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.

2. Data

The data is from Kaggle's dog breed identification competition. https://www.kaggle.com/c/dog-breed-identification/data

3. Evaluation

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

Info about the data:

- We're dealing with images (unstructured data) so we'll 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 are labeled)
- There are around 10,000+ images in the test set (these images are not labeled)

```
# Unzip th uploaded data in Google Drive
#!unzip "/content/drive/MyDrive/Dog Breed Identifier/dog-breed-identification.zip" -d "drive/
```

Get our workspace ready

- Import TenserFlow 2.7
- Import TensorFlow Hub
- Make sure we're using a GPU

```
# import necessary tools into Colab
import tensorflow as tf
import tensorflow_hub as hub
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
print("TF version", tf.__version__)
print("TF Hub Version", hub.__version__)

# check for GPU availability
print("GPU", "available :^)" if tf.config.list_physical_devices("GPU") else "not available :^
```

```
TF version 2.7.0
TF Hub Version 0.12.0
GPU not available :^(
```

Getting our data ready (turning it into Tesors)

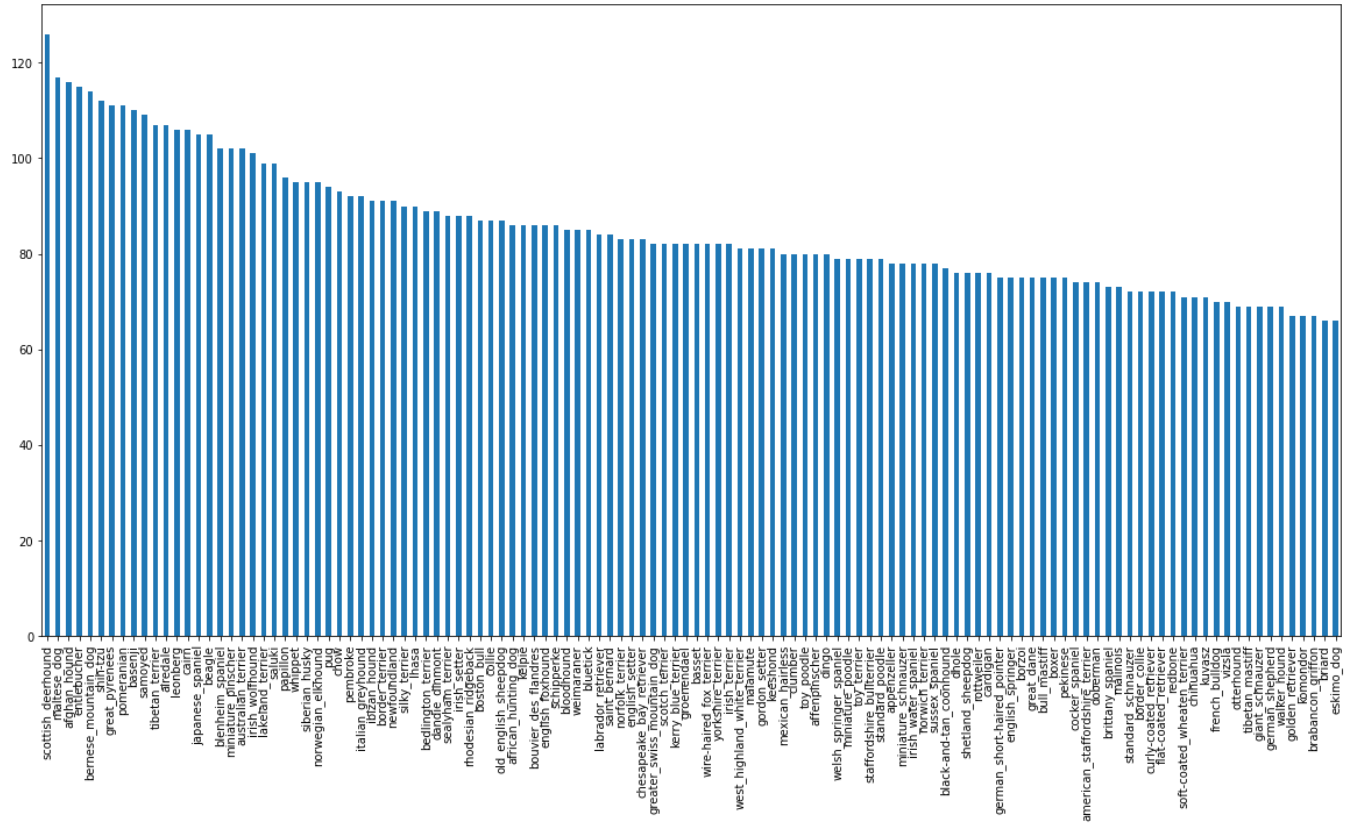
The data has to be in numerical format by turning it into Tensors

```
# Access data and check out the labels
labels_csv= pd.read_csv("/content/drive/MyDrive/Dog Breed Identifier/labels.csv")
print(labels_csv.describe())
labels_csv.head()
```

```
                                  id                breed
count                          10222                10222
unique                         10222                  120
top     ba1b7aa01e1c871c5e3acd08b65516e7  scottish_deerhound
freq                               1                  126
```

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

```
# Check images per breed
labels_csv["breed"].value_counts().plot.bar(figsize=(20, 10));
```




