🐶 End-to-end Multi-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.

When I'm sitting at the park and I take a photo of a dog, I want to know what breed of dog it is.

2. Data

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

https://www.kaggle.com/competitions/dog-breed-identification/data?select=train

3. Evaluation

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

https://www.kaggle.com/competitions/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).

```
# Unzip the uploaded data into Google Drive
# !unzip "drive/MyDrive/Dog Vision/dog-breed-identification.zip" -d "drive/MyDrive/Dog Vision/"
```

Get our workspace ready

- Import TensorFlow 2.x ✅
- Import TensorFlow Hub ✅
- Make sure we're using a GPU ✅

```
# Import necessary tools
# Import TensorFlow into Colab
import tensorflow as tf
import tensorflow_hub as hub
print("TF version", tf.__version__)
print("TF Hub version", hub.__version__)

# Check for GPU availability
print("GPU","available (YESSSS!!!!!)" if tf.config.list_physical_devices("GPU") else "not available :(")
```

```
TF version 2.12.0
TF Hub version 0.13.0
GPU available (YESSSS!!!!!)
```

Getting our data ready (turning into Tensors)

With all machine learning models, our data has to be in numerical format. So that's what we'll be doing first. Turning our images into Tensors (numerical representation).

Let's start by accessing our data and checking out the labels.

```
# Checkout the labels of our data
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
```

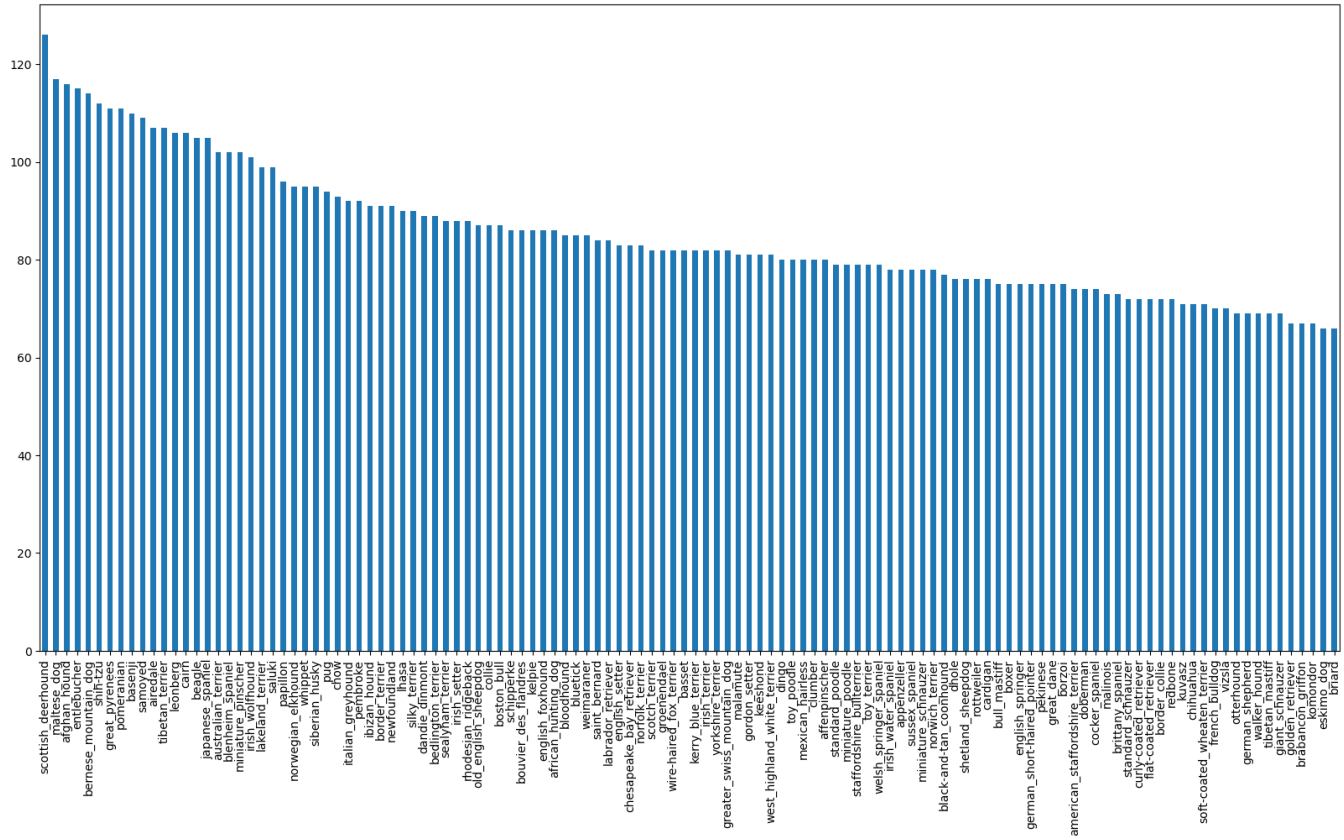
```
freq                                1                126
                                 id            breed
0  000bec180eb18c7604dcecc8fe0dba07      boston_bull
1  001513dfcb2ffafc82cccf4d8bbaba97            dingo
2  001cdf01b096e06d78e9e5112d419397         pekinese
3  00214f311d5d2247d5dfe4fe24b2303d          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 breed?
labels_csv["breed"].value_counts().plot.bar(figsize=(20,10))
```

```
<Axes: >
```




