()
featureBatchAverage = globalAverageLayer(feature_batch)
print(featureBatchAverage.shape) ## Prints the batch sizes
```

```
(32, 1280)
```

```
# Using tf.keras.layers.Dense to change featuers to a single predicition per image
predictionLayer = tf.keras.layers.Dense(1)
```

```
Model: "mobilenetv2_1.00_224"
__________________________________________________________________________
________________________
 Layer (type)                Output Shape         Param #     Connected
to
==========================================================================
========================
 input_1 (InputLayer)        [(None, 200, 100, 3  0           []
                             )]

 Conv1 (Conv2D)              (None, 100, 50, 32)  864
['input_1[0][0]']

 bn_Conv1 (BatchNormalization)  (None, 100, 50, 32)  128
['Conv1[0][0]']

 Conv1_relu (ReLU)           (None, 100, 50, 32)  0
['bn_Conv1[0][0]']

 expanded_conv_depthwise (Depth  (None, 100, 50, 32)  288
['Conv1_relu[0][0]']
 wiseConv2D)

 expanded_conv_depthwise_BN (Ba  (None, 100, 50, 32)  128
['expanded_conv_depthwise[0][0]']
 tchNormalization)

 expanded_conv_depthwise_relu (  (None, 100, 50, 32)  0
['expanded_conv_depthwise_BN[0][0
 ReLU)                                                            ]']

 expanded_conv_project (Conv2D)  (None, 100, 50, 16)  512
['expanded_conv_depthwise_relu[0]
                                                                  [0]']

 expanded_conv_project_BN (Batc  (None, 100, 50, 16)  64
['expanded_conv_project[0][0]']
 hNormalization)

 block_1_expand (Conv2D)     (None, 100, 50, 96)  1536
['expanded_conv_project_BN[0][0]'
                                                                  ]

 block_1_expand_BN (BatchNormal  (None, 100, 50, 96)  384
['block_1_expand[0][0]']
 ization)

 block_1_expand_relu (ReLU)  (None, 100, 50, 96)  0
['block_1_expand_BN[0][0]']

 block_1_pad (ZeroPadding2D)  (None, 101, 51, 96)  0
['block_1_expand_relu[0][0]']
```

```
 block_1_depthwise (DepthwiseCo  (None, 50, 25, 96)  864
['block_1_pad[0][0]']
 nv2D)

 block_1_depthwise_BN (BatchNor  (None, 50, 25, 96)  384
['block_1_depthwise[0][0]']
 malization)

 block_1_depthwise_relu (ReLU)  (None, 50, 25, 96)   0
['block_1_depthwise_BN[0][0]']

 block_1_project (Conv2D)       (None, 50, 25, 24)   2304
['block_1_depthwise_relu[0][0]']

 block_1_project_BN (BatchNorma  (None, 50, 25, 24)  96
['block_1_project[0][0]']
 lization)

 block_2_expand (Conv2D)        (None, 50, 25, 144)  3456
['block_1_project_BN[0][0]']

 block_2_expand_BN (BatchNormal  (None, 50, 25, 144)  576
['block_2_expand[0][0]']
 ization)

 block_2_expand_relu (ReLU)     (None, 50, 25, 144)  0
['block_2_expand_BN[0][0]']

 block_2_depthwise (DepthwiseCo  (None, 50, 25, 144)  1296
['block_2_expand_relu[0][0]']
 nv2D)

 block_2_depthwise_BN (BatchNor  (None, 50, 25, 144)  576
['block_2_depthwise[0][0]']
 malization)

 block_2_depthwise_relu (ReLU)  (None, 50, 25, 144)  0
['block_2_depthwise_BN[0][0]']

 block_2_project (Conv2D)       (None, 50, 25, 24)   3456
['block_2_depthwise_relu[0][0]']

 block_2_project_BN (BatchNorma  (None, 50, 25, 24)  96
['block_2_project[0][0]']
 lization)

 block_2_add (Add)              (None, 50, 25, 24)   0
['block_1_project_BN[0][0]',

'block_2_project_BN[0][0]']

 block_3_expand (Conv2D)        (None, 50, 25, 144)  3456
['block_2_add[0][0]']
```

