ML-10

NEURAL NETWORKS DEEP LEARNING, TRANSFER LEARNING AND TENSORFLOW 2

Muhammad Naqeeb| Machine-Learning | August 21, 2021

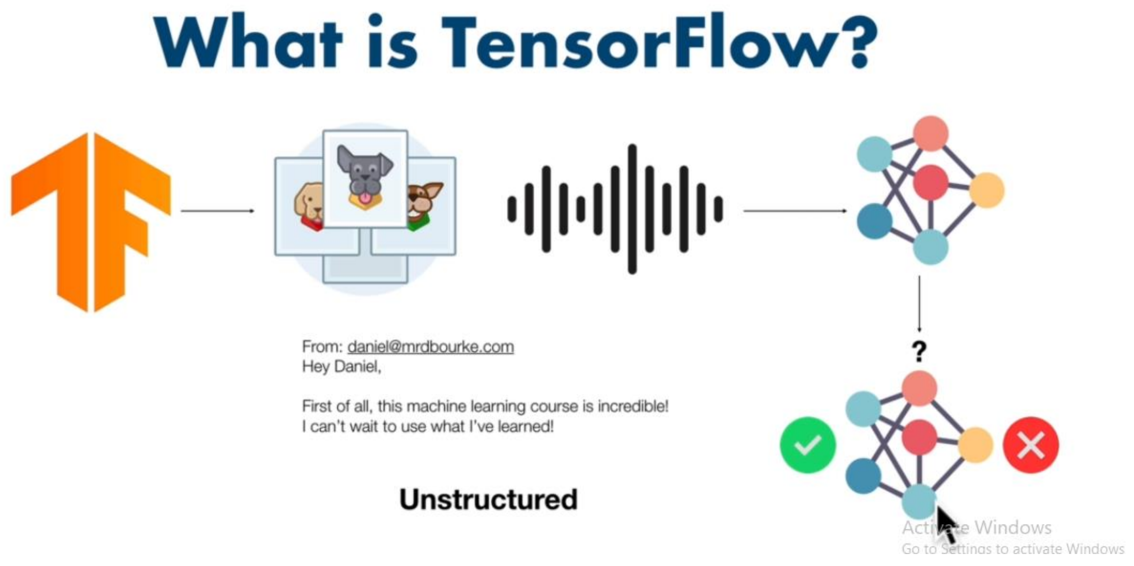
TensorFlow

TensorFlow is a deep learning or numerical computing library.

TensorFlow is used to build deep learning and neural network models to gain, insights out of unstructured data.

Deep learning is more suited to unstructured data such as photos or audio waves or natural language text. Unstructured data that doesn't really have any specific structure

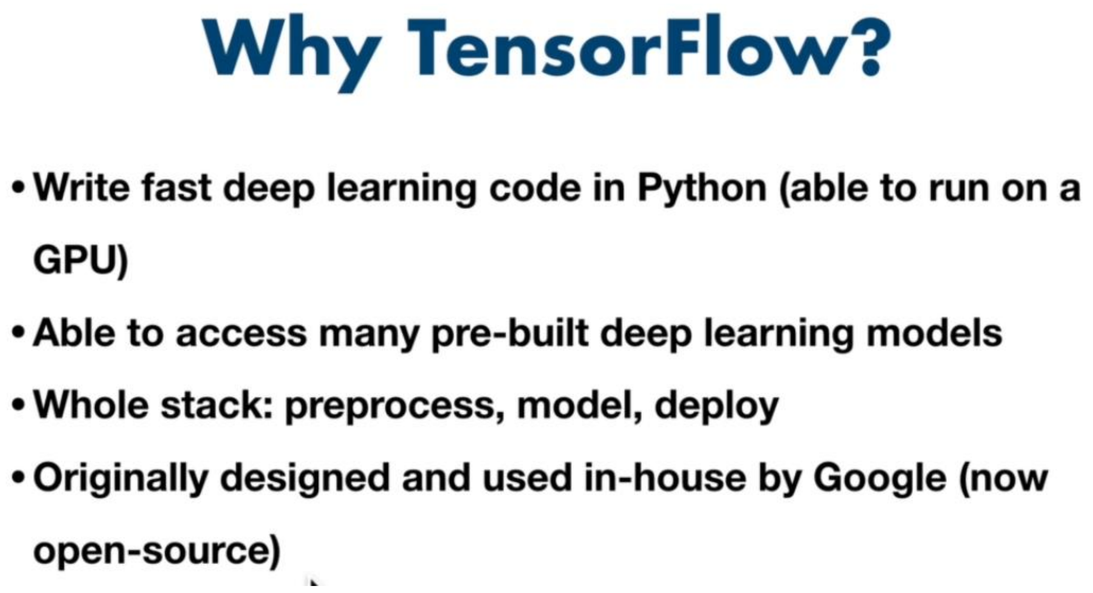
Now tensor flow allows us to build a deep learning model kind of like a neural network.



What is TensorFlow?

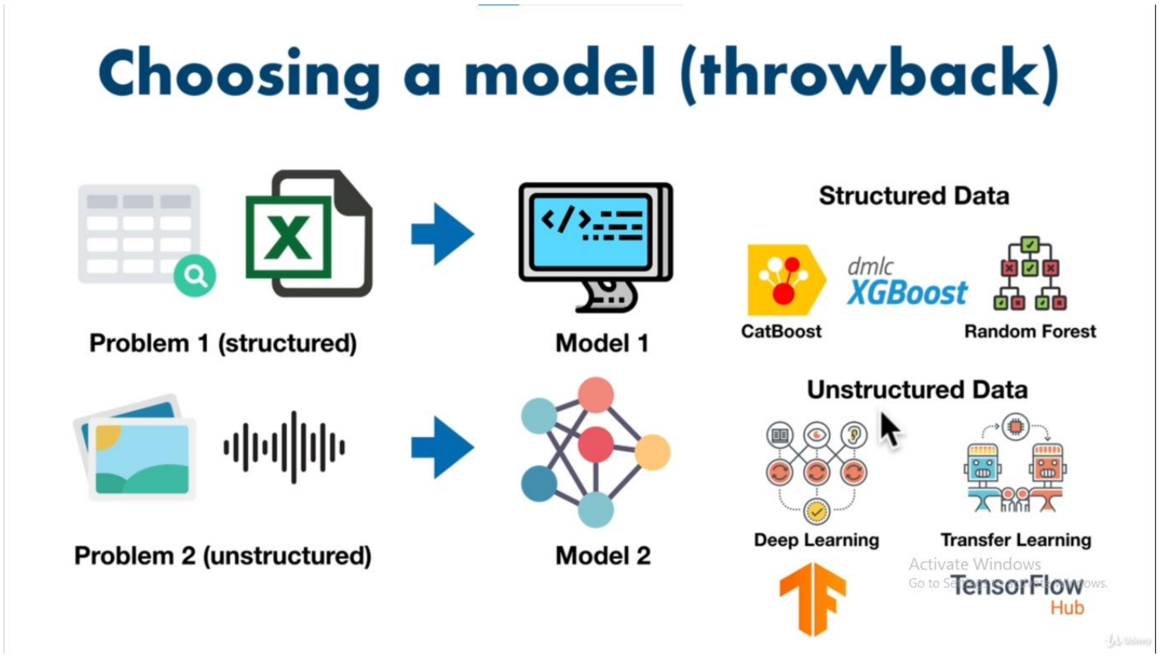
From: daniel@mrdbourke.com
Hey Daniel,

First of all, this machine learning course is incredible!
I can't wait to use what I've learned!



Why TensorFlow?

- Write fast deep learning code in Python (able to run on a GPU)
- Able to access many pre-built deep learning models
- Whole stack: preprocess, model, deploy
- Originally designed and used in-house by Google (now open-source)

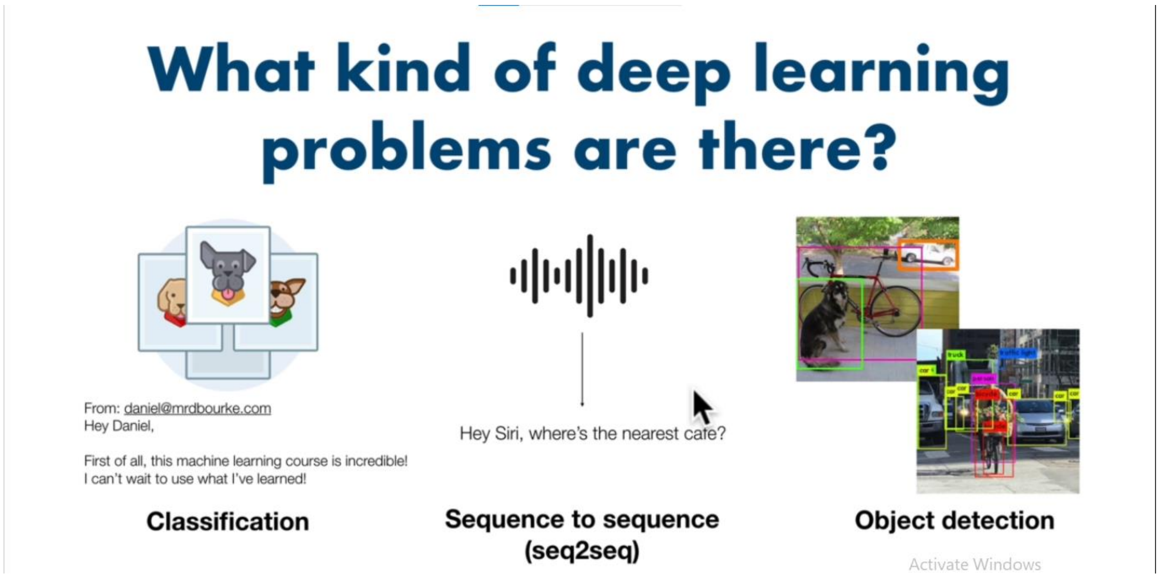


Choosing a model (throwback)