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

```
82.0
```

```
# Let's view an image
from IPython.display import Image
Image("drive/MyDrive/Dog Vision/train/000bec180eb18c7604dcecc8fe0dba07.jpg")
```

Getting images and their labels

Let's get a list of all our images file pathnames.

```
labels_csv.head()
```

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

```
# Create pathnames from image ID's
filenames = ["drive/MyDrive/Dog Vision/train/"+fname+".jpg" for fname in labels_csv["id"]]

# 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']
```

```
import os
os.listdir("drive/MyDrive/Dog Vision/train/")[:10]
```

```
['e19ef13146562537f75ddf4c447d9697.jpg',
 'df3ba5eee0c008a3284158bb89350673.jpg',
 'e00443152fb5951922730b21ba08f8f5.jpg',
 'de0966a23cecbb9f481714b059c8bcd3.jpg',
 'df775558457f8fc42f50a1b944d89328.jpg',
 'e201fdbcd92e3d0a10d612d5e0b77b35.jpg',
 'e4ca855e09eabf518ed39a0a545da981.jpg',
 'dd68215c1ac5b073fc42202f229bc283.jpg',
 'e03f2a3e636ea5900f5c57ca7c7af68c.jpg',
 'e7997562415c62141d022cfa8cae1c60.jpg']
```

```
#  Check whether 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 actual amount of files!!! Proceed.")
else:
  print("Filenames do not match actual amount of files, check the target directory.")
```

```
Filenames match actual amount of files!!! Proceed.
```

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


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

```
'tibetan_mastiff'
```

Since we've now got our training image filepaths in a list, let's prepare our labels.

```
import numpy as np
labels = labels_csv["breed"].to_numpy()
# labels = np.array(labels) # does same thing as above
labels
```

```
array(['boston_bull', 'dingo', 'pekinese', ..., 'airedale',
       'miniature_pinscher', 'chesapeake_bay_retriever'], 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("Number of labels does not matches number of filenames, check data directories!")
```

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

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

```
120
```

```
# 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, 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]),
 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])]
```

```
len(boolean_labels)
```

```
10222
```

```
print(labels[0]) # original label
print(np.where(unique_breeds == labels[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
(array([19]),)
19
[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]
```

```
print(labels[5])
print(boolean_labels[5].astype(int))
```

```
bedlington_terrier
[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 0 0 0 0 0]
```

```
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']
```

Creating our own validation set

Since the dataset from Kaggle doesn't come with a validation set, we're going to create our own.

```
# Setup X & y variables
X=filenames
y=boolean_labels
```

```
len(filenames)
```

```
10222
```

We're going to start off experimenting with ~1000 images and increase as needed.

```
# Set number of images to use for experimenting
NUM_IMAGES = 1000 #@param{type:"slider",min:1000, max:10000,step:1000}
```

NUM_IMAGES: 1000

```
# Let's split our data into train and validation sets
from sklearn.model_selection import train_test_split

# Split them into training and validation of total size 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)
```

```
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])])
```

Preprocessing Images (turning images into Tensors)

To preprocess our images which does a few things:

1. Take an image filepath as input.
2. Use TensorFlow to read the file and save it to a variable, `image`.
3. Turn our `image` (a jpg) into Tensors.
4. Normalize our image (convert color channel values from 0-255 to 0-1).
5. Resize the `image` to be a shape of(224,224)
6. Return the modified `image`

Before we do, let's see what importing an image looks like.

```
# Convert image to NumPy array
from matplotlib.pyplot import imread
image = imread(filenames[42])
image.shape
```

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

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

```
(255, 0)
```

```
image[:2]
```

```
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)
```

```
# Turn image into a 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)>
```

Now we've seen what an image looks like as a Tensor, let's make a function to preprocess them.

1. Take an image filepath as input.
2. Use TensorFlow to read the file and save it to a variable, `image`.
3. Turn our `image` (a jpg) into Tensors.
4. Normalize our image (convert color channel values from 0-255 to 0-1).
5. Resize the `image` to be a shape of(224,224)
6. Return the modified `image`

```
# Define image size
IMG_SIZE = 224

# Create a function for preprocessing images
def process_image(image_path):
  """
  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?

Let's say you're trying to process 10,000+ images in one go... they all might not fit into memory.

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

In order to use TensorFlow effectively, we need our data in the form of Tensor tuples 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 associated label,
  processes the image and returns a tuple of(image, label).
  """
  image = process_image(image_path)
  return image,label
```

```
# the above function works like below:
(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)>,
 <tf.Tensor: shape=(120,), dtype=bool, numpy=
 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,
         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,
```

Now we've got a way to turn our data into tuples of Tensors in the form: `(image,label)`, let's make a function to turn all of our data (`X` & `y`) into batches!

```
# 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=False):
  """
  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 inpuy (no labels).
  """
  # If the data is a test dataset, we probably don't have labels
  if test_data:
    print("Creating test data batches...")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X))) # only filepaths (no labels)
    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 than shuffling images
    data = data.shuffle(buffer_size=len(X))

    # Create (image,label) tuples (this also turns the image path into a preprocessed image)
    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)))
```