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

```
82.0
```

```
# check an image
from IPython.display import Image
Image("drive/MyDrive/Dog Breed Identifier/train/000bec180eb18c7604dcecc8fe0dba07.jpg")
```

Getting images and their labels

Get a list of the images and their pathnames

```
# create pathnames from image ID's
filenames = ["drive/MyDrive/Dog Breed Identifier/train/" + fname + ".jpg" for fname in labels
```

```
# checking first 10
filenames[:10]
```

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

```
# look at if the number file names is equal to the number of image files
import os
if len(os.listdir("drive/MyDrive/Dog Breed Identifier/train/")) == len(filenames):
  print("Filenames match the amount of files")
else:
  print("File names do not match the amount of files")
```

```
Filenames match the amount of files
```

```
# double checking
Image(filenames[422])
```

```
labels_csv["breed"][422]
```

```
'west_highland_white_terrier'
```

Now that the training filepaths are in a list, now I'll prepare the labels

```
# turn labels into numpy array
labels = labels_csv["breed"]
labels = np.array(labels)
labels
```

```
array(['boston_bull', 'dingo', 'pekinese', ..., 'airedale',
       'miniature_pinscher', 'chesapeake_bay_retriever'], dtype=object)
```

```
len(labels)
```

```
10222
```

```
# check for missing data
if len(labels) == len(filenames):
  print("Number of labels and filenames match")
else:
  print("Numbers don't match")
```

```
Number of labels and filenames match
```

```
# Find the number of unique labels
unique_breeds = np.unique(labels)
len(unique_breeds)
```

```
120
```

```
# 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, 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])]
```

```
# checking
len(boolean_labels)
```

```
10222
```

```
# Turning boolean array into integers
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 same label occurs
```

```
boston_bull
(array([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]
```

Creating a validation set

```
# setup X and y variables
X = filenames
y = boolean_labels
```

Start off with 1000 ish images and increase it as needed

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

NUM_IMAGES: 1000

```
# split 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)
```

```
# check training data
X_train[:2], y_train[:2]
```

```
(['drive/MyDrive/Dog Breed Identifier/train/00bee065dcec471f26394855c5c2f3de.jpg',
  'drive/MyDrive/Dog Breed Identifier/train/0d2f9e12a2611d911d91a339074c8154.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]),
  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

Write a function that:

1. Take an image filepath as input
2. Use Tenserflow to read the fila and save it to a variable `image`
3. Turn our `image` into Tensors
4. Normalize `image`
5. Resize the `image` to bea shape (224, 224)
6. Return the modified `image`

```
# define image size
IMG_SIZE = 224

# creat a function for preprocessing images
def process_image(image_path):
  """
  Takes an image file path an turns it into a tensor.
  """
  # read in an image file
  image = tf.io.read_file(image_path)

  # turn the image inot tensor with 3 color channels
  image = tf.image.decode_jpeg(image, channels=3)

  # convert the color channel values from 0-255 to 0-1 values
  image = tf.image.convert_image_dtype(image, tf.float32)

  # Resize the image to (224, 224)
  image = tf.image.resize(image, size=[IMG_SIZE, IMG_SIZE])

  return image
```

Turning the data into batches

Turn the images into batches because a GPU has a limited number of memory. That's why I'll do a 32 image batch size (if needed I'll adjust that)

The data needs to be in Tensor tuples: `(image, label)`

```
# function that returns a tuple of tensors
def get_image_label(image_path, label):
  """
  Takes image path name and the associated label, processes the image and returns a tuple of
```

```
    """
    image = process_image(image_path)
    return image, label


# demo of the above
(process_image(X[422]), tf.constant(y[422]))
```

```
    (<tf.Tensor: shape=(224, 224, 3), dtype=float32, numpy=
     array([[[0.50980395, 0.43137258, 0.3254902 ],
             [0.52258337, 0.44415197, 0.3382696 ],
             [0.5323442 , 0.45391282, 0.34803045],
             ...,
             [0.7960785 , 0.81568635, 0.82745105],
             [0.8106436 , 0.8207291 , 0.81561697],
             [0.80373776, 0.8115809 , 0.7998162 ]],

            [[0.501715  , 0.41936207, 0.30563655],
             [0.5050617 , 0.42270872, 0.3089832 ],
             [0.50872725, 0.42637432, 0.3126488 ],
             ...,
             [0.79572314, 0.815331  , 0.8270957 ],
             [0.80724794, 0.8235295 , 0.8178221 ],
             [0.7984944 , 0.81810224, 0.80241597]],

            [[0.48262435, 0.39220944, 0.28117123],
             [0.4877714 , 0.39735648, 0.28631827],
             [0.50558776, 0.41517282, 0.30413464],
             ...,
             [0.791941  , 0.8115488 , 0.82331353],
             [0.8045606 , 0.8204342 , 0.81595063],
             [0.79580706, 0.8127276 , 0.8051033 ]],

            ...,

            [[0.46269238, 0.384261  , 0.25484926],
             [0.464198  , 0.38576663, 0.25635484],
             [0.4717496 , 0.39331824, 0.26390645],
             ...,
             [0.31939158, 0.2919406 , 0.18934959],
             [0.3264622 , 0.27488655, 0.13976741],
             [0.338542  , 0.28364006, 0.13854204]],