Neural Network

Deep Learning

Kind of Deep Learning problems

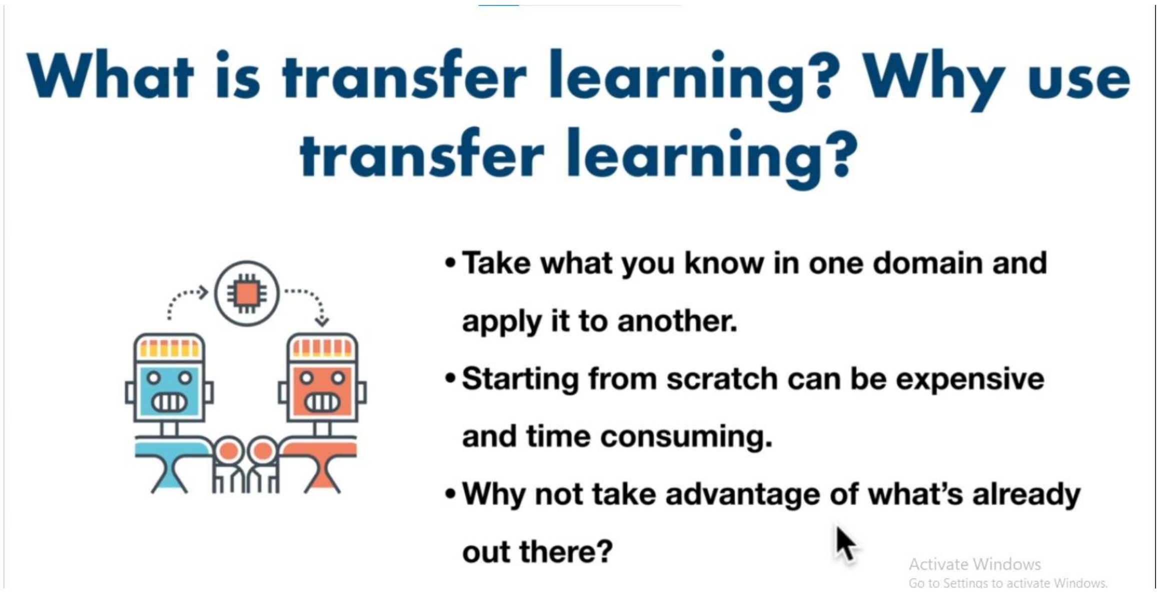


What kind of deep learning problems are there?

From: daniel@mrdbourke.com
Hey Daniel,

First of all, this machine learning course is incredible!
I can't wait to use what I've learned!

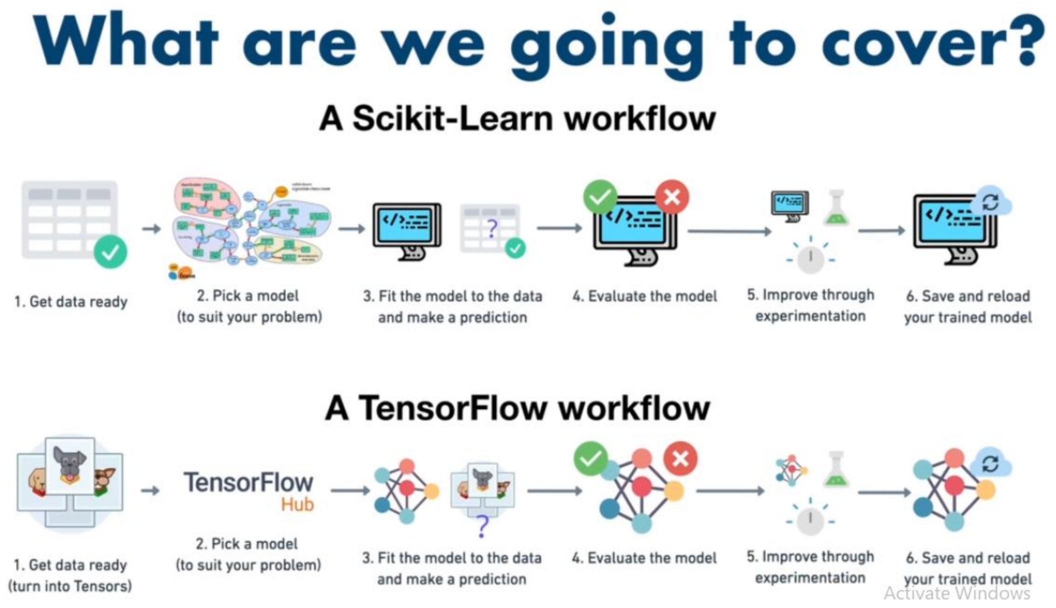
Hey Siri, where's the nearest cafe?



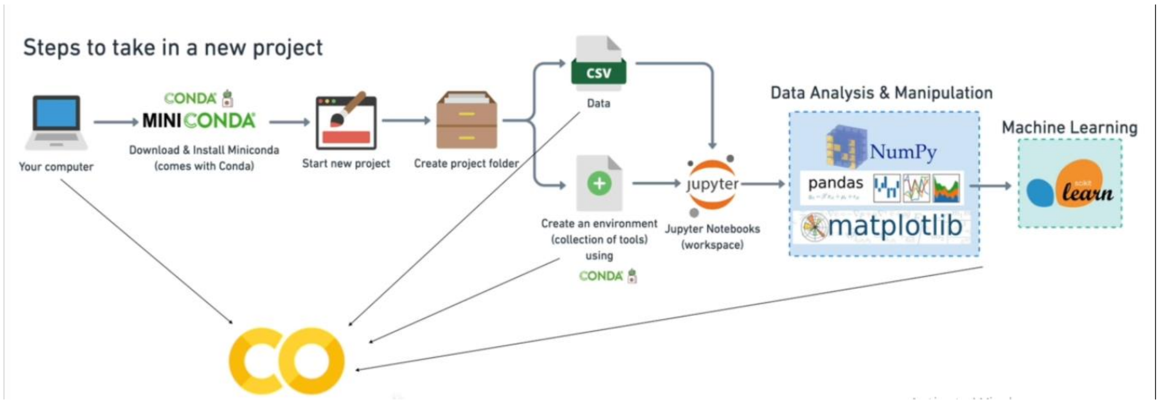
What is transfer learning? Why use transfer learning?

- Take what you know in one domain and apply it to another.
- Starting from scratch can be expensive and time consuming.
- Why not take advantage of what's already out there?

Workflow

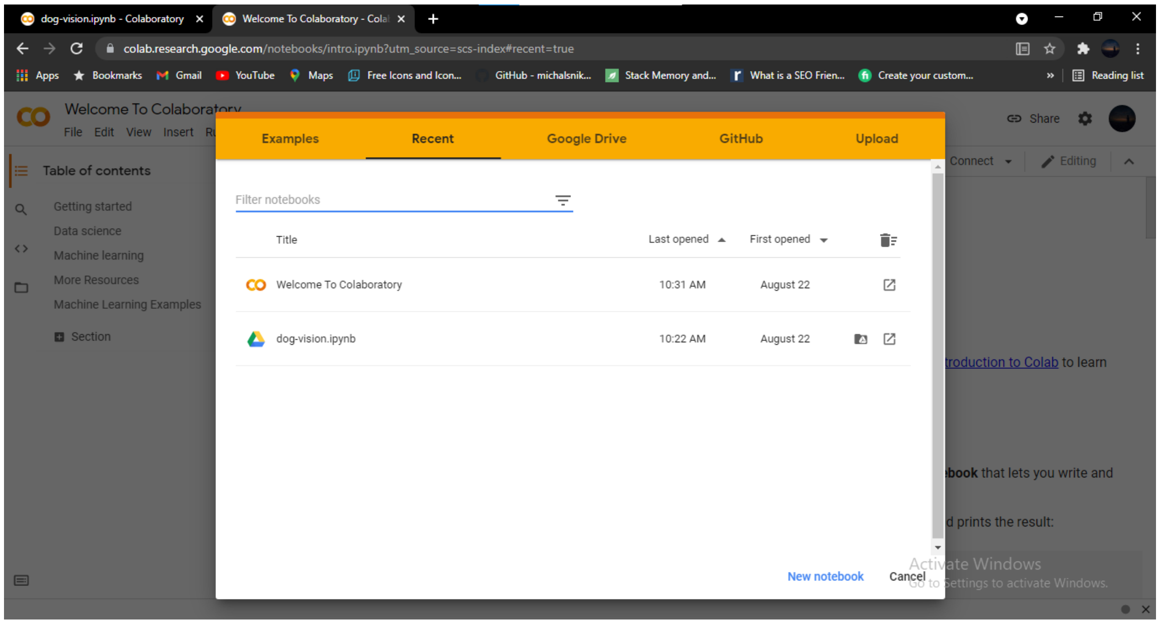

What are we going to cover?
A Scikit-Learn workflow
1. Get data ready
2. Pick a model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
A TensorFlow workflow
1. Get data ready (turn into Tensors)
TensorFlow Hub
2. Pick a model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
Activate Windows

Steps to take in a new project


Steps to take in a new project
Your computer
CONDA
MINICONDA
Download & Install Miniconda (comes with Conda)
Start new project
Create project folder
CSV
Data
Create an environment (collection of tools) using CONDA
jupyter
Jupyter Notebooks (workspace)
Data Analysis & Manipulation
NumPy
pandas
matplotlib
Machine Learning
learn

Setup colab

- Search for colab in google
- Go to colab
- Check for new python notebook



Title	Last opened	First opened
Welcome To Colaboratory	10:31 AM	August 22
dog-vision.ipynb	10:22 AM	August 22

Put the data in google drive and mount google drive

Unzip Files

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

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.

If I was sitting at that cafe and took a photo of that dog. Could my machine learning model, the one that we're going to build, tell me what breed it is.