```
 block_3_expand_BN (BatchNormal  (None, 50, 25, 144)  576
['block_3_expand[0][0]']
 ization)

 block_3_expand_relu (ReLU)     (None, 50, 25, 144)  0
['block_3_expand_BN[0][0]']

 block_3_pad (ZeroPadding2D)    (None, 51, 27, 144)  0
['block_3_expand_relu[0][0]']

 block_3_depthwise (DepthwiseCo  (None, 25, 13, 144)  1296
['block_3_pad[0][0]']
 nv2D)

 block_3_depthwise_BN (BatchNor  (None, 25, 13, 144)  576
['block_3_depthwise[0][0]']
 malization)

 block_3_depthwise_relu (ReLU)  (None, 25, 13, 144)  0
['block_3_depthwise_BN[0][0]']

 block_3_project (Conv2D)       (None, 25, 13, 32)   4608
['block_3_depthwise_relu[0][0]']

 block_3_project_BN (BatchNorma  (None, 25, 13, 32)  128
['block_3_project[0][0]']
 lization)

 block_4_expand (Conv2D)        (None, 25, 13, 192)  6144
['block_3_project_BN[0][0]']

 block_4_expand_BN (BatchNormal  (None, 25, 13, 192)  768
['block_4_expand[0][0]']
 ization)

 block_4_expand_relu (ReLU)     (None, 25, 13, 192)  0
['block_4_expand_BN[0][0]']

 block_4_depthwise (DepthwiseCo  (None, 25, 13, 192)  1728
['block_4_expand_relu[0][0]']
 nv2D)

 block_4_depthwise_BN (BatchNor  (None, 25, 13, 192)  768
['block_4_depthwise[0][0]']
 malization)

 block_4_depthwise_relu (ReLU)  (None, 25, 13, 192)  0
['block_4_depthwise_BN[0][0]']

 block_4_project (Conv2D)       (None, 25, 13, 32)   6144
['block_4_depthwise_relu[0][0]']

 block_4_project_BN (BatchNorma  (None, 25, 13, 32)  128
['block_4_project[0][0]']
```

```
 lization)

 block_4_add (Add)              (None, 25, 13, 32)   0
['block_3_project_BN[0][0]',

'block_4_project_BN[0][0]']

 block_5_expand (Conv2D)        (None, 25, 13, 192)  6144
['block_4_add[0][0]']

 block_5_expand_BN (BatchNormal  (None, 25, 13, 192)  768
['block_5_expand[0][0]']
 ization)

 block_5_expand_relu (ReLU)     (None, 25, 13, 192)  0
['block_5_expand_BN[0][0]']

 block_5_depthwise (DepthwiseCo  (None, 25, 13, 192)  1728
['block_5_expand_relu[0][0]']
 nv2D)

 block_5_depthwise_BN (BatchNor  (None, 25, 13, 192)  768
['block_5_depthwise[0][0]']
 malization)

 block_5_depthwise_relu (ReLU)  (None, 25, 13, 192)  0
['block_5_depthwise_BN[0][0]']

 block_5_project (Conv2D)       (None, 25, 13, 32)   6144
['block_5_depthwise_relu[0][0]']

 block_5_project_BN (BatchNorma  (None, 25, 13, 32)  128
['block_5_project[0][0]']
 lization)

 block_5_add (Add)              (None, 25, 13, 32)   0
['block_4_add[0][0]',

'block_5_project_BN[0][0]']

 block_6_expand (Conv2D)        (None, 25, 13, 192)  6144
['block_5_add[0][0]']

 block_6_expand_BN (BatchNormal  (None, 25, 13, 192)  768
['block_6_expand[0][0]']
 ization)

 block_6_expand_relu (ReLU)     (None, 25, 13, 192)  0
['block_6_expand_BN[0][0]']

 block_6_pad (ZeroPadding2D)    (None, 27, 15, 192)  0
['block_6_expand_relu[0][0]']
```

```
 block_6_depthwise (DepthwiseCo  (None, 13, 7, 192)  1728
['block_6_pad[0][0]']
 nv2D)

 block_6_depthwise_BN (BatchNor  (None, 13, 7, 192)  768
['block_6_depthwise[0][0]']
 malization)

 block_6_depthwise_relu (ReLU)  (None, 13, 7, 192)   0
['block_6_depthwise_BN[0][0]']

 block_6_project (Conv2D)       (None, 13, 7, 64)    12288
['block_6_depthwise_relu[0][0]']

 block_6_project_BN (BatchNorma  (None, 13, 7, 64)   256
['block_6_project[0][0]']
 lization)

 block_7_expand (Conv2D)        (None, 13, 7, 384)   24576
['block_6_project_BN[0][0]']

 block_7_expand_BN (BatchNormal  (None, 13, 7, 384)  1536
['block_7_expand[0][0]']
 ization)

 block_7_expand_relu (ReLU)     (None, 13, 7, 384)   0
['block_7_expand_BN[0][0]']

 block_7_depthwise (DepthwiseCo  (None, 13, 7, 384)  3456
['block_7_expand_relu[0][0]']
 nv2D)

 block_7_depthwise_BN (BatchNor  (None, 13, 7, 384)  1536
['block_7_depthwise[0][0]']
 malization)

 block_7_depthwise_relu (ReLU)  (None, 13, 7, 384)   0
['block_7_depthwise_BN[0][0]']

 block_7_project (Conv2D)       (None, 13, 7, 64)    24576
['block_7_depthwise_relu[0][0]']

 block_7_project_BN (BatchNorma  (None, 13, 7, 64)   256
['block_7_project[0][0]']
 lization)

 block_7_add (Add)              (None, 13, 7, 64)    0
['block_6_project_BN[0][0]',

