End-to-end Multil-class Dog Breed Classification

This notebook builds an end-to-end multi-class image classifier using TensorFlow 2.0 and TensorFlow Hub.

1. Problem

Identifying the breed of a dog given an image of a dog.

2. Data

The data we're using is from Kaggle's dog breed identification competition.

https://www.kaggle.com/c/dog-breed-identification/data

3. Evaluation

The evaluation is a file with prediction probabilities for each dog breed of each test image.

https://www.kaggle.com/c/dog-breed-identification/overview/evaluation

4. Features

Some information about the data:

- We're dealing with images (unstructured data) so it's probably best we use deep learning/transfer learning.
- There are 120 breeds of dogs (this means there are 120 different classes).
- There are around 10,000+ images in the training set (these images have labels)
- There are around 10,000+ images in the test set (these images have no labels, because we'll want to predict them)

```
# Use the '-d' parameter as the destination for where the files should go
#!unzip "drive/MyDrive/dog Vision/dog-breed-identification.zip" -d "drive/MyDrive/dog Visi
```

Get our workspace ready

- Import TenserFlow
- Import TenserFlow Hub
- we sure we use GPU

```
# Import TF 2.x
try:
  # %tensorflow_version only exists in Colab
```

```
  %tensorflow_version 2.x
except Exception:
  pass
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

```
# import necessory tools
import tensorflow as tf
import tensorflow_hub as hub
print("TF version:",tf.__version__)
print("TF Hub version:", hub.__version__)
# check for Gpu
print("GPU", "yes" if tf.config.list_physical_devices("GPU") else "not")
```

```
TF version: 2.11.0
TF Hub version: 0.12.0
GPU yes
```

Getting our data ready (tuning into Tensors)

All machine learning models, our data has to be in numerical format so change it into numbers

Tuning our Image into Tensors (numerical representation)

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

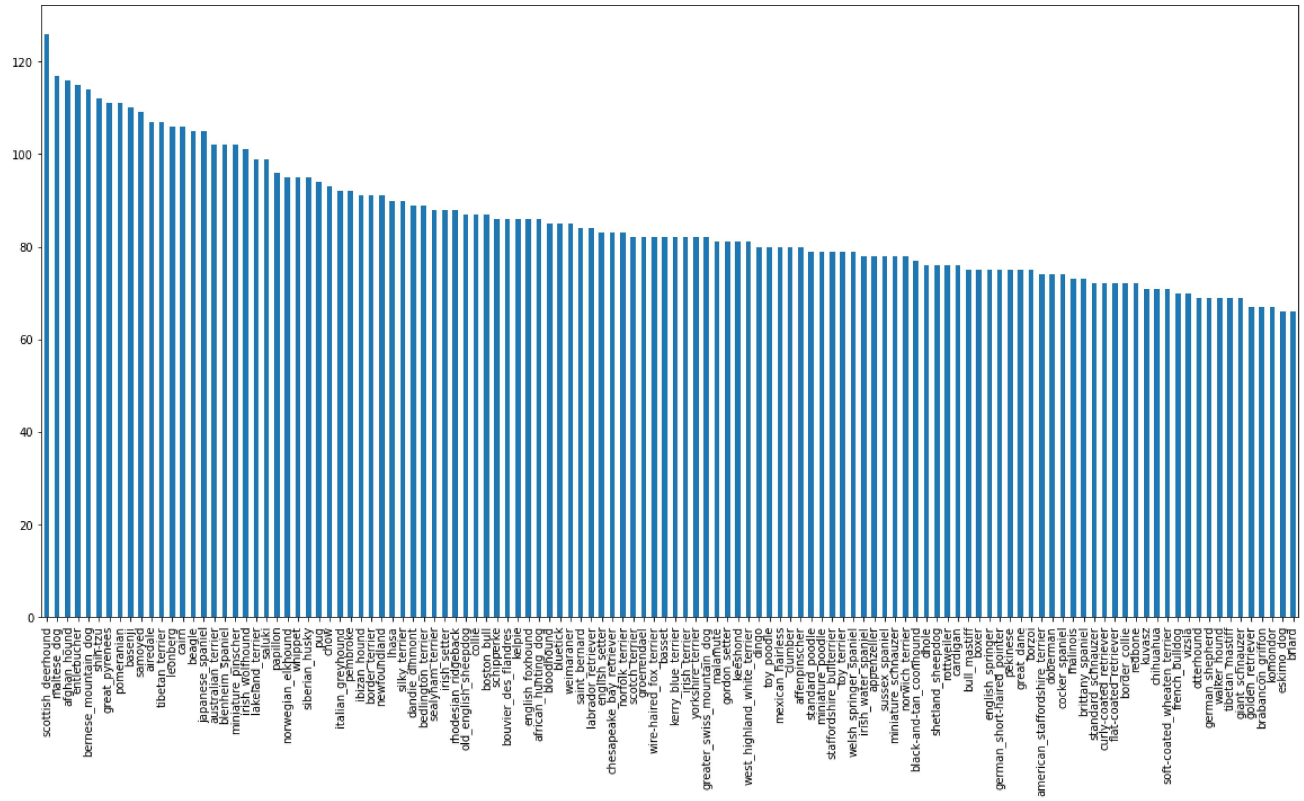
```
 #Lets start by accessing our data and chexhing out the labels
import pandas as pd
labels_csv = pd.read_csv("drive/MyDrive/dog Vision/labels.csv")
print(labels_csv.describe())
print(labels_csv.head())
```

```
                                     id               breed
count                             10222               10222
unique                            10222                 120
top     000bec180eb18c7604dcecc8fe0dba07  scottish_deerhound
freq                                  1                 126
                                 id             breed
0  000bec180eb18c7604dcecc8fe0dba07       boston_bull
1  001513dfcb2ffafc82cccf4d8bbaba97             dingo
2  001cdf01b096e06d78e9e5112d419397          pekinese
3  00214f311d5d2247d5dfe4fe24b2303d          bluetick
4  0021f9ceb3235effd7fcde7f7538ed62  golden_retriever
```

```
labels_csv.head()
```

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

```
# How many images are there of each breaad
labels_csv["breed"].value_counts().plot.bar(figsize= (20, 10));
```




```
from IPython.display import display, Image
```

```
labels_csv["breed"].value_counts().median()
```

```
82.0
```

```
# let's view an immage
from IPython.display import Image
Image("drive/MyDrive/dog Vision/train/001cdf01b096e06d78e9e5112d419397.jpg")
```

Getting images and their label

Let's get a list of all our image file pathnames

```
# Create path name from image ID's
filenames = ["drive/MyDrive/dog Vision/train/" + fname + ".jpg" for fname in labels_csv["i
```

```
# Check the first 10
filenames[:10]
```

```
['drive/MyDrive/dog Vision/train/000bec180eb18c7604dcecc8fe0dba07.jpg',
 'drive/MyDrive/dog Vision/train/001513dfcb2ffafc82cccf4d8bbaba97.jpg',
 'drive/MyDrive/dog Vision/train/001cdf01b096e06d78e9e5112d419397.jpg',
 'drive/MyDrive/dog Vision/train/00214f311d5d2247d5dfe4fe24b2303d.jpg',
 'drive/MyDrive/dog Vision/train/0021f9ceb3235effd7fcde7f7538ed62.jpg',
 'drive/MyDrive/dog Vision/train/002211c81b498ef88e1b40b9abf84e1d.jpg',
 'drive/MyDrive/dog Vision/train/00290d3e1fdd27226ba27a8ce248ce85.jpg',
 'drive/MyDrive/dog Vision/train/002a283a315af96eaea0e28e7163b21b.jpg',
 'drive/MyDrive/dog Vision/train/003df8b8a8b05244b1d920bb6cf451f9.jpg',
 'drive/MyDrive/dog Vision/train/0042188c895a2f14ef64a918ed9c7b64.jpg']
```

```
# Check wether number of filenames matches number of actual image files
import os
```