2. Data

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

3. Evaluation

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

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)

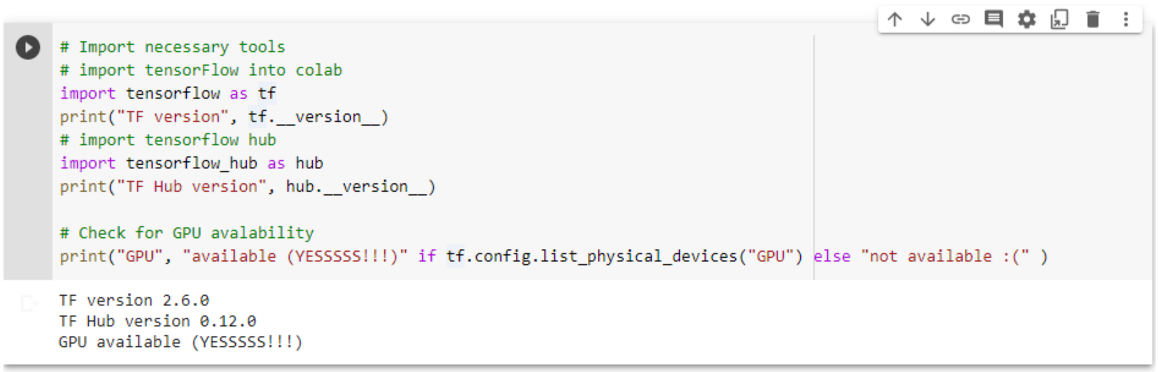
Get our workspace ready

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

For enable GPU in colab

- Go to **Runtime**
- Click **Change runtime complexity**
- Check for **GPU**
- Click **save**

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

# Check for GPU avalability
print("GPU", "available (YESSSSS!!!)" if tf.config.list_physical_device
s("GPU") else "not available :(" )
```



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

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

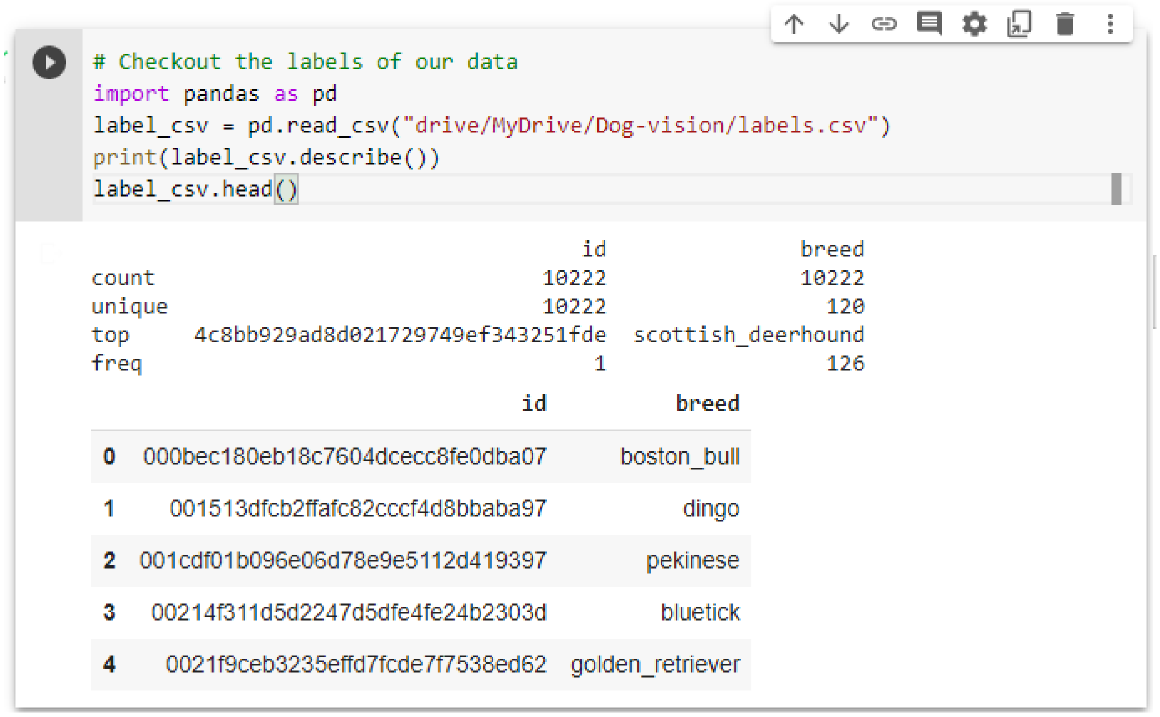
```
TF version 2.6.0
TF Hub version 0.12.0
GPU available (YESSSSS!!!)
```

1. Getting 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 representations)

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

```
# Checkout the labels of our data
import pandas as pd
label_csv = pd.read_csv("drive/MyDrive/Dog-vision/labels.csv")
print(label_csv.describe())
label_csv.head()
```

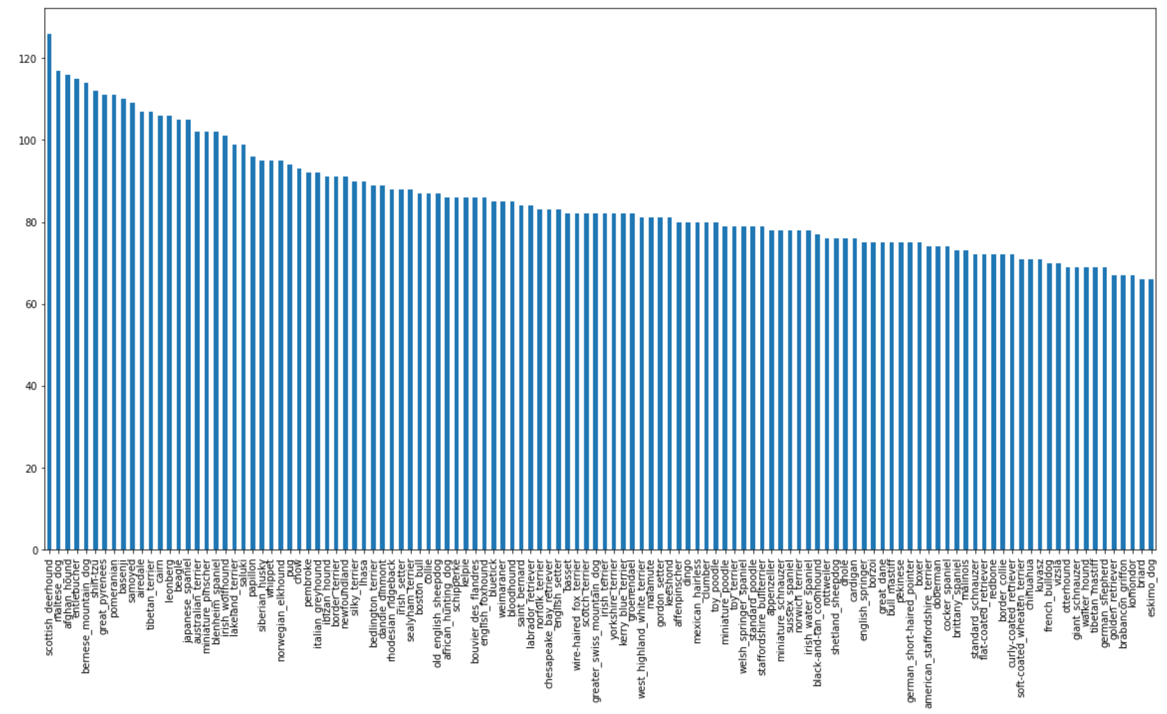


```
# Checkout the labels of our data
import pandas as pd
label_csv = pd.read_csv("drive/MyDrive/Dog-vision/labels.csv")
print(label_csv.describe())
label_csv.head()
```

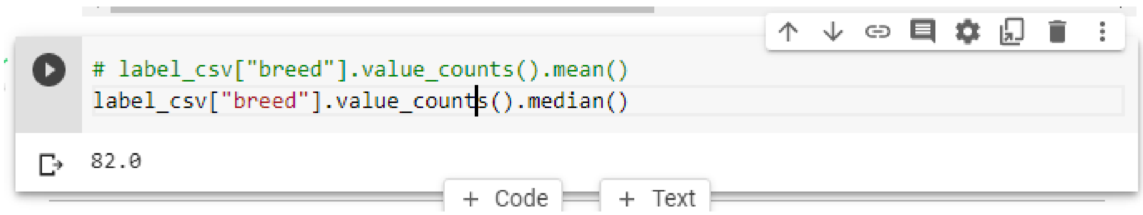
```
                                      id               breed
count                              10222               10222
unique                             10222                 120
top     4c8bb929ad8d021729749ef343251fde  scottish_deerhound
freq                                   1                 126
```

	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?
# label_csv["breed"].value_counts()
label_csv["breed"].value_counts().plot.bar(figsize = (20,10))
```



```
# label_csv["breed"].value_counts().mean()
label_csv["breed"].value_counts().median()
```



```
# label_csv["breed"].value_counts().mean()
label_csv["breed"].value_counts().median()
```

```
82.0
```

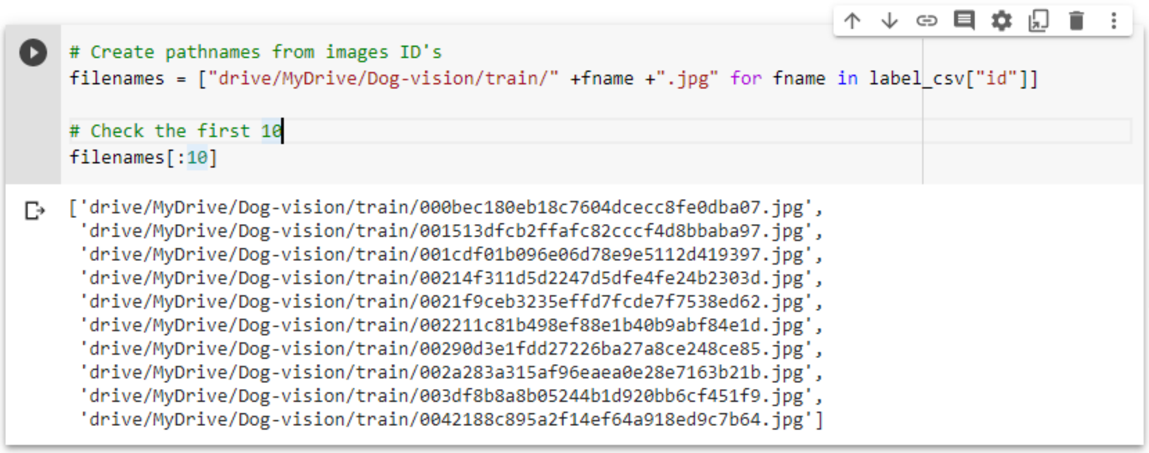
```
# Let's view an image
from IPython.display import Image
Image("drive/MyDrive/Dog-
vision/train/001513dfcb2ffafc82cccf4d8bbaba97.jpg")
```

Getting images and their labels

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

```
# Create pathnames from images ID's
```