Visualising Data Batches

Our data is now in batches, however, these can be a little hard to understand/comprehend, lett's visualize them!

```
import matplotlib.pyplot as plt

# Create a function for viewing images in a data batch
def show_25_images(images,labels):
  """
  Display 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))>
```

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


```
# Now let's visualize 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/5

```
# 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"
```

Now we've got our inputs, outputs and model ready to go. Let's put them together into a Keras deep learning model!

Knowing this, let's create a function which:

- Takes the input shape, output shape and the model we've chosen as parameters.
- Defines the layers in a Keras model in sequential fashion(do this first, then this, tthen that).
- Compiles the model (says it should be evaluated and improved).
- Builds the model (tells the model the input shape it'll be getting).
- Returns the model.

All of these steps can be found here: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/5

```
# 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()
```

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/5
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer_3 (KerasLayer)  (None, 1001)              5432713

 dense_3 (Dense)             (None, 120)               120240

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

Creating callbacks

Callbacks are helper functions a model can use during training to do such things as save its progress, check its progress or stop training early if a model stops improving.

We'll create two callbacks, one for TensorBoard which helps track our models progress and another for early stopping which prevents our model from training for too long.

TensorBoard Callback

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/TensorBoard

To setup a TensorBoard callback, we need to do 3 things:

1. Load the TensorBoard notebook extension ✅
2. Create a TensorBoard callback which is able to save logs to a directory and pass it to our model's `fit()` function. ✅
3. Visualize our models trainings logs with the `%tensorboard` magic function (we'll do this after model training)

```
# Load TensorBoard notebook extension
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_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 stop our model from overfitting by stopping training if a certain evaluation metric stops improving.

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

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

Training a model (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","available(YES!!!)" if tf.config.list_physical_devices("GPU") else "not available :(")
```

```
GPU available(YES!!!)
```

Let's create a function which trains a model.

- Create a model using `create_model()`
- Setup a TensorBoard callback using `create_tensorboard_callback()`
- Call the `fit()` function 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,
            callbacks=[tensorboard,early_stopping])
  # Return the fitted model
  return model
```

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

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/5
Epoch 1/100
25/25 [==============================] - 13s 279ms/step - loss: 4.5528 - accuracy: 0.1112 - val_loss: 3.4641 - val_accuracy: 0.1950
Epoch 2/100
25/25 [==============================] - 4s 176ms/step - loss: 1.6229 - accuracy: 0.6850 - val_loss: 2.1773 - val_accuracy: 0.4900
Epoch 3/100
25/25 [==============================] - 4s 148ms/step - loss: 0.5626 - accuracy: 0.9362 - val_loss: 1.6386 - val_accuracy: 0.5850
Epoch 4/100
25/25 [==============================] - 3s 119ms/step - loss: 0.2580 - accuracy: 0.9875 - val_loss: 1.4758 - val_accuracy: 0.6300
Epoch 5/100
25/25 [==============================] - 4s 155ms/step - loss: 0.1488 - accuracy: 0.9975 - val_loss: 1.4054 - val_accuracy: 0.6800
Epoch 6/100
25/25 [==============================] - 4s 161ms/step - loss: 0.1005 - accuracy: 0.9987 - val_loss: 1.3537 - val_accuracy: 0.6800
Epoch 7/100
25/25 [==============================] - 3s 121ms/step - loss: 0.0761 - accuracy: 1.0000 - val_loss: 1.3240 - val_accuracy: 0.6750
Epoch 8/100
25/25 [==============================] - 4s 150ms/step - loss: 0.0600 - accuracy: 1.0000 - val_loss: 1.3006 - val_accuracy: 0.6800
```

Checking the TensorBoard logs

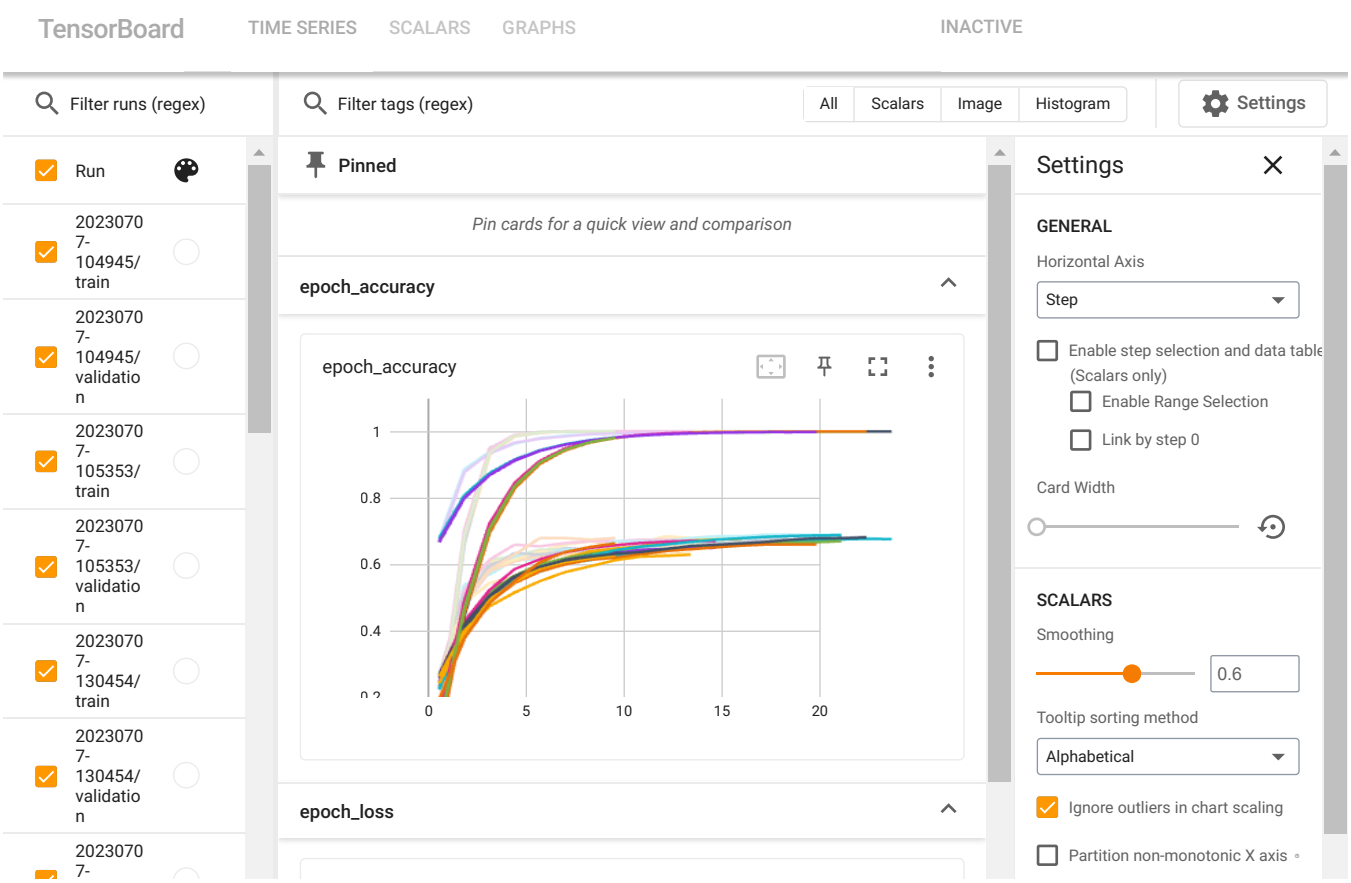
The TensorBoard magic function (`%tensorboard`) will access the logs directory we created earlier and visualize its contents.