'block_7_project_BN[0][0]']

 block_8_expand (Conv2D)        (None, 13, 7, 384)   24576
['block_7_add[0][0]']
```

```
 block_8_expand_BN (BatchNormal  (None, 13, 7, 384)  1536
['block_8_expand[0][0]']
 ization)

 block_8_expand_relu (ReLU)     (None, 13, 7, 384)   0
['block_8_expand_BN[0][0]']

 block_8_depthwise (DepthwiseCo  (None, 13, 7, 384)  3456
['block_8_expand_relu[0][0]']
 nv2D)

 block_8_depthwise_BN (BatchNor  (None, 13, 7, 384)  1536
['block_8_depthwise[0][0]']
 malization)

 block_8_depthwise_relu (ReLU)  (None, 13, 7, 384)   0
['block_8_depthwise_BN[0][0]']

 block_8_project (Conv2D)       (None, 13, 7, 64)    24576
['block_8_depthwise_relu[0][0]']

 block_8_project_BN (BatchNorma  (None, 13, 7, 64)   256
['block_8_project[0][0]']
 lization)

 block_8_add (Add)              (None, 13, 7, 64)    0
['block_7_add[0][0]',

'block_8_project_BN[0][0]']

 block_9_expand (Conv2D)        (None, 13, 7, 384)   24576
['block_8_add[0][0]']

 block_9_expand_BN (BatchNormal  (None, 13, 7, 384)  1536
['block_9_expand[0][0]']
 ization)

 block_9_expand_relu (ReLU)     (None, 13, 7, 384)   0
['block_9_expand_BN[0][0]']

 block_9_depthwise (DepthwiseCo  (None, 13, 7, 384)  3456
['block_9_expand_relu[0][0]']
 nv2D)

 block_9_depthwise_BN (BatchNor  (None, 13, 7, 384)  1536
['block_9_depthwise[0][0]']
 malization)

 block_9_depthwise_relu (ReLU)  (None, 13, 7, 384)   0
['block_9_depthwise_BN[0][0]']

 block_9_project (Conv2D)       (None, 13, 7, 64)    24576
['block_9_depthwise_relu[0][0]']
```

```
 block_9_project_BN (BatchNorma  (None, 13, 7, 64)   256
['block_9_project[0][0]']
 lization)

 block_9_add (Add)              (None, 13, 7, 64)    0
['block_8_add[0][0]',

'block_9_project_BN[0][0]']

 block_10_expand (Conv2D)       (None, 13, 7, 384)   24576
['block_9_add[0][0]']

 block_10_expand_BN (BatchNorma  (None, 13, 7, 384)  1536
['block_10_expand[0][0]']
 lization)

 block_10_expand_relu (ReLU)    (None, 13, 7, 384)   0
['block_10_expand_BN[0][0]']

 block_10_depthwise (DepthwiseC  (None, 13, 7, 384)  3456
['block_10_expand_relu[0][0]']
 onv2D)

 block_10_depthwise_BN (BatchNo  (None, 13, 7, 384)  1536
['block_10_depthwise[0][0]']
 rmalization)

 block_10_depthwise_relu (ReLU)  (None, 13, 7, 384)  0
['block_10_depthwise_BN[0][0]']

 block_10_project (Conv2D)      (None, 13, 7, 96)    36864
['block_10_depthwise_relu[0][0]']

 block_10_project_BN (BatchNorm  (None, 13, 7, 96)   384
['block_10_project[0][0]']
 alization)

 block_11_expand (Conv2D)       (None, 13, 7, 576)   55296
['block_10_project_BN[0][0]']

 block_11_expand_BN (BatchNorma  (None, 13, 7, 576)  2304
['block_11_expand[0][0]']
 lization)

 block_11_expand_relu (ReLU)    (None, 13, 7, 576)   0
['block_11_expand_BN[0][0]']

 block_11_depthwise (DepthwiseC  (None, 13, 7, 576)  5184
['block_11_expand_relu[0][0]']
 onv2D)

 block_11_depthwise_BN (BatchNo  (None, 13, 7, 576)  2304
['block_11_depthwise[0][0]']
 rmalization)
```

```
 block_11_depthwise_relu (ReLU)  (None, 13, 7, 576)  0
['block_11_depthwise_BN[0][0]']

 block_11_project (Conv2D)      (None, 13, 7, 96)    55296
['block_11_depthwise_relu[0][0]']

 block_11_project_BN (BatchNorm  (None, 13, 7, 96)   384
['block_11_project[0][0]']
 alization)

 block_11_add (Add)             (None, 13, 7, 96)    0
['block_10_project_BN[0][0]',