```
if len(os.listdir("drive/MyDrive/dog Vision/train")) == len(filenames):
  print("Filenames match")
else:
  print("not match")
```

```
Filenames match
```

```
# one more check
Image (filenames[9000])
```


```
labels_csv["breed"][9000]
```

```
'tibetan_mastiff'
```

since we've now get our training image filenames in alist let's prepare our labels

```
import numpy as np
labels = labels_csv["breed"].to_numpy() # convert labels column to NumPy array
labels[:10]
```

```
     array(['boston_bull', 'dingo', 'pekinese', 'bluetick', 'golden_retriever',
            'bedlington_terrier', 'bedlington_terrier', 'borzoi', 'basenji',
            'scottish_deerhound'], dtype=object)
```

```
len(labels)
```

```
     10222
```

```
# see if number of labels matches the number of filenames
if len(labels) == len(filenames):
  print("Number of labels matches number of filenames")
else:
  print("not match check data")
```

```
     Number of labels matches number of filenames
```

```
# Find the unique label value
unique_breeds = np.unique(labels)
len(unique_breeds)
```

```
     120
```

```
unique_breeds
```

```
     array(['affenpinscher', 'afghan_hound', 'african_hunting_dog', 'airedale',
            'american_staffordshire_terrier', 'appenzeller',
            'australian_terrier', 'basenji', 'basset', 'beagle',
            'bedlington_terrier', 'bernese_mountain_dog',
            'black-and-tan_coonhound', 'blenheim_spaniel', 'bloodhound',
            'bluetick', 'border_collie', 'border_terrier', 'borzoi',
            'boston_bull', 'bouvier_des_flandres', 'boxer',
            'brabancon_griffon', 'briard', 'brittany_spaniel', 'bull_mastiff',
            'cairn', 'cardigan', 'chesapeake_bay_retriever', 'chihuahua',
            'chow', 'clumber', 'cocker_spaniel', 'collie',
            'curly-coated_retriever', 'dandie_dinmont', 'dhole', 'dingo',
            'doberman', 'english_foxhound', 'english_setter',
            'english_springer', 'entlebucher', 'eskimo_dog',
            'flat-coated_retriever', 'french_bulldog', 'german_shepherd',
            'german_short-haired_pointer', 'giant_schnauzer',
            'golden_retriever', 'gordon_setter', 'great_dane',
            'great_pyrenees', 'greater_swiss_mountain_dog', 'groenendael',
            'ibizan_hound', 'irish_setter', 'irish_terrier',
            'irish_water_spaniel', 'irish_wolfhound', 'italian_greyhound',
            'japanese_spaniel', 'keeshond', 'kelpie', 'kerry_blue_terrier',
            'komondor', 'kuvasz', 'labrador_retriever', 'lakeland_terrier',
            'leonberg', 'lhasa', 'malamute', 'malinois', 'maltese_dog',
            'mexican_hairless', 'miniature_pinscher', 'miniature_poodle',
            'miniature_schnauzer', 'newfoundland', 'norfolk_terrier',
            'norwegian_elkhound', 'norwich_terrier', 'old_english_sheepdog',
            'otterhound', 'papillon', 'pekinese', 'pembroke', 'pomeranian',
            'pug', 'redbone', 'rhodesian_ridgeback', 'rottweiler',
            'saint_bernard', 'saluki', 'samoyed', 'schipperke',
            'scotch_terrier', 'scottish_deerhound', 'sealyham_terrier',
            'shetland_sheepdog', 'shih-tzu', 'siberian_husky', 'silky_terrier',
```

```
       'soft-coated_wheaten_terrier', 'staffordshire_bullterrier',
       'standard_poodle', 'standard_schnauzer', 'sussex_spaniel',
       'tibetan_mastiff', 'tibetan_terrier', 'toy_poodle', 'toy_terrier',
       'vizsla', 'walker_hound', 'weimaraner', 'welsh_springer_spaniel',
       'west_highland_white_terrier', 'whippet',
       'wire-haired_fox_terrier', 'yorkshire_terrier'], dtype=object)
```

```
# Turn a single label into an array of booleans
print(labels[0])
labels[0] == unique_breeds
```

```
boston_bull
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False])
```

```
# Turn every label into a boolean array
boolean_labels = [label == unique_breeds for label in labels]
boolean_labels[:2]
```

```
[array([False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False,  True, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False]),
 array([False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False,  True, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
        False, False, False, False, False, False, False, False, False,
```

```
            False, False, False, False, False, False, False, False, False,
            False, False, False])]
```

```
len (boolean_labels)
```

```
10222
```

```
# Turning boolean array into numbers

print(labels[0]) # original label
print(np.where(unique_breeds == labels[0])[0][0]) # index where label occurs
print(boolean_labels[0].argmax()) # index where label occurs in boolean array
print(boolean_labels[0].astype(int)) # there will be a 1 where the sample label occurs
```

```
boston_bull
19
19
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0]
```

```
print(labels[2])
print(boolean_labels[2].astype(int))
```

```
pekinese
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0]
```

Creating our own validation set

datasert from kaggel doesn't come with validaation set we're going to create our own

```
# set x and y
X = filenames
y = boolean_labels
```

```
len(filenames)
```

```
10222
```

we're going to start experimenting with 1000 image and increase as needed

```
# set number of image to use for experimanting
NUM_IMAGES = 1000 #@param {type:"slider", min:1000, max:10000, step:1000}
```

NUM_IMAGES: 1000

```
# Import train_test_split from Scikit-Learn
from sklearn.model_selection import train_test_split

# Split them into training and validation using NUM_IMAGES
X_train, X_val, y_train, y_val = train_test_split(X[:NUM_IMAGES],
                                                  y[:NUM_IMAGES],
                                                  test_size=0.2,
                                                  random_state=42)

len(X_train), len(y_train), len(X_val), len(y_val)
```

```
(800, 800, 200, 200)
```

```
#let's have a geez at the training data
X_train[:5], y_train[:2]
```

```
(['drive/MyDrive/dog Vision/train/00bee065dcec471f26394855c5c2f3de.jpg',
  'drive/MyDrive/dog Vision/train/0d2f9e12a2611d911d91a339074c8154.jpg',
  'drive/MyDrive/dog Vision/train/1108e48ce3e2d7d7fb527ae6e40ab486.jpg',
  'drive/MyDrive/dog Vision/train/0dc3196b4213a2733d7f4bdcd41699d3.jpg',
  'drive/MyDrive/dog Vision/train/146fbfac6b5b1f0de83a5d0c1b473377.jpg'],
 [array([False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False,  True,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False]),
  array([False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False,  True, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False, False, False, False, False, False, False,
         False, False, False])])
```

turning images into tensors

to preprocess our images into tensors we're going to write a funtion we do:

1. Take a image file path as input

2. Use TensorFlow to read the file and save it to variable 'image'
3. Turn our image (a jpg) into Tensors
4. resize the image to be a shape of (224, 224)
5. return a modified image

```
# convert a image into numpy
from matplotlib.pyplot import imread
image = imread(filenames[42])
image.shape
```

```
(257, 350, 3)
```

```
# turn image into Tensor
tf.constant(image)[:2]
```

```
<tf.Tensor: shape=(2, 350, 3), dtype=uint8, numpy=
array([[[ 89, 137,  87],
        [ 76, 124,  74],
        [ 63, 111,  59],
        ...,
        [ 76, 134,  86],
        [ 76, 134,  86],
        [ 76, 134,  86]],

       [[ 72, 119,  73],
        [ 67, 114,  68],
        [ 63, 111,  63],
        ...,
        [ 75, 131,  84],
        [ 74, 132,  84],
        [ 74, 131,  86]]], dtype=uint8)>
```