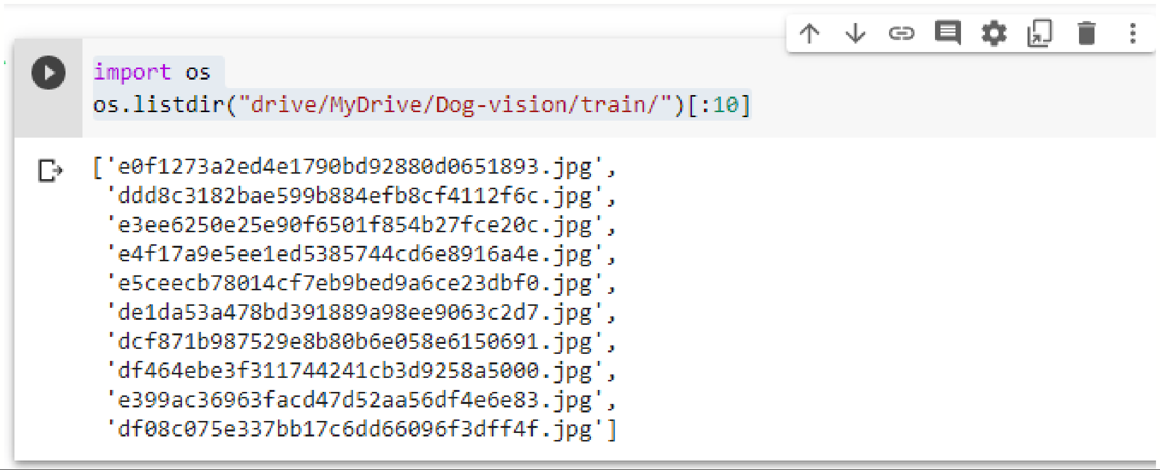
```
filenames = ["drive/MyDrive/Dog-
vision/train/" +fname +".jpg" for fname in label_csv["id"]]

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



```
# Create pathnames from images ID's
filenames = ["drive/MyDrive/Dog-vision/train/" +fname +".jpg" for fname in label_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]
```



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

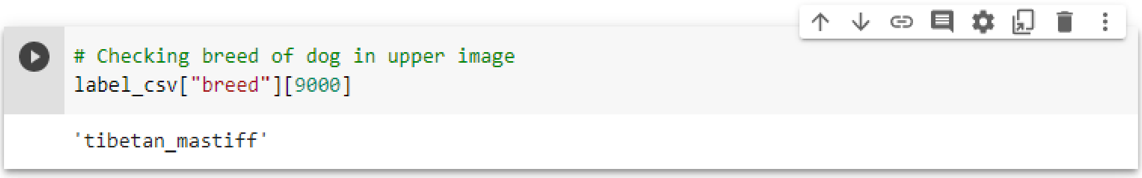
```
['e0f1273a2ed4e1790bd92880d0651893.jpg',
 'ddd8c3182bae599b884efb8cf4112f6c.jpg',
 'e3ee6250e25e90f6501f854b27fce20c.jpg',
 'e4f17a9e5ee1ed5385744cd6e8916a4e.jpg',
 'e5ceecb78014cf7eb9bed9a6ce23dbf0.jpg',
 'de1da53a478bd391889a98ee9063c2d7.jpg',
 'dcf871b987529e8b80b6e058e6150691.jpg',
 'df464ebe3f311744241cb3d9258a5000.jpg',
 'e399ac36963facd47d52aa56df4e6e83.jpg',
 'df08c075e337bb17c6dd66096f3dff4f.jpg']
```

```
# Check whether number of filenames matches number of actual image file
s
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 targ
et directory.")
```

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

```
# One more check / checking image in 9000 index
Image(filenames[9000])
```

```
# Checking breed of dog in upper image
label_csv["breed"][9000]
```



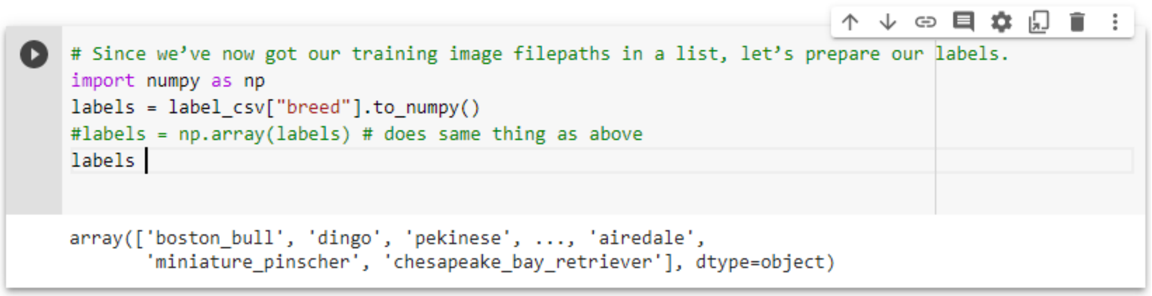
```
# Checking breed of dog in upper image
label_csv["breed"][9000]
```

```
'tibetan_mastiff'
```

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

CodeText

```
# Since we’ve now got our training image filepaths in a list, let’s pre
pare our labels.
import numpy as np
labels = label_csv["breed"].to_numpy()
#labels = np.array(labels) # does same thing as above
labels
```

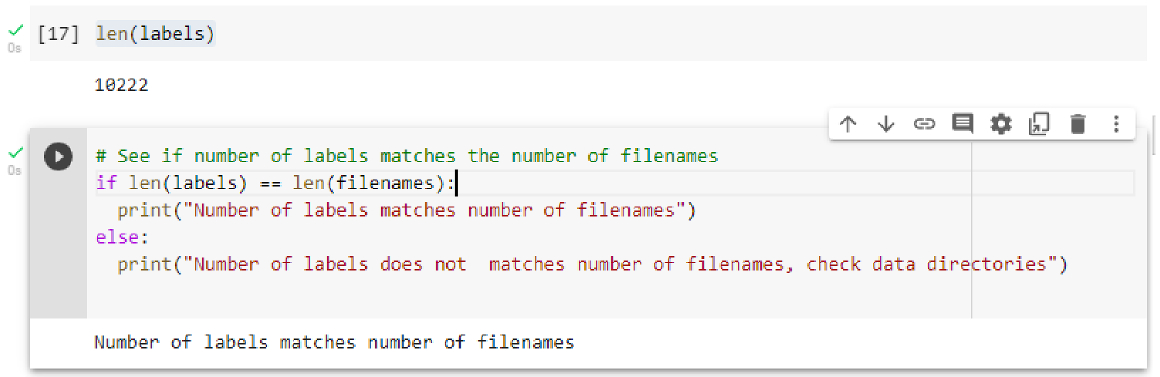


```
# Since we’ve now got our training image filepaths in a list, let’s prepare our labels.
import numpy as np
labels = label_csv["breed"].to_numpy()
#labels = np.array(labels) # does same thing as above
labels
```

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

```
len(labels)
```

```
# 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")
```



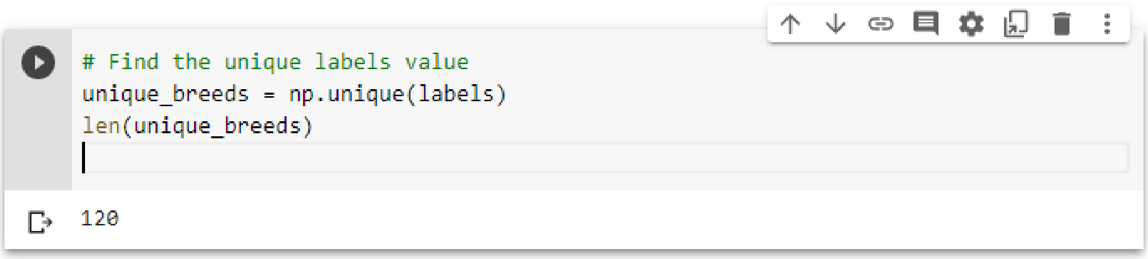
```
[17] len(labels)
```

```
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
```

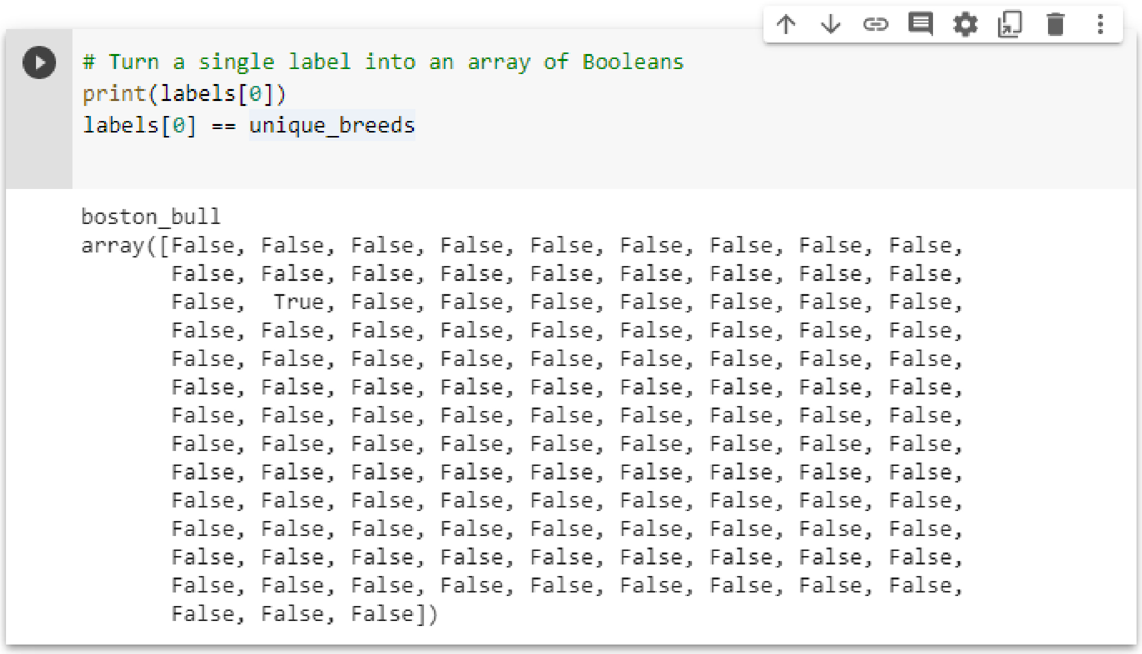
```
# Find the unique labels value
unique_breeds = np.unique(labels)
len(unique_breeds)
```



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