```
#!pip install tensorboard
```

```
!rm -rf ./logs/
```

```
%load_ext tensorboard
%tensorboard --logdir drive/MyDrive/Dog\ Vision/logs --port 6007
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6007 (pid 13463), started 0:14:51 ago. (Use '!kill 13463' to kill it.)
```



Making and evaluating predictions using a trained model

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

```
7/7 [==============================] - 2s 95ms/step
array([[5.25670010e-04, 2.54838178e-05, 1.32539636e-03, ...,
        1.94138120e-04, 8.01594579e-05, 1.73634046e-03],
       [4.42813337e-03, 3.76129232e-04, 5.65604819e-03, ...,
        9.81126650e-05, 5.78815071e-03, 1.96819819e-04],
       [1.78889124e-04, 1.55279035e-04, 2.99207022e-05, ...,
        3.60601553e-05, 7.34679925e-05, 2.36821958e-04],
       ...,
       [7.13121626e-05, 5.27951415e-05, 6.38009305e-06, ...,
        6.47461347e-05, 1.00335295e-04, 5.54120015e-05],
       [1.93668739e-03, 4.62239521e-04, 1.36376271e-04, ...,
        6.55420372e-05, 7.82154093e-05, 1.13340691e-02],
       [1.07645465e-04, 2.02113297e-05, 2.03000661e-03, ...,
        1.97442388e-03, 1.55928945e-02, 2.38977751e-04]], dtype=float32)
```

```
predictions.shape
```

```
(200, 120)
```

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

```
0.99999994
```

```
# 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"Predicted label: {unique_breeds[np.argmax(predictions[index])]}")
```