```
image.max(), image.min()
```

```
(255, 0)
```

Now we've seen what image looks like as a Tensor let's make a funtion to preprocess them

1. Take a image file path as input
2. Use TensorFlow to read the file and save it to variable 'image'
3. Turn our image (a jpg) into Tensors
4. Normalize our image (convert color channels value from 0-255, to 0-1
5. resize the image to be a shape of (224, 224)
6. return a modified image

```
# Define image size
IMG_SIZE = 224

# Create a function for preprocessing images
def process_image(image_path, img_size=IMG_SIZE):
```

```
  """
  Takes an image file path and turns the image into a Tensor.
  """
  # Read in an image file
  image = tf.io.read_file(image_path)
  # Turn the jpeg image into numerical Tensor with 3 colour channels (Red, Green, Blue)
  image = tf.image.decode_jpeg(image, channels=3)
  # Convert the colour channel values from 0-255 to 0-1 values
  image = tf.image.convert_image_dtype(image, tf.float32)
  # Resize the image to our desired value (224, 224)
  image = tf.image.resize(image, size=[IMG_SIZE, IMG_SIZE])

  return image
```

Turning our data into batches

why turn our data into batches?

as we're trying to process 10k+ images they all might not fit in the memory

so that's why we do about 32(this is batch size) image at a time you can manually adjust the batch size if need

in order to use Tensorflow effectively we need our data in the form of Tensor typle which look like this: '(image, label)

```
# Create a simple function to return a tuple (image, label)
def get_image_label(image_path, label):
  """
  Takes an image file path name and the assosciated label,
  processes the image and reutrns a typle of (image, label).
  """
  image = process_image(image_path)
  return image, label


#demo of above code
process_image(X[42], tf.constant(y[42]))
```

```
     <tf.Tensor: shape=(224, 224, 3), dtype=float32, numpy=
     array([[[0.3264178 , 0.5222886 , 0.3232816 ],
             [0.2537167 , 0.44366494, 0.24117757],
             [0.25699762, 0.4467087 , 0.23893751],
             ...,
             [0.29325107, 0.5189916 , 0.3215547 ],
             [0.29721776, 0.52466875, 0.33030328],
             [0.2948505 , 0.5223015 , 0.33406618]],

            [[0.25903144, 0.4537807 , 0.27294815],
             [0.24375686, 0.4407019 , 0.2554778 ],
             [0.2838985 , 0.47213382, 0.28298813],
             ...,
             [0.2785345 , 0.5027992 , 0.31004712],
             [0.28428748, 0.5108719 , 0.32523635],
```

```
          [0.28821915, 0.5148036 , 0.32916805]],

         [[0.20941195, 0.40692952, 0.25792548],
          [0.24045378, 0.43900946, 0.2868911 ],
          [0.29001117, 0.47937486, 0.32247734],
          ...,
          [0.26074055, 0.48414773, 0.30125174],
          [0.27101526, 0.49454468, 0.32096273],
          [0.27939945, 0.5029289 , 0.32934693]],

         ...,

         [[0.00634795, 0.03442048, 0.0258106 ],
          [0.01408936, 0.04459917, 0.0301715 ],
          [0.01385712, 0.04856448, 0.02839671],
          ...,
          [0.4220516 , 0.39761978, 0.21622123],
          [0.47932503, 0.45370543, 0.2696505 ],
          [0.48181024, 0.45828083, 0.27004552]],

         [[0.00222061, 0.02262166, 0.03176915],
          [0.01008397, 0.03669046, 0.02473482],
          [0.00608852, 0.03890046, 0.01207283],
          ...,
          [0.36070833, 0.33803678, 0.16216145],
          [0.42499566, 0.3976801 , 0.21701711],
          [0.4405433 , 0.4139589 , 0.23183356]],

         [[0.05608025, 0.06760229, 0.10401428],
          [0.05441074, 0.07435255, 0.05428263],
          [0.04734282, 0.07581793, 0.02060942],
          ...,
          [0.3397559 , 0.31265694, 0.14725602],
          [0.387725  , 0.360274  , 0.18714729],
          [0.43941984, 0.41196886, 0.23884216]]], dtype=float32)>
```

Now we've got a way to turn our data into tuples of tensors in form (image, label), let's make a function to turn all our data (x & y) into baches

```
# Define the batch size, 32 is a good start
BATCH_SIZE = 32

# Create a function to turn data into batches
def create_data_batches(X, y=None, batch_size=BATCH_SIZE, valid_data=False, test_data=Fals
  """
  Creates batches of data out of image (X) and label (y) pairs.
  Shuffles the data if it's training data but doesn't shuffle if it's validation data.
  Also accepts test data as input (no labels).
  """
  # If the data is a test dataset, we probably don't have have labels
  if test_data:
    print("Creating test data batches...")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X))) # only filepaths (no label
    data_batch = data.map(process_image).batch(BATCH_SIZE)
    return data_batch
```

```
  # If the data is a valid dataset, we don't need to shuffle it
  elif valid_data:
    print("Creating validation data batches...")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X), # filepaths
                                               tf.constant(y))) # labels
    data_batch = data.map(get_image_label).batch(BATCH_SIZE)
    return data_batch

  else:
    print("Creating training data batches...")
    # Turn filepaths and labels into Tensors
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X),
                                               tf.constant(y)))
    # Shuffling pathnames and labels before mapping image processor function is faster tha
    data = data.shuffle(buffer_size=len(X))

    # Create (image, label) tuples (this also turns the iamge path into a preprocessed ima
    data = data.map(get_image_label)

    # Turn the training data into batches
    data_batch = data.batch(BATCH_SIZE)
  return data_batch


# Create training and validation data batches
train_data = create_data_batches(X_train, y_train)
val_data = create_data_batches(X_val, y_val, valid_data=True)
```

```
Creating training data batches...
Creating validation data batches...
```

```
# Check out the different attributes of our data batches
train_data.element_spec, val_data.element_spec
```

```
((TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 120), dtype=tf.bool, name=None)),
 (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 120), dtype=tf.bool, name=None)))
```

Visulalizing Data batches

our data is now in batches, however these can be a little hard so lets visualize

```
import matplotlib.pyplot as plt

# Create a function for viewing images in a data batch
def show_25_images(images, labels):
  """
  Displays a plot of 25 images and their labels from a data batch.
  """
  # Setup the figure
  plt.figure(figsize=(10, 10))
```

```
  # Loop through 25 (for displaying 25 images)
  for i in range(25):
    # Create subplots (5 rows, 5 columns)
    ax = plt.subplot(5, 5, i+1)
    # Display an image
    plt.imshow(images[i])
    # Add the image label as the title
    plt.title(unique_breeds[labels[i].argmax()])
    # Turn the grid lines off
    plt.axis("off")


train_data
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 120), dtype=tf.bool, name=None))>
```

```
#train_images, train_labels = next(train_data.as_numpy_iterator())
#len(train_images), len(train_labels)


# now let's visualize the data in a training bartch
train_images, train_labels = next(train_data.as_numpy_iterator())
show_25_images(train_images, train_labels)
```

```
#now let visulize our validation set
val_images, val_labels = next(val_data.as_numpy_iterator())
show_25_images(val_images, val_labels)
```

Building a model

Before we build a model, there are a few things we need to define:

- The input shape (our images shape, in the form of Tensors) to our model.
- The output shape (image labels, in the form of Tensors) of our model.
- The URL of the model we want to use from TensorFlow Hub - https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/4

```
IMG_SIZE
```

```
224
```

```
# Setup input shape to the model
INPUT_SHAPE = [None, IMG_SIZE, IMG_SIZE, 3] # batch, height, width, colour channels

# Setup output shape of our model
OUTPUT_SHAPE = len(unique_breeds)

# Setup model URL from TensorFlow Hub
MODEL_URL = "https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/5"


INPUT_SHAPE

     [None, 224, 224, 3]
```

Now we've got our input, output and model ready, let's put them all together into Keras deep learning model.

let's create funtion which:

- Takes the input space and model we choose the parameter.
- Define layers in akeral model we've chosen as parameters.
- Define the layers in kerral (do this first, them this)
- Build the model
- Return the model