```
120
```

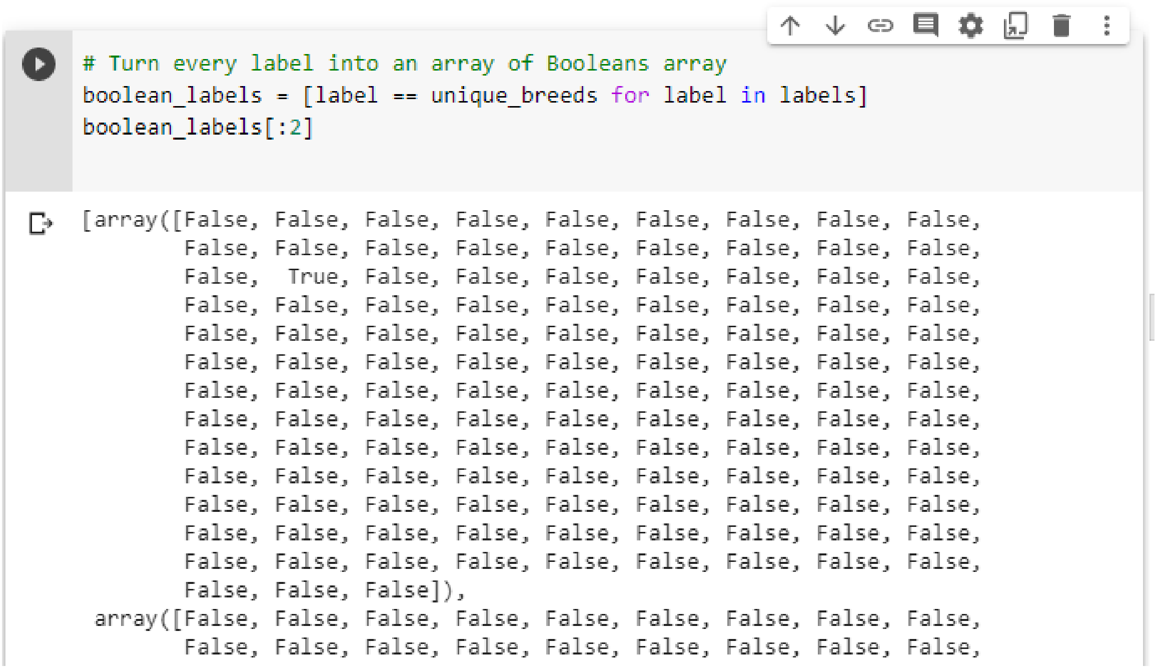
```
# Turn a single label into an array of Booleans
print(labels[0])
labels[0] == unique_breeds
```



```
# 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 an array of Booleans array
boolean_labels = [label == unique_breeds for label in labels]
boolean_labels[:2]
```

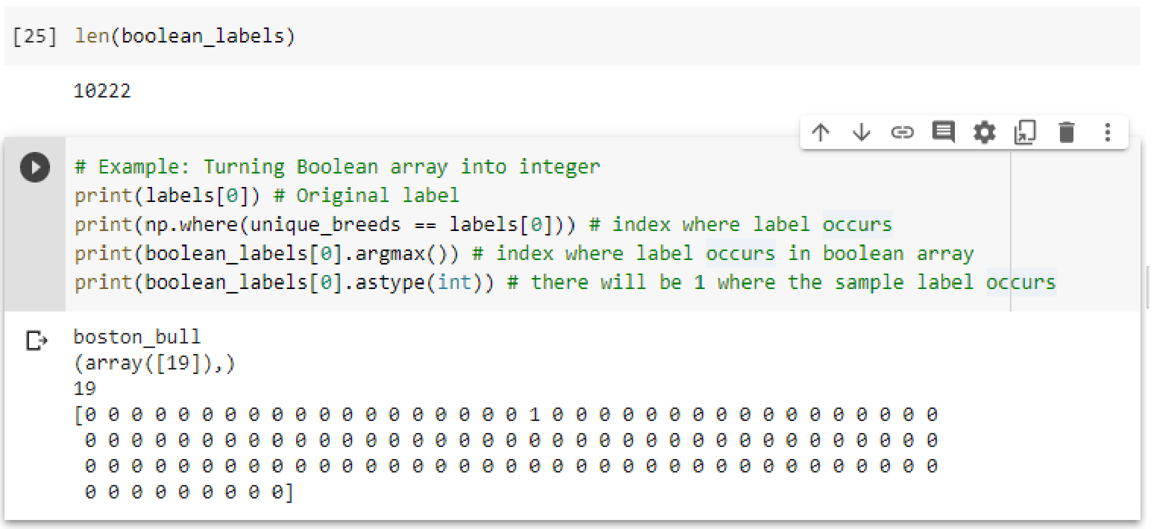


```
# Turn every label into an array of Booleans array
boolean_labels = [label == unique_breeds for label in labels]
boolean_labels[:2]
```

```
[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,
```

```
len(boolean_labels)
```

```
# Example: Turning Boolean array into integer
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 1 where the sample
 label occurs
```



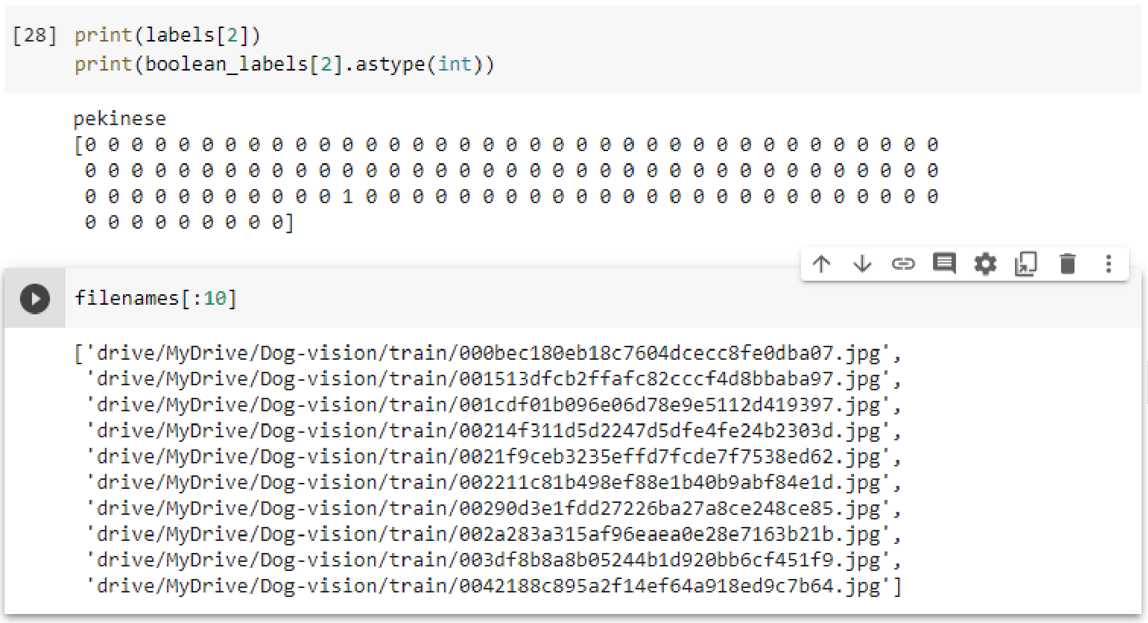
```
[25] len(boolean_labels)

10222
```

```
# Example: Turning Boolean array into integer
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 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 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]
```

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

```
filenames[:10]
```



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

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

```
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.

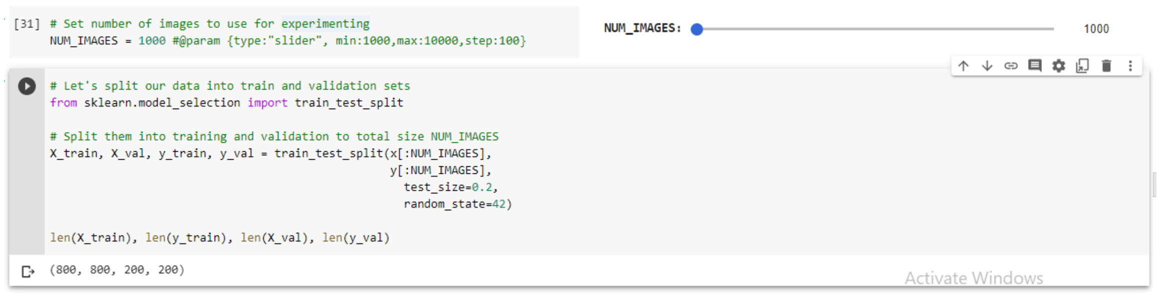
os

```
# setup x and y variables

x = filenames
y = boolean_labels
```

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:100}
```

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

# Split them into training and validation to 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)
```

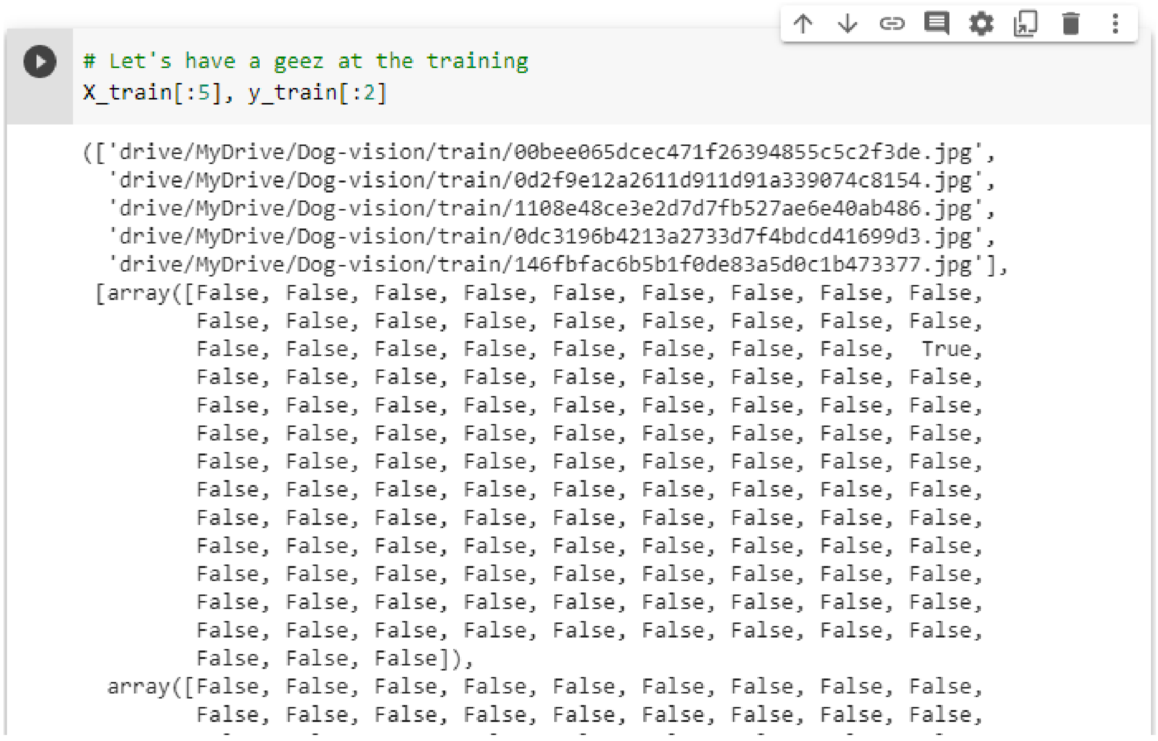


```
[31] # Set number of images to use for experimenting
     NUM_IMAGES = 1000 #@param {type:"slider", min:1000,max:10000,step:100}
```