```
[6.18372287e-04 1.22797239e-04 5.54351573e-05 7.85287993e-05
 2.92631797e-03 9.25669519e-05 1.54002992e-04 2.46337499e-04
 7.51707656e-03 3.51145454e-02 5.07769109e-05 1.58185885e-05
 1.59818315e-04 3.03746457e-03 1.27716770e-03 3.91393621e-03
 2.18784007e-05 1.91783794e-04 1.53024172e-04 1.31573970e-03
 7.79387847e-05 2.49024801e-04 1.78343871e-05 9.03472246e-05
 7.82590546e-03 6.67890825e-04 2.19289839e-04 2.14060899e-04
 9.96639355e-05 7.09833621e-05 2.89672753e-04 7.07706422e-05
 9.15579949e-05 4.43182762e-05 1.64298835e-05 2.23267289e-05
 9.46778397e-04 1.62940458e-04 1.24326412e-04 6.18459061e-02
 1.50676889e-04 1.79223462e-05 1.28578162e-02 3.41330124e-05
 6.75544317e-04 4.41540906e-05 4.26224462e-04 9.97158422e-05
 1.47678715e-04 7.66529411e-04 3.98800403e-05 8.13453516e-05
 2.43648974e-04 1.42821996e-03 2.46722011e-05 3.20261053e-04
 2.06526587e-04 1.23561520e-04 2.33531682e-04 2.37439090e-05
 6.75528136e-05 2.58558866e-04 2.40561458e-05 1.56076974e-04
 2.32573540e-04 5.71789351e-05 9.42602564e-05 5.27703844e-04
 2.74603692e-04 4.71224688e-04 7.26936487e-05 1.53683752e-04
 1.82344083e-04 1.33851834e-03 2.12542091e-05 2.36882377e-04
 7.75454027e-05 4.09641434e-05 4.64158875e-05 1.06005435e-04
 1.29767723e-05 9.87679887e-05 1.48533669e-04 3.33720789e-04
 4.67601552e-04 1.93834756e-04 2.12358820e-04 2.20853804e-06
 8.62761299e-05 2.37442716e-03 4.03905811e-04 5.15347347e-05
 4.59475938e-04 1.53148692e-04 2.31439681e-05 4.82883654e-04
 9.26860776e-06 2.52632835e-05 6.61999366e-05 8.37128173e-05
 3.18184088e-04 5.53391801e-05 1.52832974e-04 5.25709147e-05
 2.01610674e-04 1.99916321e-05 6.96903298e-05 1.51722023e-04
 4.71722888e-05 2.26531731e-04 8.33481317e-05 8.14953644e-04
 2.49439821e-04 8.33482981e-01 3.95741459e-04 1.80216541e-03
 1.62671538e-04 2.72471989e-05 3.08676460e-03 3.12467455e-04]
Max value (probability of prediction): 0.8334829807281494
Sum: 1.0
Max index: 113
Predicted label: walker_hound
```

```
unique_breeds[113]
```

```
'walker_hound'
```

Having the above functionality is great but we want to be able to do it at scale.

And it would be even better if we could see the image the prediction is being made on!

Note: Prediction probabilities are also known as confidence levels

```
# Turn predictions into their respective label (easier to understand)
def get_pred_label(prediction_probabilities):
  """
  Turns an array of predictions 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[0])
pred_label
```

```
'cairn'
```

Now since our validation data is still in a batch dataset, we'll have to unbatchify it to make predictions on the validation images and then compare those predictions to the validation labels(truth labels).

```
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))>
```

```
# Create a function to unbatch a batch dataset
def unbatchify(data):
  """
  Takes a batched dataset of (image,label) Tensors and returns separate arrays
  of iamges 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 labels
- Validation labels (truth labels)
- Validation images

Let's make some function to make these all a bit more visualize.

We'll create a function which:

- Takes an array of prediction probabilities, an array of truth tables and an array of images and integers. ✅
- Convert the prediction probabilities to a predicted label. ✅
- Plot the predicted label, its predicted probability, the truth table 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
```

```
  # 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(predictions,val_labels,val_images,36)
```

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

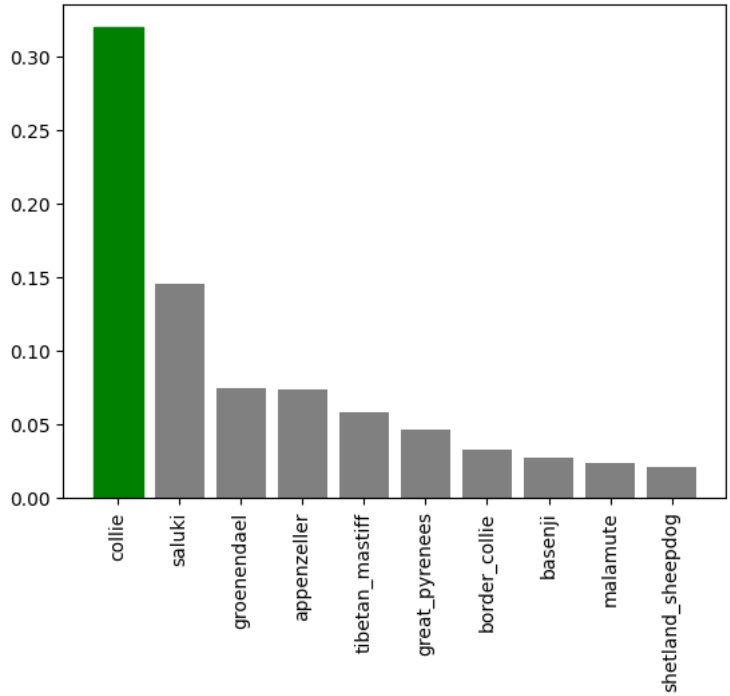
This function will:

- Take an input of prediction probabilities array and a ground truth array and an integer ✅
- Find the predicted label 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 truue label green

```
def plot_pred_conf(prediction_probabilities, labels, n=1):
  """
  Plus the top 10 highest prediction confidences along with the truth label for sample n.
  """
  pred_prob,true_label = prediction_probabilities[n], labels[n]

  # Get the predicted label
  pred_label = get_pred_label(pred_prob)

  # Find the top 10 prediction confidence indexes
  top_10_pred_indices = pred_prob.argsort()[-10:][::-1]
  # Find the top 10 prediction confidence values
  top_10_pred_values=pred_prob[top_10_pred_indices]
  # Find the top 10 prediction
  top_10_pred_labels=unique_breeds[top_10_pred_indices]