```
# Create a function which builds a Keras model
def create_model(input_shape=INPUT_SHAPE, output_shape=OUTPUT_SHAPE, model_url=MODEL_URL):
  print("Building model with:", MODEL_URL)

  # Setup the model layers
  model = tf.keras.Sequential([
    hub.KerasLayer(MODEL_URL), # Layer 1 (input layer)
    tf.keras.layers.Dense(units=output_shape,
                          activation="softmax") # Layer 2 (output layer)
  ])

  # Compile the model
  model.compile(
      loss=tf.keras.losses.CategoricalCrossentropy(),
      optimizer=tf.keras.optimizers.Adam(),
      metrics=["accuracy"]
  )

  # Build the model
  model.build(INPUT_SHAPE)

  return model


model = create_model()
model.summary()
```

```
WARNING:tensorflow:Please fix your imports. Module tensorflow.python.training.trackir
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classific
WARNING:tensorflow:From /usr/local/lib/python3.8/dist-packages/tensorflow/python/autc
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are us
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1001)              5432713

 dense (Dense)               (None, 120)               120240

=================================================================
Total params: 5,552,953
Trainable params: 120,240
Non-trainable params: 5,432,713
_________________________________________________________________
```

Creating callbacks

callbacks are helper funtion a model can use during training to do such things as save its progress, check its progress or stop training early if a model stop improving

we'll create calbacks one for TenserBored which helps track or models progress and another for early stoping which prevents our model from training for too long

TenserBoard callback

TO setup a tensorboard callback we need to set 3 things

1. Load the TensorBored notebook extension
2. create a tensorBored callback which is able to save logs to a directiory and pass it our model fit() funtion
3. visualise our models training logs with % tenserboared magic funtion

```
# Load TenserBored notebook extension
%load_ext tensorboard


import datetime

# Create a function to build a TensorBoard callback
def create_tensorboard_callback():
  # Create a log directory for storing TensorBoard logs
  logdir = os.path.join("drive/MyDrive/dog Vision/logs",
                        # Make it so the logs get tracked whenever we run an experiment
                        datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
  return tf.keras.callbacks.TensorBoard(logdir)
```

Early stopping callback early stopping helps our model from overfiting by stopping training if a certain evsuation metric stop improving

```
#create early stoping callback
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy",
                                                  patience=3)
```

Training amodel(on subset of data)

our first model is only going to train on 1000 images to make sure everything is working

```
NUM_EPOCHS = 100 #@param {type:"slider", min:10, max:100, step:10}
```

NUM_EPOCHS: 100

```
# check to make sure we're still running on a gpu
print("Gpu", "avilable (yes) "if tf.config.list_physical_devices("GPU") else "not avilable
```

```
Gpu avilable (yes)
```

Let's create a funtion which train a model.

- Create a model using create_model()
- setup a TensorBoard callback using create_tensorboard_callback()
- call the fit() funtion on our model passing it the training data, validation data, number of epochs to train for(NUM_EPOCHS) and the callbacks we'd like to use
- return the model

```
# Build a function to train and return a trained model
def train_model():
  """
  Trains a given model and returns the trained version.
  """
  # Create a model
  model = create_model()

  # Create new TensorBoard session everytime we train a model
  tensorboard = create_tensorboard_callback()

  # Fit the model to the data passing it the callbacks we created
  model.fit(x=train_data,
            epochs=NUM_EPOCHS,
            validation_data=val_data,
            validation_freq=1, # check validation metrics every epoch
            callbacks=[tensorboard, early_stopping])

  return model
```

```
# Fit the model to the data
model = train_model()
```

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classific
Epoch 1/100
25/25 [==============================] - 118s 3s/step - loss: 4.6866 - accuracy: 0.09
Epoch 2/100
25/25 [==============================] - 7s 272ms/step - loss: 1.6997 - accuracy: 0.6
Epoch 3/100
25/25 [==============================] - 4s 152ms/step - loss: 0.5775 - accuracy: 0.9
Epoch 4/100
25/25 [==============================] - 7s 267ms/step - loss: 0.2574 - accuracy: 0.9
Epoch 5/100
25/25 [==============================] - 6s 228ms/step - loss: 0.1487 - accuracy: 0.9
Epoch 6/100
25/25 [==============================] - 5s 190ms/step - loss: 0.1001 - accuracy: 0.9
Epoch 7/100
25/25 [==============================] - 3s 124ms/step - loss: 0.0757 - accuracy: 1.0
Epoch 8/100
25/25 [==============================] - 6s 233ms/step - loss: 0.0606 - accuracy: 1.0
Epoch 9/100
25/25 [==============================] - 4s 168ms/step - loss: 0.0494 - accuracy: 1.0
Epoch 10/100
25/25 [==============================] - 6s 256ms/step - loss: 0.0416 - accuracy: 1.0
Epoch 11/100
25/25 [==============================] - 6s 231ms/step - loss: 0.0358 - accuracy: 1.0
Epoch 12/100
25/25 [==============================] - 4s 151ms/step - loss: 0.0313 - accuracy: 1.0
Epoch 13/100
25/25 [==============================] - 5s 200ms/step - loss: 0.0276 - accuracy: 1.0
Epoch 14/100
25/25 [==============================] - 5s 186ms/step - loss: 0.0246 - accuracy: 1.0
Epoch 15/100
25/25 [==============================] - 4s 171ms/step - loss: 0.0221 - accuracy: 1.0
Epoch 16/100
25/25 [==============================] - 5s 187ms/step - loss: 0.0200 - accuracy: 1.0
Epoch 17/100
25/25 [==============================] - 3s 122ms/step - loss: 0.0182 - accuracy: 1.0
Epoch 18/100
25/25 [==============================] - 3s 118ms/step - loss: 0.0167 - accuracy: 1.0
Epoch 19/100
25/25 [==============================] - 4s 171ms/step - loss: 0.0153 - accuracy: 1.0
Epoch 20/100
25/25 [==============================] - 3s 119ms/step - loss: 0.0142 - accuracy: 1.0
Epoch 21/100
25/25 [==============================] - 4s 150ms/step - loss: 0.0131 - accuracy: 1.0
```