            [[0.46566233, 0.38723096, 0.25781918],
             [0.45553428, 0.3771029 , 0.24769112],
             [0.47122204, 0.39279068, 0.2633789 ],
             ...,
             [0.29241225, 0.26496127, 0.16237026],
             [0.31125656, 0.2596809 , 0.12573904],
             [0.32333636, 0.2684344 , 0.13109216]],

            [[0.48111907, 0.4026877 , 0.2732759 ],
             [0.465345  , 0.38691363, 0.25750184],
             [0.48623097, 0.4077996 , 0.2783878 ],
             ...,
```

```
               [0.35759103, 0.33014005, 0.22754906],
               [0.3200722 , 0.26849654, 0.14122052],
               [0.332152  , 0.27725005, 0.13999516]]], 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,
              False, False, False, False, False, False, False, False, False,
              False, False, False, False, False, False, False, False, False,
              False, False, False, False, False, False, False, False, False
```

Make a function that turns `X` and `y` into batches.

```
# defin the batch size, 32 is where I'll start
BATCH_SIZE = 32

# 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 input (no labels).
  """
  # if test dataset, there are no labels
  if test_data:
    print("Creating test data batches")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X)))
    data_batch = data.map(process_image).batch(BATCH_SIZE)
    return data_batch

  # If valid data set, don't shuffle it
  elif valid_data:
    print("Creating validation data batches")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X),
                                               tf.constant(y)))
    data_batch = data.map(get_image_label).batch(BATCH_SIZE)
    return data_batch
  # if training data set, shuffle
  else:
    print("Create training data batches")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(X),
                                               tf.constant(y)))
    # shuffle pathnames and labels before mapping because it's shorter that way
    data = data.shuffle(buffer_size=len(X))

    # create (X, y) tuples and turns the image path into preprossed image
    data = data.map(get_image_label)

    # turn trining data into batches
```

```
    data_batch = data.batch(BATCH_SIZE)
```

```
# 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)
```

```
Create training data batches
Creating validation data batches
```

```
# check the different attributes
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)))
```

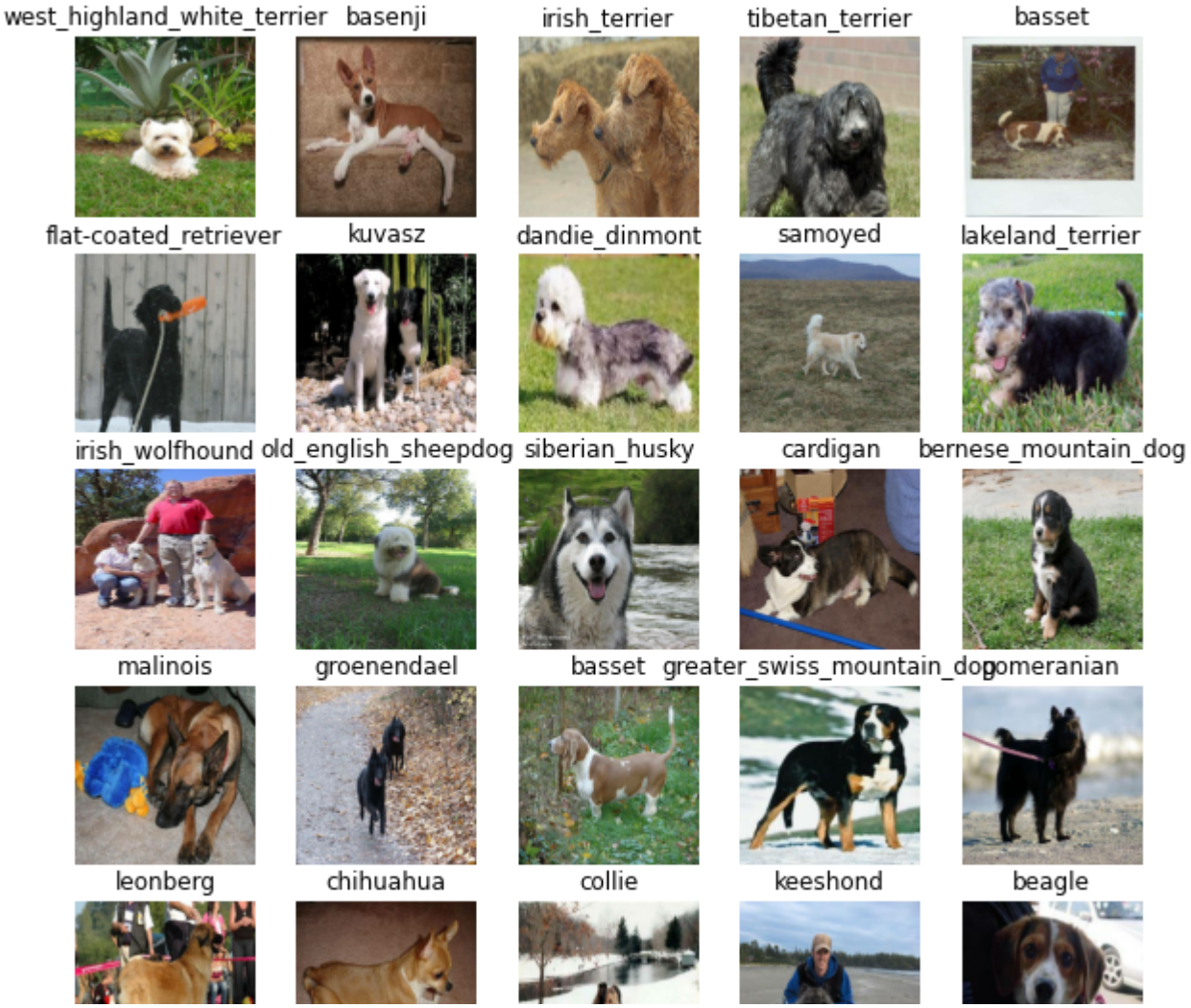
Visualizing Data Batches