'block_11_project_BN[0][0]']

 block_12_expand (Conv2D)       (None, 13, 7, 576)   55296
['block_11_add[0][0]']

 block_12_expand_BN (BatchNorma  (None, 13, 7, 576)  2304
['block_12_expand[0][0]']
 lization)

 block_12_expand_relu (ReLU)    (None, 13, 7, 576)   0
['block_12_expand_BN[0][0]']

 block_12_depthwise (DepthwiseC  (None, 13, 7, 576)  5184
['block_12_expand_relu[0][0]']
 onv2D)

 block_12_depthwise_BN (BatchNo  (None, 13, 7, 576)  2304
['block_12_depthwise[0][0]']
 rmalization)

 block_12_depthwise_relu (ReLU)  (None, 13, 7, 576)  0
['block_12_depthwise_BN[0][0]']

 block_12_project (Conv2D)      (None, 13, 7, 96)    55296
['block_12_depthwise_relu[0][0]']

 block_12_project_BN (BatchNorm  (None, 13, 7, 96)   384
['block_12_project[0][0]']
 alization)

 block_12_add (Add)             (None, 13, 7, 96)    0
['block_11_add[0][0]',

'block_12_project_BN[0][0]']

 block_13_expand (Conv2D)       (None, 13, 7, 576)   55296
['block_12_add[0][0]']

 block_13_expand_BN (BatchNorma  (None, 13, 7, 576)  2304
['block_13_expand[0][0]']
```

```
 lization)

 block_13_expand_relu (ReLU)    (None, 13, 7, 576)   0
['block_13_expand_BN[0][0]']

 block_13_pad (ZeroPadding2D)   (None, 15, 9, 576)   0
['block_13_expand_relu[0][0]']

 block_13_depthwise (DepthwiseC  (None, 7, 4, 576)   5184
['block_13_pad[0][0]']
 onv2D)

 block_13_depthwise_BN (BatchNo  (None, 7, 4, 576)   2304
['block_13_depthwise[0][0]']
 rmalization)

 block_13_depthwise_relu (ReLU)  (None, 7, 4, 576)   0
['block_13_depthwise_BN[0][0]']

 block_13_project (Conv2D)      (None, 7, 4, 160)    92160
['block_13_depthwise_relu[0][0]']

 block_13_project_BN (BatchNorm  (None, 7, 4, 160)   640
['block_13_project[0][0]']
 alization)

 block_14_expand (Conv2D)       (None, 7, 4, 960)    153600
['block_13_project_BN[0][0]']

 block_14_expand_BN (BatchNorma  (None, 7, 4, 960)   3840
['block_14_expand[0][0]']
 lization)

 block_14_expand_relu (ReLU)    (None, 7, 4, 960)    0
['block_14_expand_BN[0][0]']

 block_14_depthwise (DepthwiseC  (None, 7, 4, 960)   8640
['block_14_expand_relu[0][0]']
 onv2D)

 block_14_depthwise_BN (BatchNo  (None, 7, 4, 960)   3840
['block_14_depthwise[0][0]']
 rmalization)

 block_14_depthwise_relu (ReLU)  (None, 7, 4, 960)   0
['block_14_depthwise_BN[0][0]']

 block_14_project (Conv2D)      (None, 7, 4, 160)    153600
['block_14_depthwise_relu[0][0]']

 block_14_project_BN (BatchNorm  (None, 7, 4, 160)   640
['block_14_project[0][0]']
 alization)
```

```
 block_14_add (Add)             (None, 7, 4, 160)    0
['block_13_project_BN[0][0]',

'block_14_project_BN[0][0]']

 block_15_expand (Conv2D)       (None, 7, 4, 960)    153600
['block_14_add[0][0]']

 block_15_expand_BN (BatchNorma  (None, 7, 4, 960)   3840
['block_15_expand[0][0]']
 lization)

 block_15_expand_relu (ReLU)    (None, 7, 4, 960)    0
['block_15_expand_BN[0][0]']

 block_15_depthwise (DepthwiseC  (None, 7, 4, 960)   8640
['block_15_expand_relu[0][0]']
 onv2D)

 block_15_depthwise_BN (BatchNo  (None, 7, 4, 960)   3840
['block_15_depthwise[0][0]']
 rmalization)

 block_15_depthwise_relu (ReLU)  (None, 7, 4, 960)   0
['block_15_depthwise_BN[0][0]']

 block_15_project (Conv2D)      (None, 7, 4, 160)    153600
['block_15_depthwise_relu[0][0]']

 block_15_project_BN (BatchNorm  (None, 7, 4, 160)   640
['block_15_project[0][0]']
 alization)

 block_15_add (Add)             (None, 7, 4, 160)    0
['block_14_add[0][0]',

'block_15_project_BN[0][0]']

 block_16_expand (Conv2D)       (None, 7, 4, 960)    153600
['block_15_add[0][0]']

 block_16_expand_BN (BatchNorma  (None, 7, 4, 960)   3840
['block_16_expand[0][0]']
 lization)

 block_16_expand_relu (ReLU)    (None, 7, 4, 960)    0
['block_16_expand_BN[0][0]']

 block_16_depthwise (DepthwiseC  (None, 7, 4, 960)   8640
['block_16_expand_relu[0][0]']
 onv2D)

 block_16_depthwise_BN (BatchNo  (None, 7, 4, 960)   3840
['block_16_depthwise[0][0]']
```