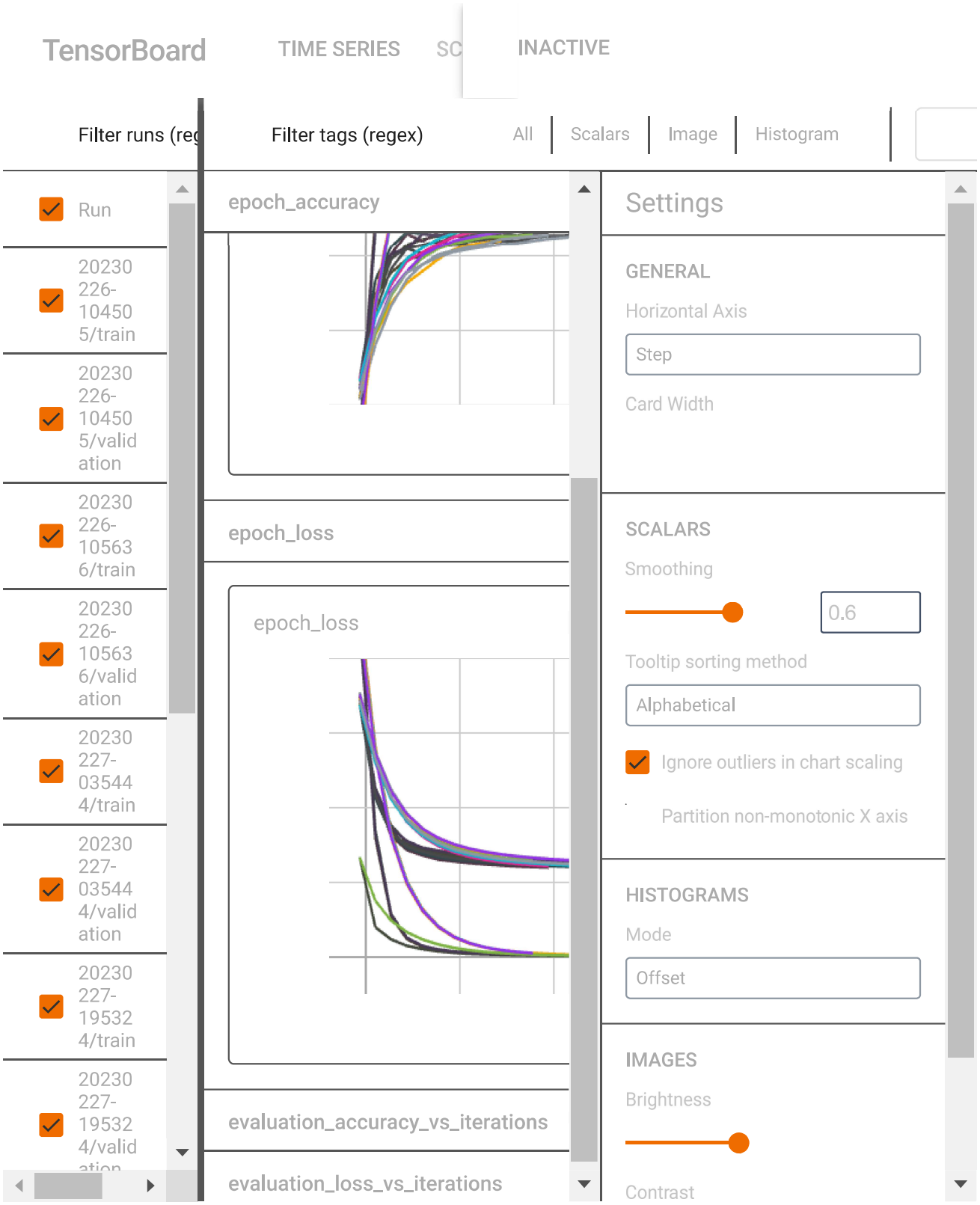
Question: it looks like our model is overfitting because it's performing bettter on the training dataset, now we have to prevent this

Note: overfiting is good things it mean our model is learning

Checking the TensorBoard logs

The tensorboard magic funtion (%tensorboard) will access the logs directory we created earlier and visualize its

```
%tensorboard --logdir drive/MyDrive/dog\ Vision/logs
```



Making and evaluation prediction using a trained mode

```
val_data
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 120), dtype=tf.bool, name=None))>
```

```
#Make prediction on the validation data( not used to train)
predictions = model.predict(val_data, verbose=1)
predictions
```

```
7/7 [==============================] - 2s 110ms/step
array([[5.5887608e-04, 3.7422622e-05, 4.3527267e-04, ..., 9.6607349e-05,
        3.7021960e-05, 3.6021501e-03],
       [5.8099249e-04, 4.4168308e-04, 7.1741315e-03, ..., 5.0537736e-04,
        8.6551317e-04, 1.1108863e-04],
       [1.1914876e-05, 9.2963433e-05, 9.3570234e-05, ..., 2.3514256e-05,
        1.3531989e-05, 1.6326634e-05],
       ...,
       [4.0605960e-07, 3.6040146e-06, 1.3801128e-05, ..., 5.9686784e-07,
        1.0119476e-05, 9.9214958e-06],
       [2.6371100e-03, 1.3476177e-04, 3.0641491e-05, ..., 7.4622651e-05,
        1.7876535e-05, 1.5269596e-02],
       [2.8074608e-04, 5.0363928e-05, 2.2356566e-04, ..., 4.5520077e-03,
        3.6310891e-04, 2.4030908e-04]], dtype=float32)
```

```
predictions.shape
```

```
(200, 120)
```

```
np.sum(predictions[0])
```

```
0.99999994
```

```
np.sum(predictions[1])
```

```
1.0000001
```

```
len(y_val)
```

```
200
```

```
len(unique_breeds)
```

```
120
```

```
# First prediction
index=42
print(predictions[index])
print(f"Max value (probability of prediction): {np.max(predictions[index])}")
print(f"Sum: {np.sum(predictions[index])}")
print(F"Max index: {np.argmax(predictions[index])}")
print(f"prediction label: {unique_breeds[np.argmax(predictions[index])]}")
```

```
[8.20327987e-05 7.89824207e-06 4.29773354e-05 1.12899543e-05
 9.72048147e-04 2.53537019e-05 3.81863574e-05 9.00838582e-04
 1.72189600e-03 2.42623035e-02 3.57387689e-05 1.81031760e-06
 5.89359697e-05 1.37524924e-03 3.67526984e-04 1.87534699e-03
 6.19850289e-06 2.47546122e-04 2.84035486e-05 4.87918842e-05
 1.28817583e-05 2.11111968e-04 5.46215488e-05 5.67920324e-06
 6.00469345e-03 8.14471787e-06 1.17331356e-05 1.22872416e-05
 6.51717564e-05 1.65542970e-05 1.74517390e-05 1.49477455e-05
 6.58916679e-06 9.84527196e-06 1.24053495e-05 9.49125115e-06
 9.83546197e-05 6.92596150e-05 1.60686504e-05 1.40489146e-01
 5.06710312e-05 6.46327953e-06 9.12093790e-04 4.55272044e-07
 4.70150844e-04 8.26575779e-05 3.48368158e-05 4.50116495e-05
 7.26891221e-06 2.77275667e-05 2.91929555e-05 4.15743780e-05
 1.00616577e-04 3.77456145e-03 2.28306635e-05 1.41870201e-04
 9.40239261e-06 1.53879355e-05 1.66538339e-05 2.50135072e-05
 1.56810820e-05 3.48350062e-04 1.69014754e-06 1.11326035e-05
 2.91798169e-05 2.54306851e-05 1.25151710e-05 4.28605999e-04
 6.31184012e-05 3.87946011e-06 2.00269205e-05 1.51602071e-05
 1.76989270e-05 8.36382242e-05 7.92730716e-05 2.70709006e-05
 2.17459619e-05 4.99418275e-06 6.03963144e-06 1.55487956e-04
 1.40801853e-06 2.62454523e-05 2.70585624e-05 4.40572505e-04
 3.51065828e-04 4.61304489e-06 3.45284170e-05 1.10182623e-06
 5.60571016e-06 3.80217971e-04 1.06786953e-04 7.95481128e-07
 5.76259568e-04 5.34872815e-05 1.09417324e-05 1.83027696e-05
 1.61075568e-05 1.08493168e-05 6.91577316e-06 2.64678729e-05
 6.76083482e-06 1.47478413e-05 4.15795730e-05 3.30648472e-05
 1.11242683e-04 3.45200206e-05 7.03897531e-05 1.38518760e-06
 8.68570132e-05 1.10214358e-04 1.60305826e-05 3.11137701e-04
 8.43346497e-05 8.09729338e-01 1.06234998e-04 1.60526673e-04
 2.05086108e-05 3.32247691e-06 4.71748179e-04 4.46096492e-05]
Max value (probability of prediction): 0.8097293376922607
Sum: 0.9999998807907104
Max index: 113
prediction label: walker_hound
```

Having the above functionality is great but we want to be able to do it scale and it would be even better if we could see the image the prediction is being made on

Note: prediction are also known as confidence levels

```
# Turn prediction probabilities into their respective label (easier to understand)
def get_pred_label(prediction_probabilities):
  """
  Turns an array of prediction probabilities into a label.
  """
  return unique_breeds[np.argmax(prediction_probabilities)]

# Get a predicted label based on an array of prediction probabilities
pred_label = get_pred_label(predictions[81])
pred_label
```

```
'dingo'
```

```
val_data
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 120), dtype=tf.bool, name=None))>
```