```
# function for viewing images in data batch
def show_25_images(images, labels):
  """
  Displays a plot of 25 images and their labels from a data batch
  """
  # setup figure
  plt.figure(figsize=(10, 10))
  # loop through 25
  for i in range(25):
    # create subplots
    ax = plt.subplot(5, 5, i+1)
    # display 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_images, train_labels = next(train_data.as_numpy_iterator())
len(train_images), len(train_labels)
```

```
(32, 32)
```

```
# Use visualiztion  function
show_25_images(train_images, train_labels)
```




```
# visualize val set
val_images, val_labels = next(val_data.as_numpy_iterator())
show_25_images(val_images, val_labels)
```

Building the model

Things that are defined:

1. The input shape
2. The output shape
3. The URL of the model we want to use.

```
# setup intput shape to the model
INPUT_SHAPE = [None, IMG_SIZE, IMG_SIZE, 3]

# setup output shape
OUTPUT_SHAPE = len(unique_breeds)

# setup model URL from TenserFlor Hub
MODEL_URL = "https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/5"
```

Put inputs and outputs together in a Keras DL model.

Create a function that:

- Takes the input, output shape and model
- Defines the layers in the Keras model in a sequential fashion
- Compiles the model (tells how it should be evaluated and improved)
- Build the model (tells the model the input shape)
- Returns the model

Steps are from here: https://www.tensorflow.org/guide/keras/sequential_model

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

```
  Create a function that builds a Keras model in sequential fashion, compiles the model and b
  """

  #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/classificati
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
_________________________________________________________________
```

Create callbacks

Callback can be used during training to check or save its progress, or to stopr training early if the model stops improving

Create two callbacks:

1. TensorBoard to help track our models progress
2. Another for early stopping to prevent our model from training for too long

TensorBoard Callback

Three things to setup TensorBoard callback:

1. Load tensorboard notebook extension.
2. Create a tensorboard callback that can save logs to a directory and then pass it to the models `fit()` function.
3. Visualize the models training logs with the `%tensorboard` magic function.(after model training)

```
# Load TensorBoard notebook
%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 Breed Identifier/logs",
                        datetime.datetime.now().strftime("%Y%m%d%-%H%M%S")) # logs get tracke

  return tf.keras.callbacks.TensorBoard(logdir)
```

Early Stopping Callback

Stops overfitting by stopping training if a certain evaluation metric stops improving.

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

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

Training a model on a subset of data

First model will be trained on 1000 images to make sure I didn't mess up 😅

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

NUM_EPOCHS: 100

```
# check that the GPU is still connected
print("GPU", "available :^)"if tf.config.list_physical_devices("GPU") else "not available :^(
```

```
    GPU available :^)
```

Create a function that trains a model

- Create a model using `create_model()`
- Setup a TensorBoard 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

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

  # create ne TensorBoard session everytim a model is trained
  tensorboard = create_tensorboard_callback()

  # fit model to data passing it the callbacks
  model.fit(x=train_data,
            epochs=NUM_EPOCHS,
            validation_data=val_data,
            validation_freq=1,
            callbacks=[tensorboard, early_stopping])
  # return fitted model
  return model


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

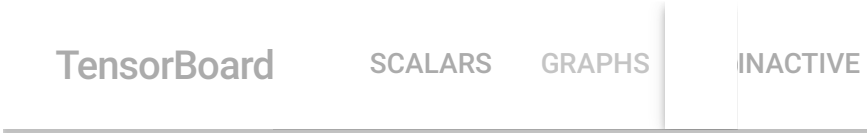
```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classificati
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

Checking the TensorBoard logs

TensorBoard magic function (`%tensorboard`) will access log directory that was created up top and visualize its contents.

```
        16             validation_freq=1,
```

```
%tensorboard --logdir drive/MyDrive/Dog\ Breed\ Identifier/logs
```



Making and evaluating prediction using the trained model

Ignore outliers in chart scalin

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

```
2/7 [=======>......................] - ETA: 14s
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-56-4476cb4a5493> in <module>()
      1 # predictions on the validation data (not used to train on)
----> 2 predictions = model.predict(val_data, verbose=1)
      3 predictions

8 frames
/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in
quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     57     ctx.ensure_initialized()
     58     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 59                                         inputs, attrs, num_outputs)
     60   except core._NotOkStatusException as e:
     61     if name is not None:

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW

```
predictions.shape
```

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

```
[1.24306956e-04 7.59398754e-05 1.90456354e-04 6.89914887e-05
 1.27923302e-03 4.79078262e-05 1.78473420e-04 4.71156818e-04
 6.57411711e-03 3.33541669e-02 8.13074148e-05 2.07506710e-05
 2.36903504e-03 7.01529766e-03 1.79723254e-03 2.09501712e-03
 4.63328288e-05 7.45164522e-04 1.13913280e-04 5.72121819e-04
 2.07568264e-05 2.76977895e-04 1.67627240e-05 3.18348575e-05
 1.33370394e-02 8.45316681e-05 6.67491186e-05 2.15364125e-04
 1.83231547e-04 3.03428951e-05 5.09509482e-05 3.93913913e-04
```