```
 rmalization)

 block_16_depthwise_relu (ReLU)  (None, 7, 4, 960)   0
['block_16_depthwise_BN[0][0]']

 block_16_project (Conv2D)      (None, 7, 4, 320)    307200
['block_16_depthwise_relu[0][0]']

 block_16_project_BN (BatchNorm  (None, 7, 4, 320)   1280
['block_16_project[0][0]']
 alization)

 Conv_1 (Conv2D)                (None, 7, 4, 1280)   409600
['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalization)  (None, 7, 4, 1280)  5120
['Conv_1[0][0]']

 out_relu (ReLU)                (None, 7, 4, 1280)   0
['Conv_1_bn[0][0]']

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

```
predictionBatch = predictionLayer(featureBatchAverage)
print(predictionBatch.shape)
```

```
(32, 1)
```

Compiling Model

```
# Building model
inputs = tf.keras.Input(shape=(200, 100, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = globalAverageLayer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = predictionLayer(x)
model = tf.keras.Model(inputs, outputs)


# Compiling model
learningRate = 0.0001
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=learningRate),
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=['accuracy'])


# Printing summary
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 200, 100, 3)]     0

 sequential_1 (Sequential)   (None, 200, 100, 3)       0

 tf.math.truediv (TFOpLambda  (None, 200, 100, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 200, 100, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 7, 4, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense_1 (Dense)             (None, 1)                 1281

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

Fitting Model

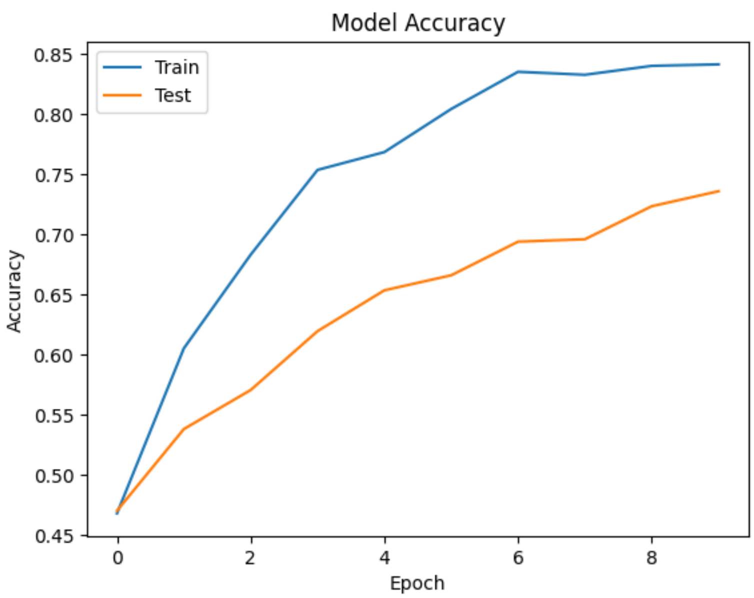
```
## Fits the model
epochs = 10
history = model.fit(train,
                    epochs=epochs,
                    validation_data=test)
```