Since our validaton data is batch data set so we have to unbatch it to make on the prediction on the validation images and the compare thoes prediction to the validation labels(truth label)

```
#create a funtion to unbatch the dataset
# Create a function to unbatch a batch dataset
def unbatchify(data):
  """
  Takes a batched dataset of (image, label) Tensors and reutrns separate arrays
  of images and labels.
  """
  images = []
  labels = []
  # Loop through unbatched data
  for image, label in data.unbatch().as_numpy_iterator():
    images.append(image)
    labels.append(unique_breeds[np.argmax(label)])
  return images, labels

# Unbatchify the validation data
val_images, val_labels = unbatchify(val_data)
val_images[0], val_labels[0]
```

```
(array([[[0.29599646, 0.43284872, 0.3056691 ],
         [0.26635826, 0.32996926, 0.22846507],
         [0.31428418, 0.27701408, 0.22934894],
         ...,
         [0.77614343, 0.82320225, 0.8101595 ],
         [0.81291157, 0.8285351 , 0.8406944 ],
         [0.8209297 , 0.8263737 , 0.8423668 ]],

        [[0.2344871 , 0.31603682, 0.19543913],
         [0.3414841 , 0.36560842, 0.27241898],
         [0.45016077, 0.40117094, 0.33964607],
         ...,
         [0.7663987 , 0.8134138 , 0.81350833],
         [0.7304248 , 0.75012016, 0.76590735],
         [0.74518913, 0.76002574, 0.7830809 ]],

        [[0.30157745, 0.3082587 , 0.21018331],
         [0.2905954 , 0.27066195, 0.18401104],
         [0.4138316 , 0.36170745, 0.2964005 ],
         ...,
         [0.79871625, 0.8418535 , 0.8606443 ],
         [0.7957738 , 0.82859945, 0.8605655 ],
         [0.75181633, 0.77904975, 0.8155256 ]],

        ...,
```

```
              [[0.9746779 , 0.9878955 , 0.9342279 ],
               [0.99153054, 0.99772066, 0.9427856 ],
               [0.98925114, 0.9792082 , 0.9137934 ],
               ...,
               [0.0987601 , 0.0987601 , 0.0987601 ],
               [0.05703771, 0.05703771, 0.05703771],
               [0.03600177, 0.03600177, 0.03600177]],

              [[0.98197854, 0.9820659 , 0.9379411 ],
               [0.9811992 , 0.97015417, 0.9125648 ],
               [0.9722316 , 0.93666023, 0.8697186 ],
               ...,
               [0.09682598, 0.09682598, 0.09682598],
               [0.07196062, 0.07196062, 0.07196062],
               [0.0361607 , 0.0361607 , 0.0361607 ]],

              [[0.97279435, 0.9545954 , 0.92389745],
               [0.963602  , 0.93199134, 0.88407487],
               [0.9627158 , 0.91253304, 0.8460338 ],
               ...,
               [0.08394483, 0.08394483, 0.08394483],
               [0.0886985 , 0.0886985 , 0.0886985 ],
               [0.04514172, 0.04514172, 0.04514172]]], dtype=float32), 'cairn')
```

```
get_pred_label(val_labels[0])
```

```
     'affenpinscher'
```

Now we've got ways to get

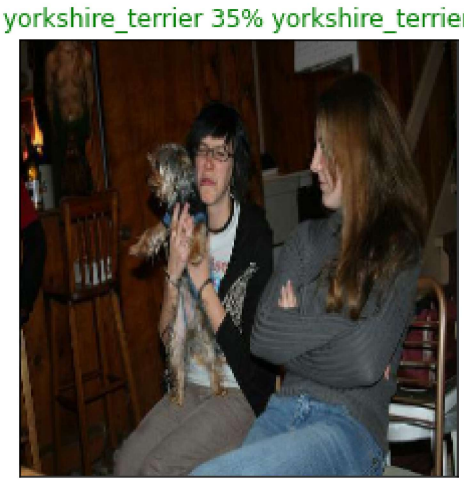
- prediction label
- validation label
- validation images Let make some funtio to visualise it

we crate funtion which:

- take an array prediction probablites, an array of truth labels and an array of images and iterator
- convert the prediction probablites to a predicted label
- plot the prediction probablites, the truth label and the target image on a single plot

```
def plot_pred(prediction_probabilities, labels, images, n=1):
  """
  View the prediction, ground truth and image for sample n
  """
  pred_prob, true_label, image = prediction_probabilities[n], labels[n], images[n]

  # Get the pred label
  pred_label = get_pred_label(pred_prob)

  # Plot image & remove ticks
  plt.imshow(image)
  plt.xticks([])
```

```
  plt.yticks([])

  # Change the colour of the title depending on if the prediction is right or wrong
  if pred_label == true_label:
    color = "green"
  else:
    color = "red"

  # Change plot title to be predicted, probability of prediction and truth label
  plt.title("{} {:2.0f}% {}".format(pred_label,
                                    np.max(pred_prob)*100,
                                    true_label),
                                    color=color)


plot_pred(prediction_probabilities=predictions,
          labels=val_labels,
          images=val_images,
          n=10)
```



Now we've got one function to visualize our models top prediction, let's make another to view our models top 10 predictions.

This function will:

- Take an input of prediction probabilities array and a ground truth array and an integer
- Find the prediction using get_pred_label()
- Find the top 10: Prediction probabilities indexes

Prediction probabilities values

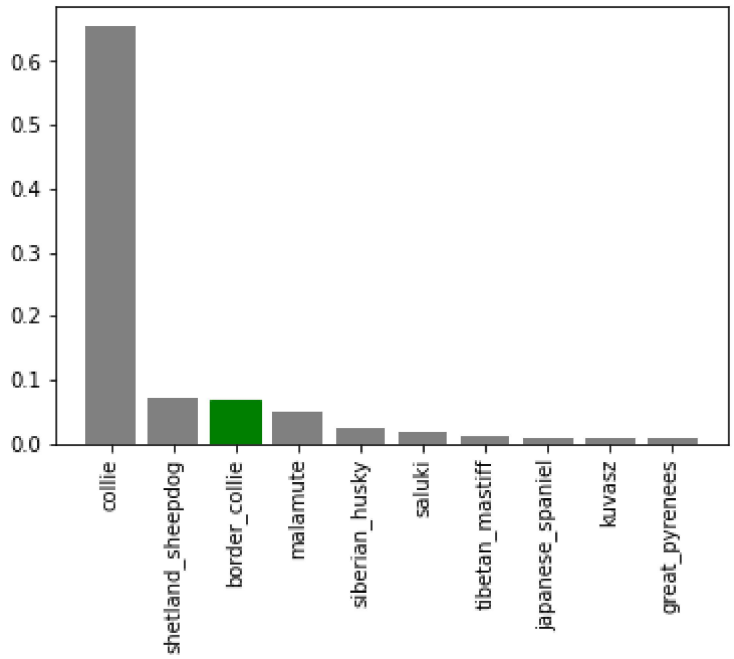
Prediction labels

- Plot the top 10 prediction probability values and labels, coloring the true label green ✅

```
def plot_pred_conf(prediction_probabilities, labels, n=1):
  """
  plus the top 10 highest preddiction confidence along with the truth label for sample n
  """
```