```
       3.64091102e-05 1.34550673e-05 6.15824902e-05 2.17158551e-04
       3.10434145e-04 1.12664828e-03 1.41421682e-04 3.14678013e-01
       2.14036801e-04 3.73189105e-05 5.40519226e-03 1.10154278e-05
       3.25166679e-04 3.43227875e-05 3.52709525e-04 6.85334322e-04
       1.35212222e-05 4.29256295e-04 1.41342927e-04 7.00257660e-05
       1.80110612e-04 1.48151407e-03 2.08097463e-05 3.72617098e-04
       2.13694380e-04 2.35952211e-05 1.76288711e-04 2.59588414e-04
       2.50986632e-05 8.77440849e-04 1.21370476e-05 1.01466539e-04
       4.43027675e-05 7.21494071e-05 5.40546243e-05 3.88546701e-04
       2.50868674e-04 1.53709843e-04 4.70330124e-04 2.67961586e-05
       6.13210505e-05 2.80884968e-04 5.98775914e-05 2.07673784e-05
       1.14561604e-04 8.47455594e-05 3.06809306e-05 5.05825388e-04
       4.87682200e-06 1.47580009e-04 4.71039057e-05 5.12759434e-04
       4.27719817e-04 1.19311466e-04 8.09227204e-05 5.00523856e-06
       1.40864213e-05 2.13727588e-03 4.23408928e-04 1.77149814e-05
       1.61246152e-03 2.69420387e-04 1.87507794e-05 8.12767903e-05
       7.27817087e-06 4.79529481e-05 7.14432099e-05 4.13986563e-05
       2.45181902e-04 9.36052456e-05 8.09364647e-05 9.00898813e-05
       1.84752498e-04 3.38674472e-05 6.04947039e-04 1.18089345e-04
       4.20262186e-05 1.31627792e-04 6.25466128e-05 2.19904585e-04
       2.31870814e-04 5.87293744e-01 1.19319884e-04 4.18462907e-04
       9.92042114e-05 3.64100160e-05 8.94907280e-04 3.13224940e-04]
Max value (probability of prediction): 0.5872937440872192
Sum: 1.0
Max index: 113
Predicted label: walker_hound
```

```
# Turn prediction probabilities in their labels
def get_pred_label(prediction_probabilities):
  """
  Turns an array of predictions into a label.
  """
  return unique_breeds[np.argmax(prediction_probabilities)]

# get a predicted labe based on prediction probabilities
pred_label = get_pred_label(predictions[99])
pred_label
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-57-1748971d40f3> in <module>()
      7
      8 # get a predicted labe based on prediction probabilities
----> 9 pred_label = get_pred_label(predictions[99])
     10 pred_label

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

SEARCH STACK OVERFLOW

Unbatchifying the dataset

```
# function for unbatchifying
def unbatchify(data):
  """
  Turns batched dataset of (image, label) Tensors, into separate arrays of images and labels.
  """
  images = []
  labels = []
  # loup through unbatched data
  for image, label in val_data.unbatch().as_numpy_iterator():
    images.append(image)
    labels.append(unique_breeds[np.argmax(label)])
  return images, labels

# Unbatchifing 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, 0.2770141 , 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 , 0.9125331 , 0.8460338 ],
       ...,
       [0.08394483, 0.08394483, 0.08394483],
       [0.0886985 , 0.0886985 , 0.0886985 ],
       [0.04514172, 0.04514172, 0.04514172]]], dtype=float32), 'cairn')
```

Make functions to visualize:

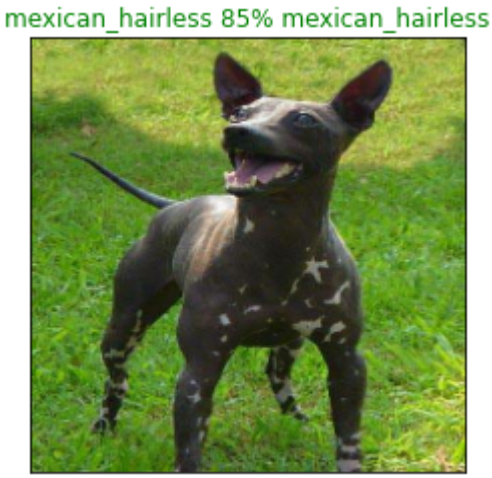
- Prediction labels
- Validation labels
- Validation images

Create a function that:

- Takes an array of prediction probabilities, an array of truth labels and an array of images and an integer.
- Converts the prediction probabilities to a prediction label.
- Plots the predicted label, its predicted probability, truth label and the targeted image in one 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 and remove ticks
  plt.imshow(image)
  plt.xticks([])
  plt.yticks([])

  # Make the color of the title green for right and red for wrong
  if pred_label == true_label:
    color = "green"
  else:
    color = "red"