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

# Split them into training and validation to 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)
```

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



```python
# Let's have a geez at the training
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, 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,
```

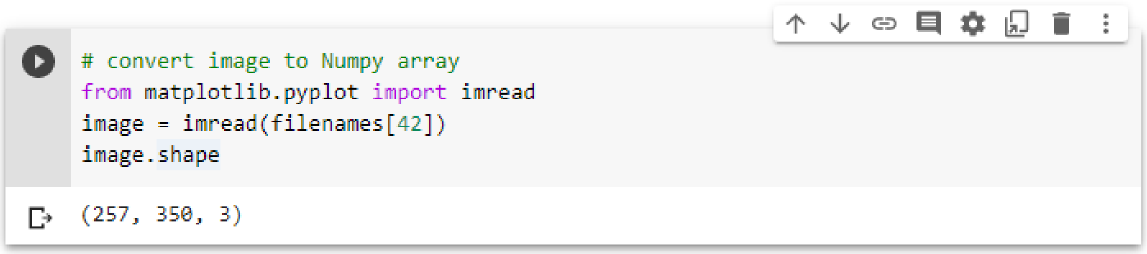
Preprocessing Images (turning images into Tensors)

To preprocess our images into Tensors we're going to write a function which does a few things:

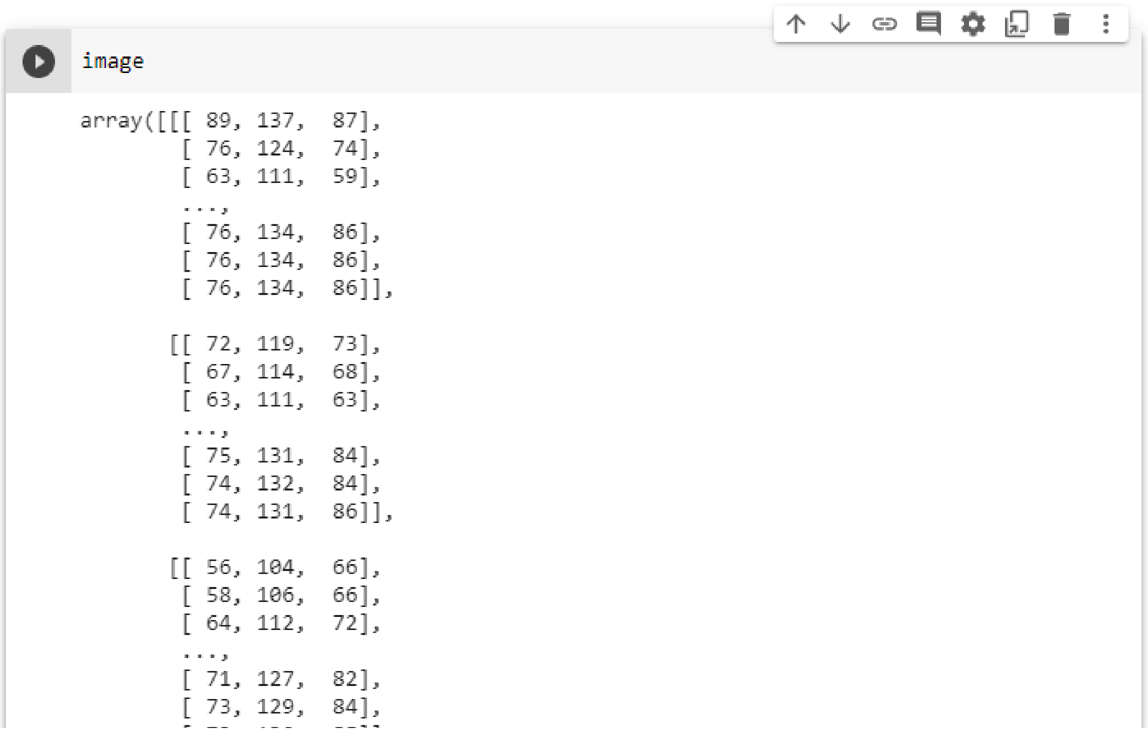
2. Take an `image` filepath as input
3. Use TensorFlow to read the file and save it to a variable, `image`
4. turn our `image` (a jpg) into Tensors
5. Resize the `image` to be a shape of (224,224)
6. Return the modified `image`

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

```python
# convert image to Numpy array
from matplotlib.pyplot import imread
image = imread(filenames[42])
image.shape
```


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

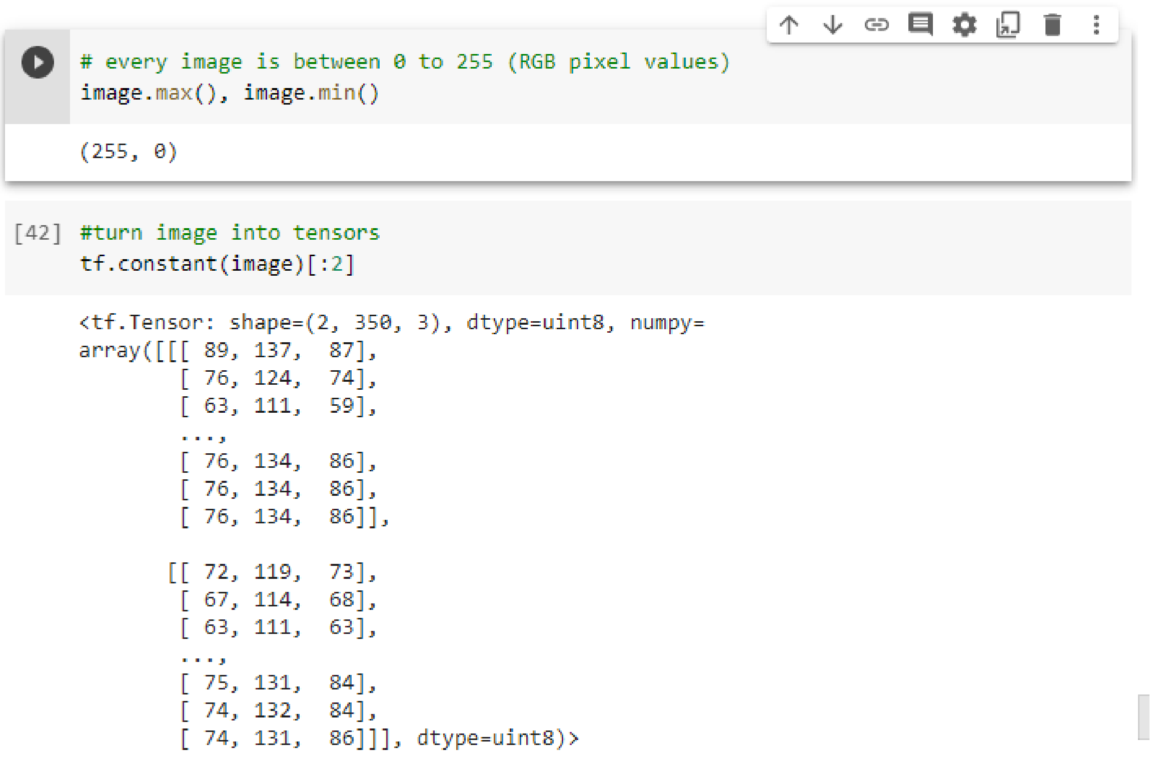
```
(257, 350, 3)
```


```
image
```

```
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]],

       [[ 56, 104,  66],
        [ 58, 106,  66],
        [ 64, 112,  72],
        ...,
        [ 71, 127,  82],
        [ 73, 129,  84],
```

```
# every image is between 0 to 255 (RGB pixel values)
image.max(), image.min()
```

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



```
# every image is between 0 to 255 (RGB pixel values)
image.max(), image.min()
```

```
(255, 0)
```

```
[42] #turn image into tensors
     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 pre process 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 to 255 to 0 to 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 preprocesing image

def process_image(image_path, img_size=IMG_SIZE):
  """
  Takes an image filepath 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 color channels (RG
B)
  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 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?

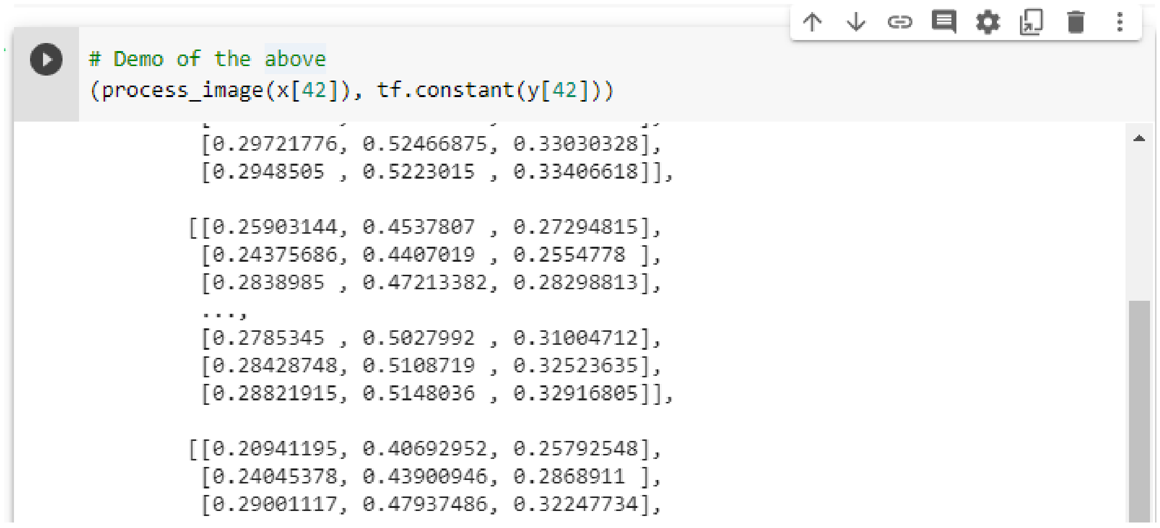
Let's say you're trying to process 10,000+ images in one go... they all might not fit into memory.