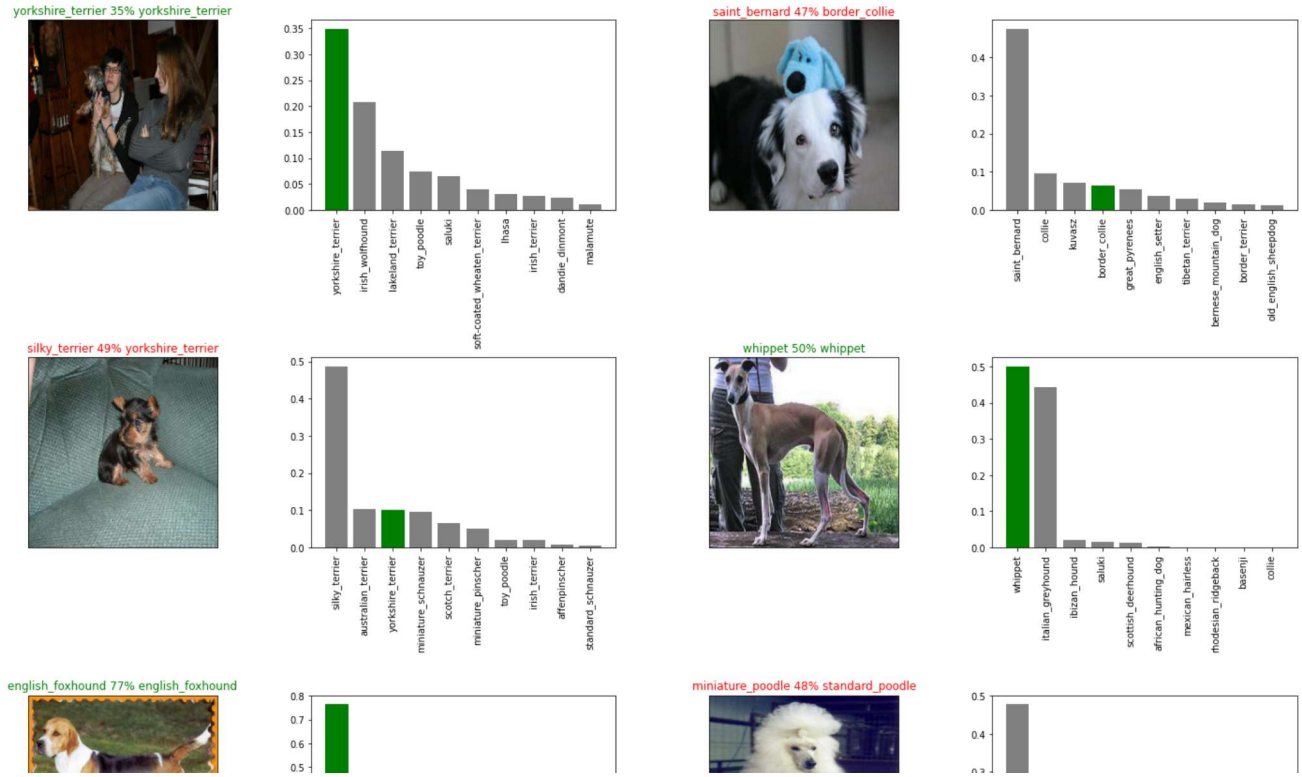
```
  pred_prob, true_label = prediction_probabilities[n], labels[n]

  #Get the prediction label
  pred_label = get_pred_label(pred_prob)

  # find the top 10 prediction confidance indexes
  top_10_pred_indexes = pred_prob.argsort()[-10:][::-1]
  # find the top 10 prediction confidance values
  top_10_pred_values = pred_prob[top_10_pred_indexes]
  # find the top 10 prediction labels
  top_10_pred_labels = unique_breeds[top_10_pred_indexes]

  # set up  the plot
  top_plot = plt.bar(np.arange(len(top_10_pred_labels)),
                     top_10_pred_values,
                     color="grey")
  plt.xticks(np.arange(len(top_10_pred_labels)),
             labels=top_10_pred_labels,
             rotation="vertical")

  #change color of true label
  if np.isin(true_label, top_10_pred_labels):
    top_plot[np.argmax(top_10_pred_labels == true_label)].set_color("green")
  else:
     pass
```

```
plot_pred_conf(prediction_probabilities=predictions,
               labels=val_labels,
               n=9)
```



Now we've got some funtion to help us visualize our prediction and evaluate our model let's check out few

```
# Let's check out a few predictions and their different values
i_multiplier = 10
num_rows = 3
num_cols = 2
num_images = num_rows*num_cols
plt.figure(figsize=(10*num_cols, 5*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_pred(prediction_probabilities=predictions,
            labels=val_labels,
            images=val_images,
            n=i+i_multiplier)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_pred_conf(prediction_probabilities=predictions,
                 labels=val_labels,
                 n=i+i_multiplier)
plt.tight_layout(h_pad=1.0)
plt.show()
```



Saving and reloading a trained model

```
# create a funtion to save a model
def save_model(model, suffix=None):
  """
  Save a given model in a models directory and appends a sufix (string).
  """
  # create a model directory pathname with current time
  modeldir = os.path.join("drive/MyDrive/dog Vision/models",
                          datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
  model_path = modeldir + "_" + suffix + ".h5" # save formate of model
  print(f"Saving model to: {model_path}...")
  model.save(model_path)
  return model_path


# Create a function to load a trained model
def load_model(model_path):
  """
  Loads a saved model from a specified path.
  """
  print(f"Loading saved model from: {model_path}")
  model = tf.keras.models.load_model(model_path,
                                     custom_objects={"KerasLayer":hub.KerasLayer})
  return model


#save our model trained on 1000 images
save_model(model, suffix="1000-images-mobilenetv2-Adam")
```

```
Saving model to: drive/MyDrive/dog Vision/models/20230301-190651_1000-images-mobilene
```

```
# load a trained model
loaded_1000_image_model = load_model('drive/MyDrive/dog Vision/models/20230228-205458_1000
```

```
Loading saved model from: drive/MyDrive/dog Vision/models/20230228-205458_1000-images
```

```
# Evaluate the pre-saved model
model.evaluate(val_data)
```

```
7/7 [==============================] - 1s 83ms/step - loss: 1.2047 - accuracy: 0.6756
[1.2046617269515991, 0.675000011920929]
```

```
#evaluate the loaded model
loaded_1000_image_model.evaluate(val_data)
```

```
7/7 [==============================] - 2s 130ms/step - loss: 1.2663 - accuracy: 0.685
[1.2662937641143799, 0.6850000023841858]
```

Trainig a big dog model on full data

```
len(X), len(y)
```

```
(10222, 10222)
```

```
X[:10]
```

```
['drive/MyDrive/dog Vision/train/000bec180eb18c7604dcecc8fe0dba07.jpg',
 'drive/MyDrive/dog Vision/train/001513dfcb2ffafc82cccf4d8bbaba97.jpg',
 'drive/MyDrive/dog Vision/train/001cdf01b096e06d78e9e5112d419397.jpg',
 'drive/MyDrive/dog Vision/train/00214f311d5d2247d5dfe4fe24b2303d.jpg',
 'drive/MyDrive/dog Vision/train/0021f9ceb3235effd7fcde7f7538ed62.jpg',
 'drive/MyDrive/dog Vision/train/002211c81b498ef88e1b40b9abf84e1d.jpg',
 'drive/MyDrive/dog Vision/train/00290d3e1fdd27226ba27a8ce248ce85.jpg',
 'drive/MyDrive/dog Vision/train/002a283a315af96eaea0e28e7163b21b.jpg',
 'drive/MyDrive/dog Vision/train/003df8b8a8b05244b1d920bb6cf451f9.jpg',
 'drive/MyDrive/dog Vision/train/0042188c895a2f14ef64a918ed9c7b64.jpg']
```

```
len(X_train)
```

```
800
```

```
# create a data batch with the full data set
full_data = create_data_batches(X, y)
```

```
Creating training data batches...
```

```
full_data
```

```
<BatchDataset element_spec=(TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32,
name=None), TensorSpec(shape=(None, 120), dtype=tf.bool, name=None))>
```

```
full_model = create_model()
```

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classific
```

```
#create full modael  callbacks
full_model_tensorboard = create_tensorboard_callback()
# No validation set when training on all the data so we can't moniter validation accuracy
full_model_early_stopping = tf.keras.callbacks.EarlyStopping(monitor="accuracy",
                                                             patience=3)
```

this cell may be mone more time to complete