  # Plot title to 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=70)
```



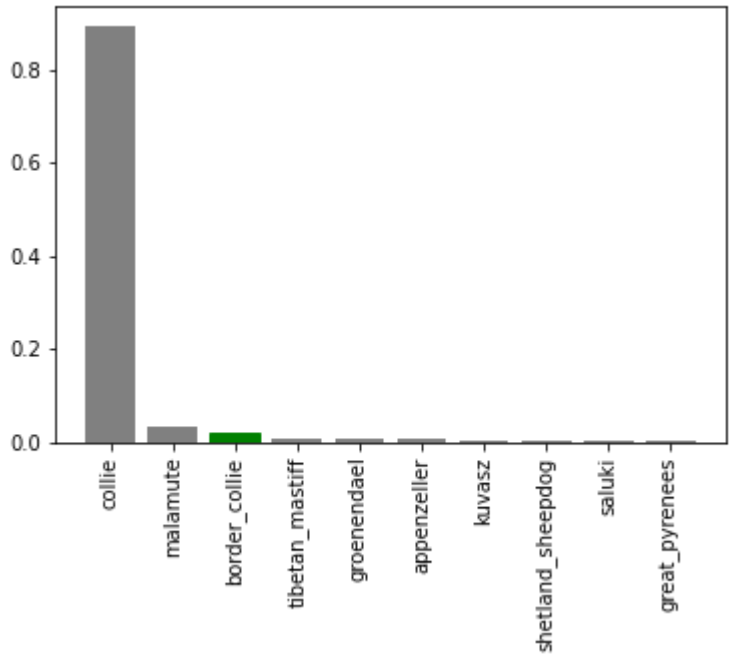
Function to view top 10 predictions

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 probability values and labels, coloring true labels green

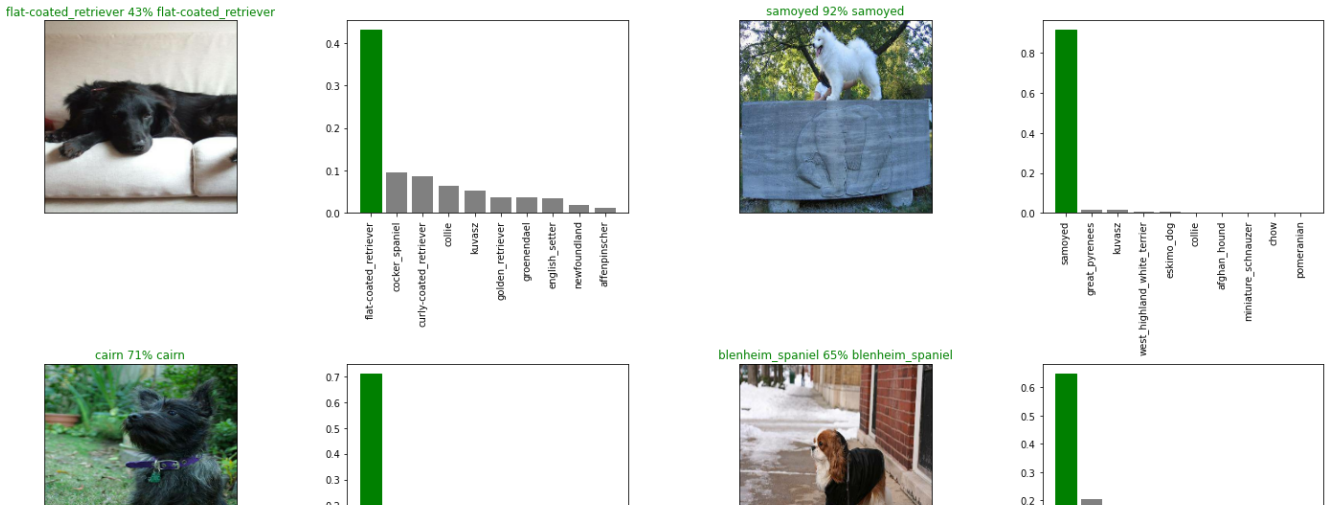
```
def plot_pred_conf(prediction_probabilities, labels, n=1):
  """
  Plots the top 10 highest prediction confidences along with the truth label for sample n.
  """
  pred_prob, true_label = prediction_probabilities[n], labels[n]

  # Get the prediction label
  pred_label = get_pred_label(pred_prob)

  # Top 10 predicition confidence indexes
```

```
  top_10_pred_indexes = pred_prob.argsort()[-10:][::-1]

  # Top 10 prediction confidence values
  top_10_pred_values = pred_prob[top_10_pred_indexes]

  # Top 10 prediction labels
  top_10_pred_labels = unique_breeds[top_10_pred_indexes]

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




```
# FUnction for checking out a few predictions and their different values
def plot_pred_dif(prediction_probabilities, labels, n=1):
  """
  Checks out a few predcions and their 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(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)


  plot_pred_dif(prediction_probabilities=predictions,
                labels=val_labels)
```