  # Setup plot
  top_plot = plt.bar(np.arange(len(top_10_pred_labels)),
                     top_10_pred_values,
                     color="grey")
```

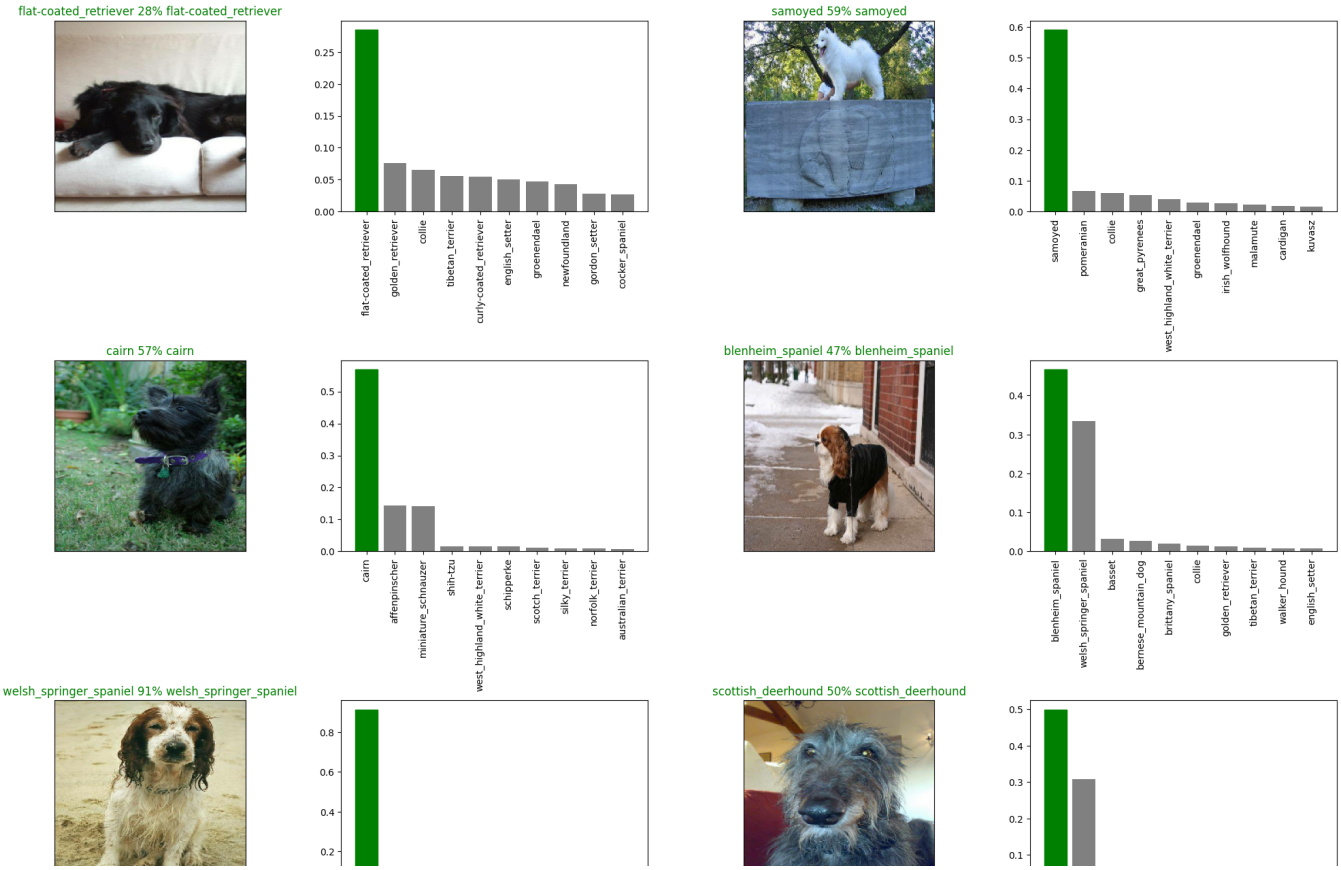
```
  plt.xticks(np.arange(len(top_10_pred_labels)),
             labels=top_10_pred_labels,
             rotation="vertical")

  # 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(predictions, val_labels, 9)
```



Now we've got some function to help us visualize our predictions and evaluate our model, let's check out a few.

```
# Let's check out a few predictions and their different values
i_multiplier = 20
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(predictions,val_labels,val_images,i+i_multiplier)
  plt.subplot(num_rows, 2*num_cols,2*i+2)
  plot_pred_conf(predictions,val_labels,i+i_multiplier)
plt.tight_layout(h_pad=1.0)
plt.show()
```



Saving and reloading a trained model

```
# Create a function to save a model
def save_model(model,suffix=None):
  """
  Saves a given model in a models directory and appends a suffix (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 format 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-moblienetv2-Adam")
```

```
Saving model to: drive/MyDrive/Dog Vision/models/20230711-05011689051687-1000-images-moblienetv2-Adam.h5...
'drive/MyDrive/Dog Vision/models/20230711-05011689051687-1000-images-moblienetv2-Adam.h5'
```

```
# Load a trained model
loaded_1000_image_model=load_model('drive/MyDrive/Dog Vision/models/20230710-06461688971571-1000-images-moblienetv2-Adam.h5')
```

```
Loading saved model from: drive/MyDrive/Dog Vision/models/20230710-06461688971571-1000-images-moblienetv2-Adam.h5
```

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