```
# Fit the full model to the full data
full_model.fit(x=full_data,
               epochs=NUM_EPOCHS,
               callbacks=[full_model_tensorboard, full_model_early_stopping])
```

```
Epoch 1/100
320/320 [==============================] - 44s 124ms/step - loss: 1.3366 - accuracy:
Epoch 2/100
320/320 [==============================] - 33s 103ms/step - loss: 0.3972 - accuracy:
Epoch 3/100
320/320 [==============================] - 52s 160ms/step - loss: 0.2387 - accuracy:
Epoch 4/100
320/320 [==============================] - 34s 105ms/step - loss: 0.1558 - accuracy:
Epoch 5/100
320/320 [==============================] - 45s 140ms/step - loss: 0.1073 - accuracy:
Epoch 6/100
320/320 [==============================] - 44s 137ms/step - loss: 0.0770 - accuracy:
Epoch 7/100
320/320 [==============================] - 38s 119ms/step - loss: 0.0591 - accuracy:
Epoch 8/100
320/320 [==============================] - 39s 121ms/step - loss: 0.0462 - accuracy:
Epoch 9/100
320/320 [==============================] - 37s 114ms/step - loss: 0.0381 - accuracy:
Epoch 10/100
320/320 [==============================] - 44s 139ms/step - loss: 0.0309 - accuracy:
Epoch 11/100
320/320 [==============================] - 43s 135ms/step - loss: 0.0264 - accuracy:
Epoch 12/100
320/320 [==============================] - 40s 124ms/step - loss: 0.0219 - accuracy:
Epoch 13/100
320/320 [==============================] - 36s 112ms/step - loss: 0.0211 - accuracy:
Epoch 14/100
320/320 [==============================] - 33s 104ms/step - loss: 0.0177 - accuracy:
Epoch 15/100
320/320 [==============================] - 34s 105ms/step - loss: 0.0183 - accuracy:
<keras.callbacks.History at 0x7f4af06a75b0>
```

```
save_model(full_model, suffix ="full-image-set-mobilenetv2-Adam")
```

```
Saving model to: drive/MyDrive/dog Vision/models/20230301-191659_full-image-set-mobil
'drive/MyDrive/dog Vision/models/20230301-191659_full-image-set-mobilenetv2-Adam.h5'
```

```
#load the full model
loaded_full_model = load_model('drive/MyDrive/dog Vision/models/20230228-212927_full-image
```

```
Loading saved model from: drive/MyDrive/dog Vision/models/20230228-212927_full-image-
```

Making prediction on test dataset

since our model has been trained on images in the form of Tensor batches to make predictions on the test data, we'll have to get it into the same formate as we created data_batches() earlier which can take a list of filename as input and convert into tensor batches To make predictions on the test data we'll:

- Get the image filenamess
- convert filenames into test data batches using create databatches() and setting test_data parameter True as test data don't have labels
- Make prediction array by passing the test batches to the predict() method called on our model

```
# load the image filenames
test_path ="drive/MyDrive/dog Vision/test/"
test_filenames = [test_path + fname for fname in os.listdir(test_path)]
test_filenames[:10]
```

```
['drive/MyDrive/dog Vision/test/e427b9e1ab1b7f09cfb02ac073f56f2d.jpg',
 'drive/MyDrive/dog Vision/test/e421e13d0d4c977ab89fa9965221809c.jpg',
 'drive/MyDrive/dog Vision/test/e30a97eb637239e3b15c903529e04800.jpg',
 'drive/MyDrive/dog Vision/test/ddc0134def0da0472e55271da8b57c44.jpg',
 'drive/MyDrive/dog Vision/test/e411a1b3681604f6321af7cb8e8f2de7.jpg',
 'drive/MyDrive/dog Vision/test/e334f758c7944df19c98d49498d28c64.jpg',
 'drive/MyDrive/dog Vision/test/deaf748bed865ec76758f086db04bfac.jpg',
 'drive/MyDrive/dog Vision/test/e103930a8a4416f076c81a51907a7512.jpg',
 'drive/MyDrive/dog Vision/test/e278b2f68fa5d146e0bf06eee5faa0e3.jpg',
 'drive/MyDrive/dog Vision/test/e71d9f32ea6eb1c2d944e8f4e811c209.jpg']
```

```
len(test_filenames)
```

```
10357
```

```
#create test data batch
test_data = create_data_batches(test_filenames, test_data=True)
```

```
Creating test data batches...
```

```
test_data
```

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

```
#Mkae prediction on test data batch using the loded full model
test_predictions = loaded_full_model.predict(test_data,
                                             verbose=1)
```

```
324/324 [==============================] - 148s 445ms/step
```

```
# save prediction(numpy array) to csv file for accesss later
#np.savetxt("drive/MyDrive/dog Vision/preds_array.csv", test_predictions, delimiter=",")
```

```
# load predictions(numpy array) from csv file
test_predictions=np.loadtxt("drive/MyDrive/dog Vision/preds_array.csv", delimiter=",")
```

```
test_predictions[:10]
```

```
array([[5.04669773e-09, 8.73544170e-09, 6.07846107e-10, ...,
        7.51192372e-11, 9.99876499e-01, 4.93539787e-09],
       [2.35612148e-13, 6.02668249e-10, 1.78500215e-07, ...,
        7.32975947e-09, 1.21073871e-08, 1.12957907e-12],
       [1.84990611e-04, 1.08305764e-09, 1.03033591e-08, ...,
        6.73671394e-08, 1.00510517e-10, 1.62898202e-03],
       ...,
       [1.66965686e-09, 1.08444431e-06, 1.84264370e-09, ...,
        4.07181346e-08, 8.86748530e-05, 1.66152798e-08],
       [1.97171513e-02, 8.37505067e-05, 1.91008009e-08, ...,
        1.03585848e-10, 1.44219886e-10, 8.52987289e-01],
       [7.16603554e-10, 1.41188269e-04, 4.35292313e-05, ...,
        2.85418741e-06, 5.09059755e-04, 5.79477230e-04]])
```

```
test_predictions.shape
```

```
(10357, 120)
```

Making prerdictions on custom images we'll

- get the filepath of our own images
- Turn the filepath into data batches using create_data_batches() and sinve our imagess won't have labels we set the test_data parameter to true
- pass the custom image data batch to our model's predict() method
- conver the preediction output probbalities to prediction lebls
- compare the predicted labels to the custom images

```
#get custom image filepaths
custom_path = "drive/MyDrive/dog Vision/my-dog-photo/"
custom_image_paths = [custom_path + fname for fname in os.listdir(custom_path)]


custom_image_paths

     ['drive/MyDrive/dog Vision/my-dog-photo/desi1.jpeg',
      'drive/MyDrive/dog Vision/my-dog-photo/german.jpeg']


#turn custom image into batch datasets
custom_data= create_data_batches(custom_image_paths, test_data=True)
custom_data

     Creating test data batches...
     <BatchDataset element_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32,
     name=None)>


# make prediction on the custom data
custom_preds = loaded_full_model.predict(custom_data)

     1/1 [==============================] - 0s 39ms/step


custom_preds.shape

     (2, 120)


 # Get custom image prediction labels
custom_pred_labels = [get_pred_label(custom_preds[i]) for i in range (len(custom_preds))]
custom_pred_labels

     ['dingo', 'tibetan_mastiff']


# get custom images (our unbatchify() ) funtion won't work since are't label we have to fi
custom_images=[]
# loop throug unbatched data
for image in custom_data.unbatch().as_numpy_iterator():
  custom_images.append(image)


import enum
# check custom image prediction
plt.figure (figsize=(10, 10))
for i, image in enumerate(custom_images):
  plt.subplot(1, 3, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.title(custom_pred_labels[i])
  plt.imshow(image)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-5126b9ad9d64> in <module>
      1 import enum
      2 # check custom image prediction
----> 3 plt.figure (figsize=(10, 10))
      4 for i, image in enumerate(custom_images):
      5   plt.subplot(1, 3, i+1)

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