Saving and reloading the model

```
# Function to save model
def save_model(model, suffix=None):
  """
  Saves the model in the models directory and appends a suffix.
  """
  # Create a model directory pathname with timestamp
  modeldir = os.path.join("drive/MyDrive/Dog Breed Identifier/Models",
                          datetime.datetime.now().strftime("%Y%m%d-%H%M%s"))
  # save format
  model_path = modeldir + "-" + suffix + ".h5"
  print(f"Saving model to: {model_path}...")
  model.save(model_path)
  return model_path


# Function to load 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


# check if save works
save_model(model, suffix="1000-images-mobilenetv2-Adam")
```

```
Saving model to: drive/MyDrive/Dog Breed Identifier/Models/20220117-15191642432790-1000-
'drive/MyDrive/Dog Breed Identifier/Models/20220117-15191642432790-1000-images-mobilene
tv2-Adam.h5'
```

```
# check if load works
```

```
loaded_1000_image_model = load_model("drive/MyDrive/Dog Breed Identifier/Models/20220117-1519
```

```
Loading saved model from: drive/MyDrive/Dog Breed Identifier/Models/20220117-1519164243
```

```
# check pre-saved model
model.evaluate(val_data)
```

```
7/7 [==============================] - 1s 118ms/step - loss: 1.2953 - accuracy: 0.6500
[1.2952961921691895, 0.6499999761581421]
```

```
# check saved model data
loaded_1000_image_model.evaluate(val_data)
```

```
7/7 [==============================] - 2s 120ms/step - loss: 1.2953 - accuracy: 0.6500
[1.2952961921691895, 0.6499999761581421]
```

Training model on the full data

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

```
Create training data batches
```

```
# Checking
full_data
```

```
<BatchDataset shapes: ((None, 224, 224, 3), (None, 120)), types: (tf.float32, tf.bool)>
```

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

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

```
# create full model callbacks
full_model_tensorboard = create_tensorboard_callback()
# To prevent overfitting
full_model_early_stopping = tf.keras.callbacks.EarlyStopping(monitor="accuracy",
                                                             patience=3)
```

NOTE: The cell below will take a little bit (like 30ish minutes) because the GPU has to load all of the images into memory.

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

```
Epoch 1/100
320/320 [==============================] - 53s 148ms/step - loss: 1.3171 - accuracy: 0.6
Epoch 2/100
320/320 [==============================] - 48s 151ms/step - loss: 0.3980 - accuracy: 0.8
Epoch 3/100
320/320 [==============================] - 50s 156ms/step - loss: 0.2375 - accuracy: 0.9
Epoch 4/100
320/320 [==============================] - 51s 161ms/step - loss: 0.1565 - accuracy: 0.9
Epoch 5/100
320/320 [==============================] - 52s 161ms/step - loss: 0.1054 - accuracy: 0.9
Epoch 6/100
320/320 [==============================] - 51s 159ms/step - loss: 0.0768 - accuracy: 0.9
Epoch 7/100
320/320 [==============================] - 56s 175ms/step - loss: 0.0583 - accuracy: 0.9
Epoch 8/100
320/320 [==============================] - 56s 173ms/step - loss: 0.0471 - accuracy: 0.9
Epoch 9/100
320/320 [==============================] - 55s 172ms/step - loss: 0.0368 - accuracy: 0.9
Epoch 10/100
320/320 [==============================] - 55s 172ms/step - loss: 0.0319 - accuracy: 0.9
Epoch 11/100
320/320 [==============================] - 56s 173ms/step - loss: 0.0258 - accuracy: 0.9
Epoch 12/100
320/320 [==============================] - 55s 173ms/step - loss: 0.0229 - accuracy: 0.9
Epoch 13/100
320/320 [==============================] - 55s 172ms/step - loss: 0.0200 - accuracy: 0.9
Epoch 14/100
320/320 [==============================] - 55s 173ms/step - loss: 0.0177 - accuracy: 0.9
Epoch 15/100
320/320 [==============================] - 56s 175ms/step - loss: 0.0158 - accuracy: 0.9
Epoch 16/100
320/320 [==============================] - 56s 176ms/step - loss: 0.0140 - accuracy: 0.9
Epoch 17/100
320/320 [==============================] - 57s 177ms/step - loss: 0.0129 - accuracy: 0.9
Epoch 18/100
320/320 [==============================] - 58s 180ms/step - loss: 0.0128 - accuracy: 0.9
Epoch 19/100
320/320 [==============================] - 57s 178ms/step - loss: 0.0110 - accuracy: 0.9
<keras.callbacks.History at 0x7f2baefcf4d0>
```

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

```
Saving model to: drive/MyDrive/Dog Breed Identifier/Models/20220117-16091642435743-full-
'drive/MyDrive/Dog Breed Identifier/Models/20220117-16091642435743-full-image-set-mobil
enetv2-Adam.h5'
```

```
# load in full model
loaded_full_model = load_model("drive/MyDrive/Dog Breed Identifier/Models/20220117-1609164243
```

```
Loading saved model from: drive/MyDrive/Dog Breed Identifier/Models/20220117-1609164243
```

Making predictions on the test data set

Turn test data into Tensor batches using `create_data_batches()`

To make predictions:

- Get test image filenames
- Convert using `create_data_batches()`
- Set thee `test_data` parameter to `True`
- Make a predictions array by passing the test batches to the `predict()` method called on the model

```
# Load test image filenames
test_path = "drive/MyDrive/Dog Breed Identifier/test/"
test_filenames = [test_path + fname for fname in os.listdir(test_path)]
test_filenames[:10]
```

```
['drive/MyDrive/Dog Breed Identifier/test/e29d2336a8559d96973c874c9c6c17c6.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e219af838e1d6a18224eb9b478944778.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e090f0f0ebc83ddf5f649a841493868b.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/dd3c80cee38d165aaf48083f4a4a0071.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e2537e98808877c707bfe8ca53e303b7.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e41802f671c437c10e843400dcea40fb.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e7e9dd51302fe08c1c89a550e183cf07.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e0d51afc60c25eb2205be1644af09cc5.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/dfe02d52ca281aaca6215a42fee6245c.jpg',
 'drive/MyDrive/Dog Breed Identifier/test/e1e79b3edfb3579e46ad914bf755dbbc.jpg']
```