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 sinmple function to return a tuple `(image, label)`

def get_image_label(image_path, label):
  """
  Takes an image file path name and the assosciated label,
  processes the image and returns a tuple of image (image, label).
  """

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

```
# Demo of the above
(process_image(x[42]), tf.constant(y[42]))
```

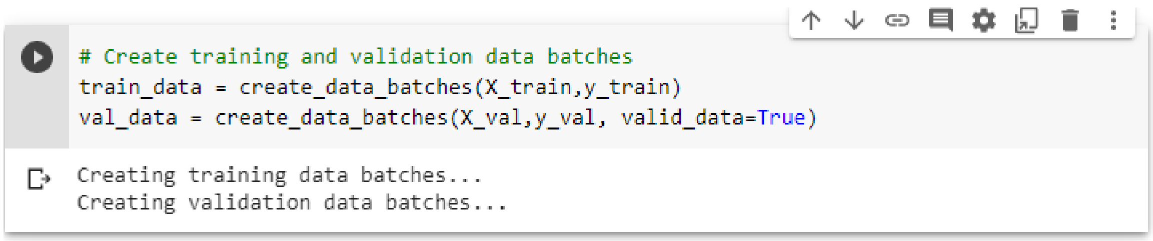

```
# Demo of the above
(process_image(x[42]), tf.constant(y[42]))
```

```
         [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],
```

Now we've got a way to turn our data into tuples of Tensors in the forn: `(image, label)`, let's make a function to turn all of our data (`x` and `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= F
alse, test_data=False): ####
  """
Creares batches of data out of image (x) and label (y) pairs.
Shufles the data if it's training data but doesn't shuffle if it's vali
dation data.
Also accepts test data as input (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 fi
lepaths (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), # filepa
ths
                                          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 maping image processor funct
ion is faster then shuffling images
    data = data.shuffle(buffer_size = len(x))

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

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

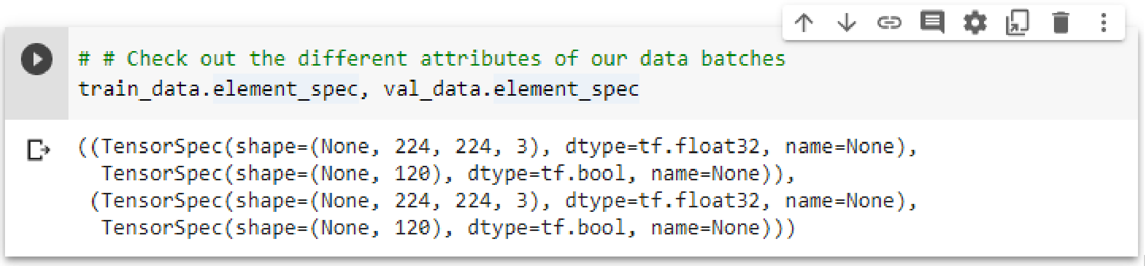
```
# Create training and validation data batches
train_data = create_data_batches(X_train,y_train)
val_data = create_data_batches(X_val,y_val, valid_data=True)
```



```
# 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
```



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

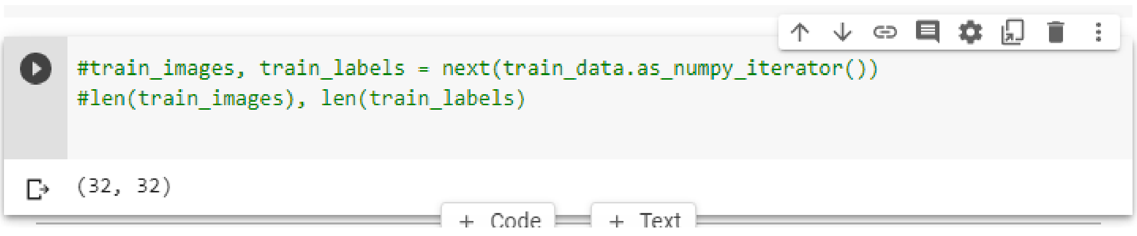
```
((TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 120), dtype=tf.bool, name=None)),
 (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 120), dtype=tf.bool, name=None)))
```

Visualizing data batches

our data in batches can be a bit of a hard concept to understand. Let's build a function which helps us visualize what's going on under the hood.

```
import matplotlib.pyplot as plt

# Create a function for viewing images in a data batch

def show_25_images(images, labels):
  """
  Displays 25 images 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 line off
    plt.axis("off")
```

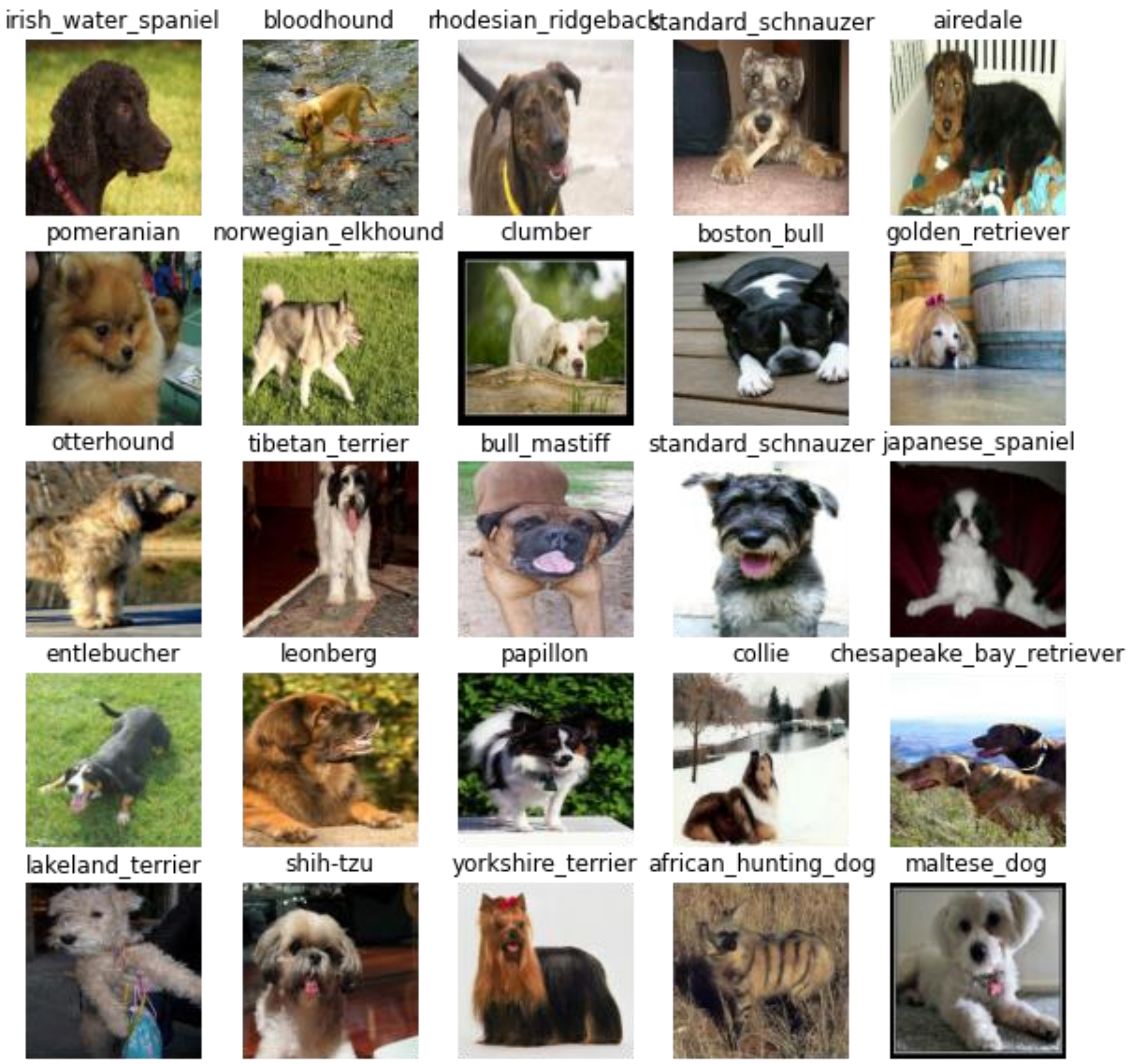
```
#train_images, train_labels = next(train_data.as_numpy_iterator())
#len(train_images), len(train_labels)
```



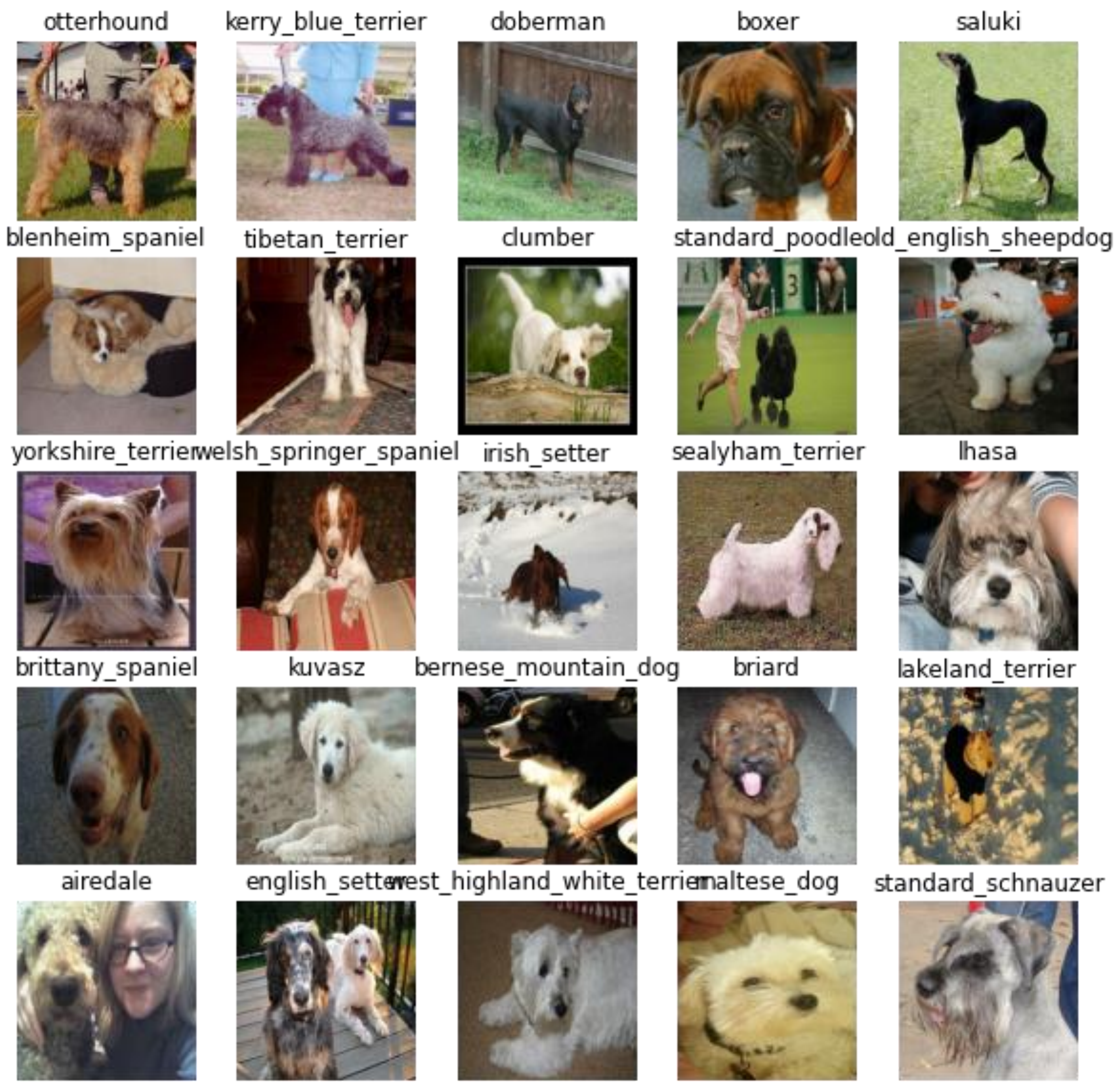
```
#train_images, train_labels = next(train_data.as_numpy_iterator())
#len(train_images), len(train_labels)
```

```
(32, 32)
```