```
7/7 [==============================] - 1s 85ms/step - loss: 1.3006 - accuracy: 0.6800
[1.3006116151809692, 0.6800000071525574]
```

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

```
7/7 [==============================] - 1s 89ms/step - loss: 1.2006 - accuracy: 0.6900
[1.2005747556686401, 0.6899999976158142]
```

Training a big dog model 🐶 (on the full data)

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

```
(10222, 10222)
```

```
# 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))>
```

```
# Create a model for full model
full_model = create_model()
```

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/5
```

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

```
# 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 [==============================] - 37s 103ms/step - loss: 1.3341 - accuracy: 0.6635
Epoch 2/100
320/320 [==============================] - 34s 107ms/step - loss: 0.4010 - accuracy: 0.8809
Epoch 3/100
320/320 [==============================] - 34s 106ms/step - loss: 0.2381 - accuracy: 0.9344
Epoch 4/100
320/320 [==============================] - 33s 101ms/step - loss: 0.1550 - accuracy: 0.9657
Epoch 5/100
320/320 [==============================] - 34s 107ms/step - loss: 0.1078 - accuracy: 0.9768
Epoch 6/100
320/320 [==============================] - 34s 107ms/step - loss: 0.0799 - accuracy: 0.9851
Epoch 7/100
320/320 [==============================] - 33s 103ms/step - loss: 0.0593 - accuracy: 0.9915
Epoch 8/100
320/320 [==============================] - 34s 107ms/step - loss: 0.0471 - accuracy: 0.9936
Epoch 9/100
320/320 [==============================] - 33s 103ms/step - loss: 0.0370 - accuracy: 0.9957
Epoch 10/100
320/320 [==============================] - 33s 104ms/step - loss: 0.0329 - accuracy: 0.9963
Epoch 11/100
320/320 [==============================] - 34s 108ms/step - loss: 0.0266 - accuracy: 0.9978
Epoch 12/100
320/320 [==============================] - 34s 106ms/step - loss: 0.0238 - accuracy: 0.9976
Epoch 13/100
320/320 [==============================] - 33s 104ms/step - loss: 0.0193 - accuracy: 0.9986
Epoch 14/100
320/320 [==============================] - 33s 104ms/step - loss: 0.0180 - accuracy: 0.9985
Epoch 15/100
320/320 [==============================] - 33s 103ms/step - loss: 0.0155 - accuracy: 0.9988
Epoch 16/100
320/320 [==============================] - 34s 107ms/step - loss: 0.0155 - accuracy: 0.9984
Epoch 17/100
320/320 [==============================] - 34s 108ms/step - loss: 0.0133 - accuracy: 0.9987
Epoch 18/100
320/320 [==============================] - 34s 105ms/step - loss: 0.0148 - accuracy: 0.9980
<keras.callbacks.History at 0x7f033c3cfb80>
```

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

```
Saving model to: drive/MyDrive/Dog Vision/models/20230711-05131689052381-full-image-set-mobilenetv2-Adam.h5...
'drive/MyDrive/Dog Vision/models/20230711-05131689052381-full-image-set-mobilenetv2-Adam.h5'
```

```
# Load in the full model
loaded_full_model = load_model('drive/MyDrive/Dog Vision/models/20230710-11281688988534-full-image-set-mobilenetv2-Adam.h5')
```

```
Loading saved model from: drive/MyDrive/Dog Vision/models/20230710-11281688988534-full-image-set-mobilenetv2-Adam.h5
```

Making predictions on the test dataset

Since our model has been trailed on images in the form of Tensor batches, to make predictions on the test data, we'll have to get it into the same format.

we created `create_data_batches()` earlier which can take a list of filenames as input and convert them into Tensor batches.

To make predictions on the test data, we'll:

- Get the test image filenames ✅
- Convert the filenames into test data batches using `create_data_batches()` and setting the `test_data` parameter to `True` (since the test data doesn't have labels). ✅
- Make a predictions array by passing the test batches to the `predict()` method called on our model.

```
# Load test 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/de30e8d3cfd89cb021800007879f271b.jpg',
 'drive/MyDrive/Dog Vision/test/e241e149039740594a6af60dc18b269b.jpg',
 'drive/MyDrive/Dog Vision/test/e3c97ed588b32f49c7aae65cf91f17ba.jpg',
 'drive/MyDrive/Dog Vision/test/df58b248e14af15c43fb4a3ebc00e9a3.jpg',
 'drive/MyDrive/Dog Vision/test/e5f207e00213cc76be3ae9219a2e49cb.jpg',
 'drive/MyDrive/Dog Vision/test/e4607de37fdda1509487e042b6ca309a.jpg',
 'drive/MyDrive/Dog Vision/test/e596c0d97b381e876d46dda2eb60074c.jpg',
 'drive/MyDrive/Dog Vision/test/df2ce797bf398414aac7e20119c17cb6.jpg',
 'drive/MyDrive/Dog Vision/test/e00e5f7f822fd670bba314f87cce8428.jpg',
 'drive/MyDrive/Dog Vision/test/e079440ee0061b92ec22faf17be13908.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)>
```

```
# Make predictions on test data batch using the loaded full model
test_predictions = loaded_full_model.predict(test_data,
                                             verbose=1)
```

```
324/324 [==============================] - 40s 122ms/step
```

```
# Save predictions (Numpy array) to csv file (for access 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]
```

```
test_predictions.shape
```

```
(10357, 120)
```

```
test_predictions[:10]
```

```
array([[6.84861057e-09, 4.62954198e-12, 6.99940175e-11, ...,
        4.61346131e-13, 2.65197428e-08, 7.69998199e-09],
```

```
       [2.86082913e-09, 1.32134687e-07, 1.06946640e-09, ...,
        1.08532794e-10, 1.15802283e-08, 3.55831773e-08],
       [1.46219390e-04, 4.72112183e-10, 6.02694616e-10, ...,
        1.98041392e-11, 9.18782272e-10, 3.58254731e-10],
       ...,
       [4.20422930e-06, 4.05125282e-12, 1.28752120e-09, ...,
        5.32449529e-10, 1.90799847e-06, 1.61506068e-06],
       [9.97094421e-11, 2.32177444e-10, 7.40620509e-10, ...,
        3.41029360e-09, 3.51664531e-11, 2.20650520e-09],
       [5.11684140e-10, 5.17475085e-09, 9.59084808e-12, ...,
        3.36848672e-14, 1.38500398e-12, 3.18560933e-13]], dtype=float32)
```