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

```
Creating test data batches
```

NOTE The next cell will take like maybe an hour to run 😅

```
# Prediction array
#test_predictions = loaded_full_model.predict(test_data,
                                              verbose=1)
```

```
324/324 [==============================] - 869s 3s/step
```

```
# Save predictions to csv file
#np.savetxt("drive/MyDrive/Dog Breed Identifier/preds_array.csv", test_predictions, delimiter
```

```
# Load predictions from csv
test_predictions = np.loadtxt("drive/MyDrive/Dog Breed Identifier/preds_array.csv", delimiter
```

```
test_predictions[:10]
```

```
array([[1.08666697e-07, 1.41948773e-08, 3.22431681e-08, ...,
        1.18795462e-09, 1.56976370e-04, 1.10274367e-08],
       [7.92604107e-11, 4.80689710e-09, 2.62021373e-08, ...,
        1.78886221e-05, 5.14278287e-10, 8.97830310e-10],
       [5.42786389e-12, 3.95445765e-09, 9.73456732e-11, ...,
        2.14865480e-07, 2.52733980e-06, 1.45722615e-10],
       ...,
       [2.18277751e-08, 2.20443021e-11, 2.85038703e-13, ...,
        4.21420676e-09, 1.19169924e-10, 6.25289820e-11],
       [6.85834110e-08, 2.93119667e-10, 1.14738228e-11, ...,
        5.77665936e-11, 3.19699212e-10, 1.01442104e-07],
       [2.23612484e-09, 9.42444256e-13, 8.19952284e-09, ...,
        8.63977778e-10, 6.13988576e-08, 3.77298579e-06]])
```

```
test_predictions.shape
```

```
(10357, 120)
```

Preparing test data set predictions for Kaggle

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

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

id	affenpinscher	afghan_hound	african_hunting_dog	airedale	american_staffordshir

0 rows × 121 columns

```
# appending test image ID's to preds_df
#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
0	e29d2336a8559d96973c874c9c6c17c6	NaN	NaN	NaN
1	e219af838e1d6a18224eb9b478944778	NaN	NaN	NaN
2	e090f0f0ebc83ddf5f649a841493868b	NaN	NaN	NaN
3	dd3c80cee38d165aaf48083f4a4a0071	NaN	NaN	NaN
4	e2537e98808877c707bfe8ca53e303b7	NaN	NaN	NaN

5 rows × 121 columns

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

	id	affenpinscher	afghan_hound	african_hunting_dog
0	e29d2336a8559d96973c874c9c6c17c6	1.08667e-07	1.41949e-08	3.22432e-08
1	e219af838e1d6a18224eb9b478944778	7.92604e-11	4.8069e-09	2.62021e-08
2	e090f0f0ebc83ddf5f649a841493868b	5.42786e-12	3.95446e-09	9.73457e-11
3	dd3c80cee38d165aaf48083f4a4a0071	1.68976e-05	2.39422e-10	1.07246e-11
4	e2537e98808877c707bfe8ca53e303b7	7.59418e-14	5.16135e-12	7.61826e-15

5 rows × 121 columns

```
# save pred_df to csv
#preds_df.to_csv("drive/MyDrive/Dog Breed Identifier/full_model_preds_submission_1_mobilenetV
                index=False)
```

Predictions on my dogs!

What to do:

- Get the filepaths
- Turn them into data batches using `create_data_batches` and set `test_data` parameter to `True`
- Pass the custom image data batch to our model's `predict()`
- Convert the prediction outpur probabilities to prediction labels
- Compare the predicted label to the custom images

```
# setup the custom path
custom_path = "drive/MyDrive/Dog Breed Identifier/MyDogs/"
custom_image_paths = [custom_path + fname for fname in os.listdir(custom_path)]
```

```
custom_image_paths
```

```
['drive/MyDrive/Dog Breed Identifier/MyDogs/2475344D-1D29-49D1-8833-DF27942B9768.jpeg',
 'drive/MyDrive/Dog Breed Identifier/MyDogs/0AD0F0E1-1628-42EB-BE4E-C20149FFF18C.jpeg',
 'drive/MyDrive/Dog Breed Identifier/MyDogs/F38E6B93-C09F-4399-9D4E-4FA7A59956A3.jpeg',
 'drive/MyDrive/Dog Breed Identifier/MyDogs/8DA971AD-3182-4FB0-A67E-C7D157E31076.jpeg']
```

```
# turn images into batches
custom_data = create_data_batches(custom_image_paths, test_data=True)
custom_data
```

```
Creating test data batches
<BatchDataset shapes: (None, 224, 224, 3), types: tf.float32>
```

```
# Make predictions
custom_preds = loaded_full_model.predict(custom_data)
custom_preds.shape
```

```
(4, 120)
```

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

```
['keeshond', 'labrador_retriever', 'eskimo_dog', 'eskimo_dog']
```

```
# get custom images
custom_images = []
# loop through unbatched data
```

```
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, 4, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.title(custom_pred_labels[i])
  plt.imshow(image)
```