```
Epoch 1/10
63/63 [==============================] - 58s 838ms/step - loss: 0.9881 - accuracy: 0.4700 - val_loss: 0.8405 - val_accuracy: 0.4678
Epoch 2/10
63/63 [==============================] - 59s 920ms/step - loss: 0.8186 - accuracy: 0.5380 - val_loss: 0.6594 - val_accuracy: 0.6052
Epoch 3/10
63/63 [==============================] - 59s 931ms/step - loss: 0.7639 - accuracy: 0.5705 - val_loss: 0.5421 - val_accuracy: 0.6832
Epoch 4/10
63/63 [==============================] - 60s 951ms/step - loss: 0.6733 - accuracy: 0.6195 - val_loss: 0.4619 - val_accuracy: 0.7537
Epoch 5/10
63/63 [==============================] - 61s 970ms/step - loss: 0.6346 - accuracy: 0.6535 - val_loss: 0.4221 - val_accuracy: 0.7686
```

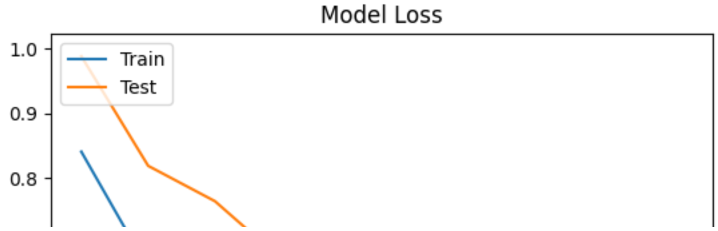
```
Epoch 6/10
63/63 [==============================] - 59s 941ms/step - loss: 0.6164 - accuracy: 0.6660 - val_loss: 0.3752 - val_accuracy: 0.8045
Epoch 7/10
63/63 [==============================] - 58s 926ms/step - loss: 0.5695 - accuracy: 0.6940 - val_loss: 0.3335 - val_accuracy: 0.8354
Epoch 8/10
63/63 [==============================] - 59s 941ms/step - loss: 0.5637 - accuracy: 0.6960 - val_loss: 0.3210 - val_accuracy: 0.8329
Epoch 9/10
63/63 [==============================] - 60s 947ms/step - loss: 0.5379 - accuracy: 0.7235 - val_loss: 0.3058 - val_accuracy: 0.8403
Epoch 10/10
63/63 [==============================] - 53s 850ms/step - loss: 0.5270 - accuracy: 0.7360 - val_loss: 0.2950 - val_accuracy: 0.8416
```

Plotting Accuracy and Loss to Epoch

```
## Makes a plot with the accuracy of the model
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show

prev_acc = history.history['accuracy']
```




```
## Makes a plot with the loss of the model
plt.plot(history.history['val_loss'])
plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show

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



This model uses the CNN model previously constructed. We can see that at least in regards to its current data of cats and dogs it has a success rate of around 75 percent after 10 epochs. We can see from the loss diagram that, as the model progressed through the epochs, that the model was able to improve in its predictions -- a fact that is reflected in the way accuracy rised throughout each epoch. Though not required, in the next steps we are going to fine tune the model to attempt to see if we can improve the performance of the model at all.

Additional Steps: Fine Tuning

Preparing for Fine Tuning

```
## Allows for finetuning from other people or ourselves
base_model.trainable = True
```

```
# Unfreezing the base model and setting bottom layers to untrainable
fine_tune_at = 100
for layer in base_model.layers[:fine_tune_at]:
  layer.trainable = False
```