Test data predictions

- Create a pandas DataFrame with an ID column as well as a column for each dog breed ✅
- Add data to the ID column by extracting the test image ID's from their filepaths
- Add data (the prediction probabilities) to each of the dog breed columns
- Export the DataFrame as a CSV.

```
# Create a pandas DataFrame with empty columns
preds_df = pd.DataFrame(columns=["id"]+list(unique_breeds))
preds_df.head()
```

id	affenpinscher	afghan_hound	african_hunting_dog	airedale	american_staffordshire_terrier	appenzeller	australian_terrier	b

0 rows × 121 columns

```
# Append test image ID's to predictions DataFrame
test_ids = [os.path.splitext(path)[0] for path in os.listdir(test_path)]
preds_df["id"] = test_ids
```

```
preds_df.head()
```

	id	affenpinscher	afghan_hound	african_hunting_dog	airedale	american_staffordshire_terrier	ap
0	de30e8d3cfd89cb021800007879f271b	NaN	NaN	NaN	NaN	NaN	
1	e241e149039740594a6af60dc18b269b	NaN	NaN	NaN	NaN	NaN	
2	e3c97ed588b32f49c7aae65cf91f17ba	NaN	NaN	NaN	NaN	NaN	
3	df58b248e14af15c43fb4a3ebc00e9a3	NaN	NaN	NaN	NaN	NaN	
4	e5f207e00213cc76be3ae9219a2e49cb	NaN	NaN	NaN	NaN	NaN	

5 rows × 121 columns

```
# Add the prediction probabilities to each dog breed column
preds_df[list(unique_breeds)]=test_predictions
preds_df.head()
```

	id	affenpinscher	afghan_hound	african_hunting_dog	airedale	american_staffordshire_terrier	a
0	de30e8d3cfd89cb021800007879f271b	6.848611e-09	4.629542e-12	6.999402e-11	1.152329e-12	1.478229e-12	
1	e241e149039740594a6af60dc18b269b	2.860829e-09	1.321347e-07	1.069466e-09	1.018921e-08	3.878894e-11	
2	e3c97ed588b32f49c7aae65cf91f17ba	1.462194e-04	4.721122e-10	6.026946e-10	2.990963e-09	6.250708e-08	
3	df58b248e14af15c43fb4a3ebc00e9a3	8.885264e-01	7.502086e-10	6.674922e-07	2.941582e-10	2.869572e-11	
4	e5f207e00213cc76be3ae9219a2e49cb	6.842721e-08	1.012004e-06	3.639626e-13	1.527982e-09	5.884661e-14	

5 rows × 121 columns

```
# Save our predictions dataframe to CSV
preds_df.to_csv("drive/MyDrive/Dog Vision/full_model_predictions_mobilenetv2.csv",
                index=False)
```

Making predictions on custom images

To make predictions on custom images, we'll:

- Get the filepaths of our own images.
- Turn the filepaths into data abtches using `create_data_batches`. And since our custom images won't have labels, we set the `test_data` parameter to `True`.
- Pass the custom image data batch to our model's `predict()` method.
- Convert the prediction output probabilities to prediction labels.
- Compae the predicted labels to the custom images.

```
# Get custom image filepaths
custom_path = "drive/MyDrive/Dog Vision/dog-photos/"
custom_image_paths = [custom_path + fname for fname in os.listdir(custom_path)]
# Turn custom images into batch datasets
custom_data = create_data_batches(custom_image_paths,test_data=True)
# Make predictions on the cuustom data
custom_preds = loaded_full_model.predict(custom_data)
# Get custom image prediction labels
custom_pred_labels = [get_pred_label(custom_preds[i]) for i in range(len(custom_preds))]
```

```
Creating test data batches...
1/1 [==============================] - 1s 784ms/step
```

```
custom_pred_labels
```

```
['lakeland_terrier', 'golden_retriever', 'labrador_retriever']
```

```
custom_images = []
for image in custom_data.unbatch().as_numpy_iterator():
  custom_images.append(image)
```

```
# Check custom image predictions
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)
```