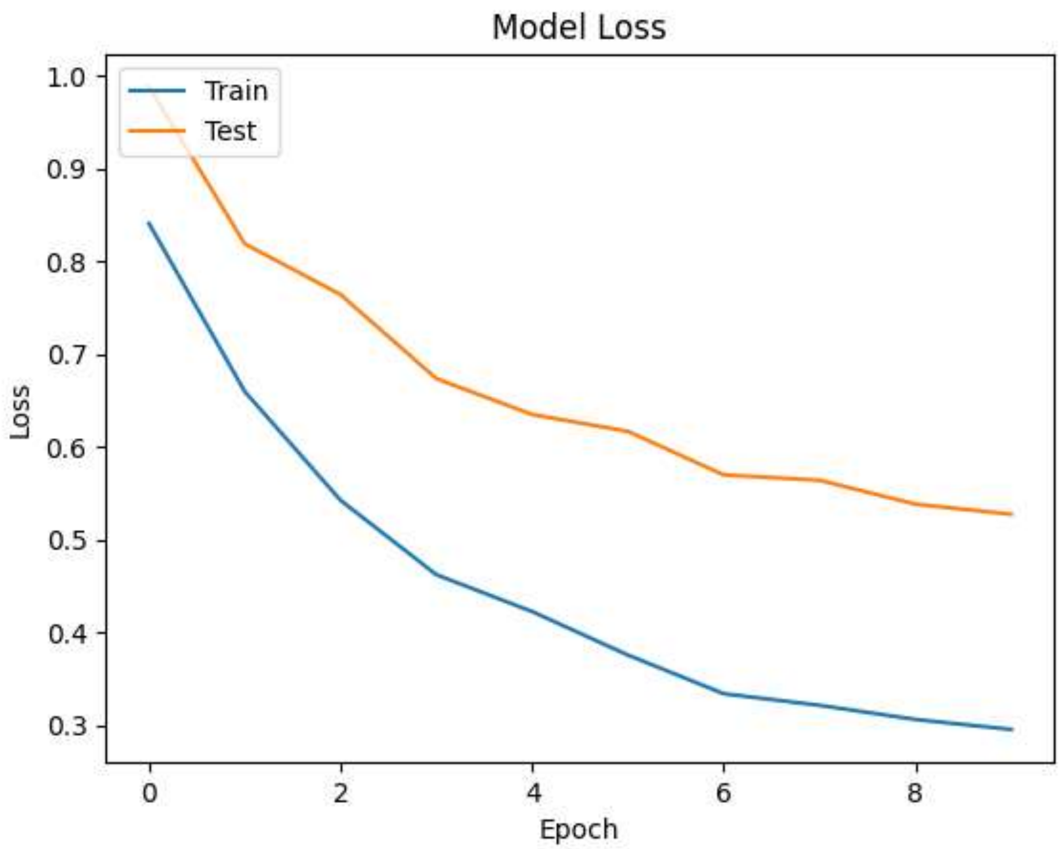
Compiling Model

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

Showing Summary

```
model.summary()

## Model has 8 different levels to train and most are transformation layers though and
## only at the mobilenet and dense layers are we training the model
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 200, 100, 3)]     0

 sequential_1 (Sequential)   (None, 200, 100, 3)       0

 tf.math.truediv (TFOpLambda  (None, 200, 100, 3)      0
 )

 tf.math.subtract (TFOpLambd  (None, 200, 100, 3)      0
 a)

 mobilenetv2_1.00_224 (Funct  (None, 7, 4, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout_1 (Dropout)         (None, 1280)              0

 dense_1 (Dense)             (None, 1)                 1281

=================================================================
Total params: 2,259,265
```


Model Loss
Train
Test
Loss
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8
Epoch

```
Trainable params: 1,862,721
Non-trainable params: 396,544
_________________________________________________________________
```

Fitting Model

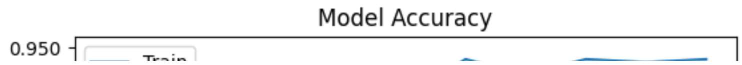
```
## We have to pass through the hyper parameters that will define the new model. This shows through as additonal epochs 11-20
## that can be seen as additional training

tunedEpochs = 10
totalEpochs =  epochs + tunedEpochs

history_fine = model.fit(
    train,
    epochs=totalEpochs,
    initial_epoch=history.epoch[-1],
    validation_data=test
  )
```

```
Epoch 10/20
63/63 [==============================] - 99s 1s/step - loss: 0.4866 - accuracy: 0.7690 - val_loss: 0.2118 - val_accuracy: 0.8936
Epoch 11/20
63/63 [==============================] - 92s 1s/step - loss: 0.3859 - accuracy: 0.8125 - val_loss: 0.1547 - val_accuracy: 0.9356
Epoch 12/20
63/63 [==============================] - 86s 1s/step - loss: 0.3524 - accuracy: 0.8360 - val_loss: 0.1558 - val_accuracy: 0.9381
Epoch 13/20
63/63 [==============================] - 89s 1s/step - loss: 0.3420 - accuracy: 0.8395 - val_loss: 0.1635 - val_accuracy: 0.9134
Epoch 14/20
63/63 [==============================] - 92s 1s/step - loss: 0.3268 - accuracy: 0.8455 - val_loss: 0.1499 - val_accuracy: 0.9369
Epoch 15/20
63/63 [==============================] - 92s 1s/step - loss: 0.2991 - accuracy: 0.8645 - val_loss: 0.1442 - val_accuracy: 0.9307
Epoch 16/20
63/63 [==============================] - 91s 1s/step - loss: 0.3091 - accuracy: 0.8525 - val_loss: 0.1553 - val_accuracy: 0.9455
Epoch 17/20
63/63 [==============================] - 99s 2s/step - loss: 0.3062 - accuracy: 0.8635 - val_loss: 0.1470 - val_accuracy: 0.9394
Epoch 18/20
63/63 [==============================] - 85s 1s/step - loss: 0.2901 - accuracy: 0.8745 - val_loss: 0.1439 - val_accuracy: 0.9455
Epoch 19/20
63/63 [==============================] - 86s 1s/step - loss: 0.2794 - accuracy: 0.8740 - val_loss: 0.1333 - val_accuracy: 0.9443
Epoch 20/20
63/63 [==============================] - 92s 1s/step - loss: 0.2611 - accuracy: 0.8820 - val_loss: 0.1310 - val_accuracy: 0.9455
```

Plotting Accuracy and Loss to Epoch

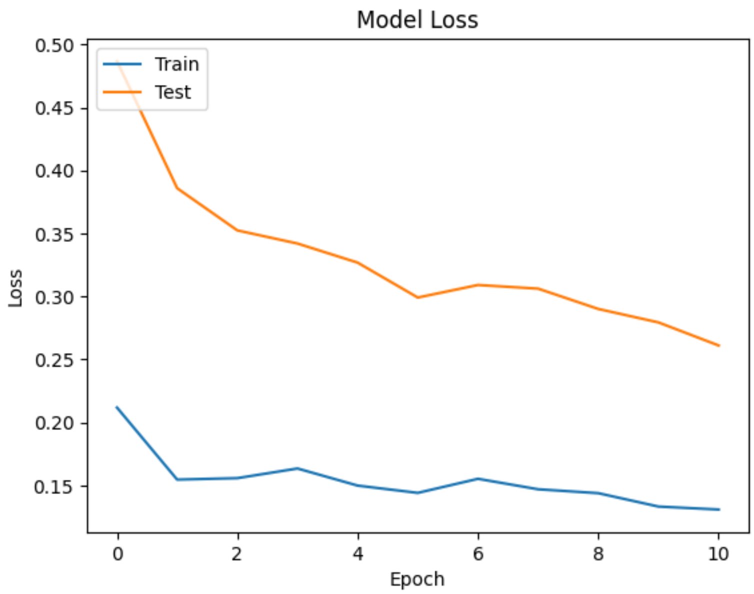
```
## Makes a plot with the accuracy of the model
plt.plot(history_fine.history['val_accuracy'])
plt.plot(history_fine.history['accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show
```

<function matplotlib.pyplot.show(close=None, block=None)>



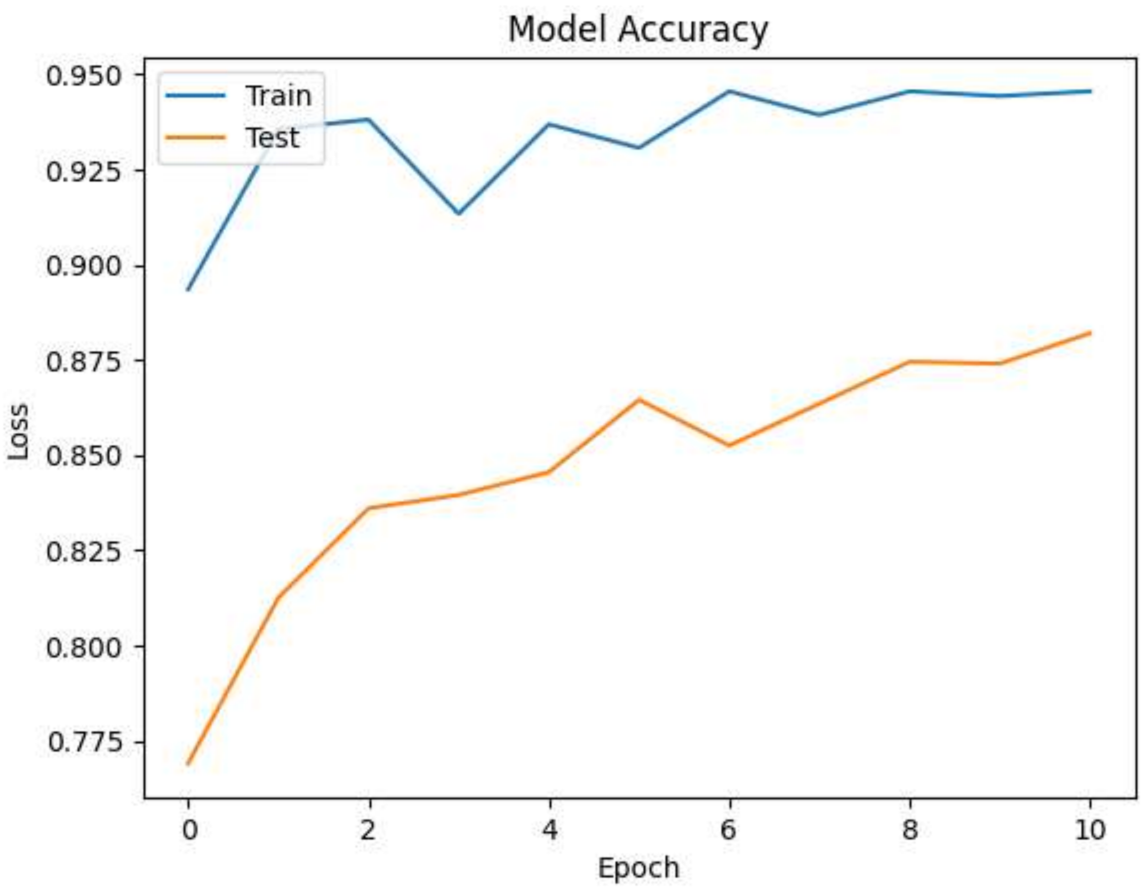

```
## Makes a plot with the loss of the model
plt.plot(history_fine.history['val_loss'])
plt.plot(history_fine.history['loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc = 'upper left')
plt.show
```

<function matplotlib.pyplot.show(close=None, block=None)>



With fine tuning, we were able to steadily increase the accuracy of our model and, in turn, decrease the loss. One important thing to note is that the test subset has the greatest response to fine tuning, as seen in both the "Model Accuracy" and "Model Loss" graphs. In contrast, the train subset had little impact, but steadily showed improvement with each epoch.

✓ 2s completed at 1:48 PM


Model Accuracy
Train
Test
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
Loss
0
2
4
6
8
10
Epoch