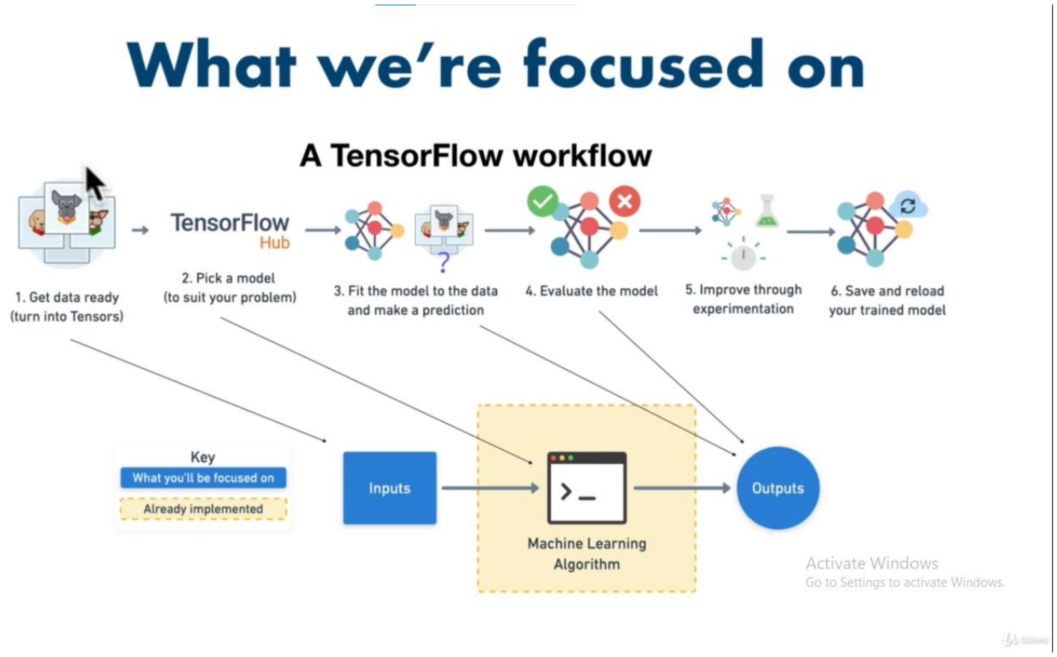
```
# Now let's visualize the data in a training batch
train_images, train_labels = next(train_data.as_numpy_iterator())
show_25_images(train_images, train_labels)
```



```
# Now let's Visualize our validation set
val_images, val_labels = next(train_data.as_numpy_iterator())
show_25_images(val_images, val_labels)
```



Preparing Our Inputs and Outputs


What we're focused on
A TensorFlow workflow
1. Get data ready (turn into Tensors)
TensorFlow Hub
2. Pick a model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
Key
What you'll be focused on
Already implemented
Inputs
Machine Learning Algorithm
Outputs
Activate Windows
Go to Settings to activate Windows.

Creating and training a model

Now our data is ready, let's prepare it modelling. We'll use an existing model from TensorFlow Hub.

TensorFlow Hub is a resource where you can find pretrained machine learning models for the problem you're working on.
Using a pretrained machine learning model is often referred to as **transfer learning**.

Why use a pretrained model?

Building a machine learning model and training it on lots from scratch can be expensive and time consuming.

Transfer learning helps eliviate some of these by taking what another model has learned and using that information with your own problem.

How do we choose a model?
Since we know our problem is image classification (classifying different dog breeds), we can navigate the TensorFlow Hub page by our problem domain (image).
We start by choosing the image problem domain, and then can filter it down by subdomains, in our case, image classification.

Doing this gives a list of different pretrained models we can apply to our task.

Clicking on one gives us information about the model as well as instructions for using it. For example, clicking on the mobilenet_v2_130_224 model, tells us this model takes an input of images in the shape 224, 224. It also says the model has been trained in the domain of image classification.

Let's try it out.

Building a model

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

- The input shape (images, 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.

These things will be standard practice with whatever machine learning model you use. And because we're using TensorFlow, everything will be in the form of Tensors.

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

# Setup output shape to the model
OUTPUT_SHAPE = len(unique_breeds)
# Setup model URL from TensorFlow Hub
MODEL_URL = "https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/cla
ssification/4"
```

Now we've got the inputs, outputs and model we're using ready to go. We can start to put them together into a **Keras** deep learning model

There are many ways of building a model in TensorFlow but one of the best ways to get started is to use the Keras API.

Defining a deep learning model in Keras can be as straightforward as saying, "here are the layers of the model, the input shape and the output shape, let's go!"

Knowing this, let's create a function which:

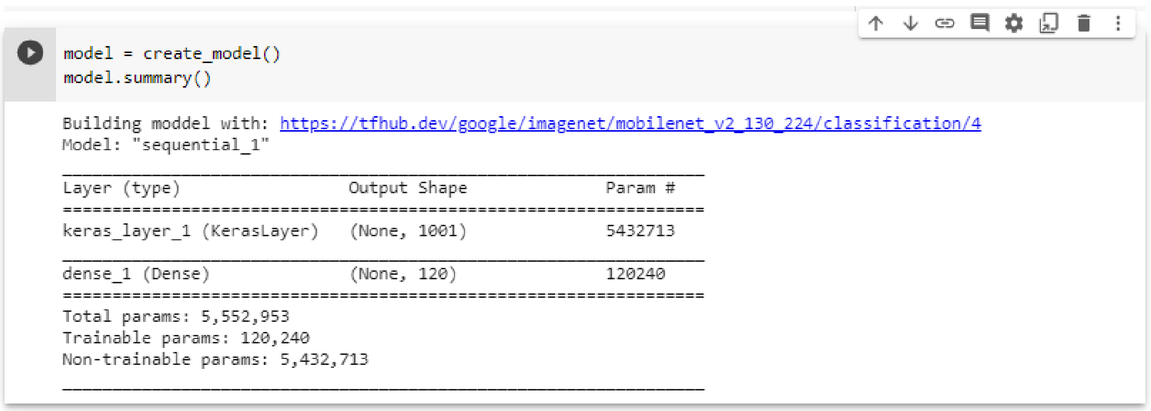
- Takes the input shape, output shape and the model we've chosen's URL as parameters.
- Defines the layers in a Keras model in a sequential fashion (do this first, then this, then that).
- Compiles the model (says how it should be evaluated and improved).
- Builds the model (tells it what kind of input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

All of these steps can be found here: www.tensorflow.org/guide/keras/overview

We'll take a look at the code first, then dicuss each part.

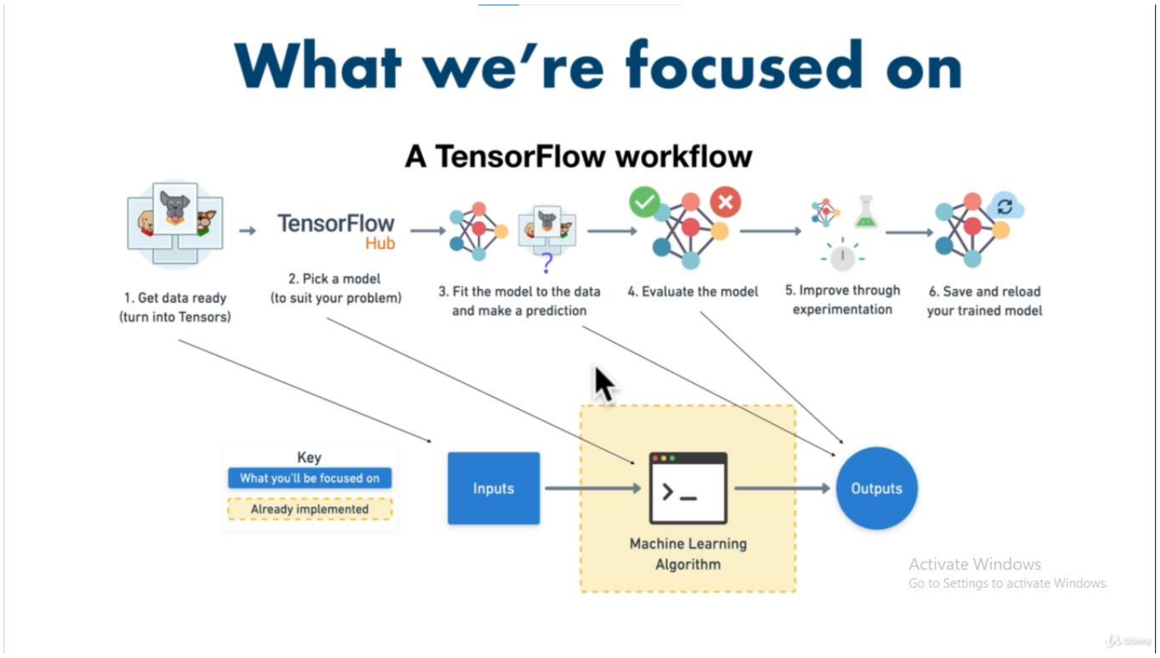
```
# Create a function which builds a Keras model

def create_model(shape=INPUT_SHAPE, output_shape=OUTPUT_SHAPE, model_ur
l=MODEL_URL):
  print("Building moddel 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") #Lay
er 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()
```



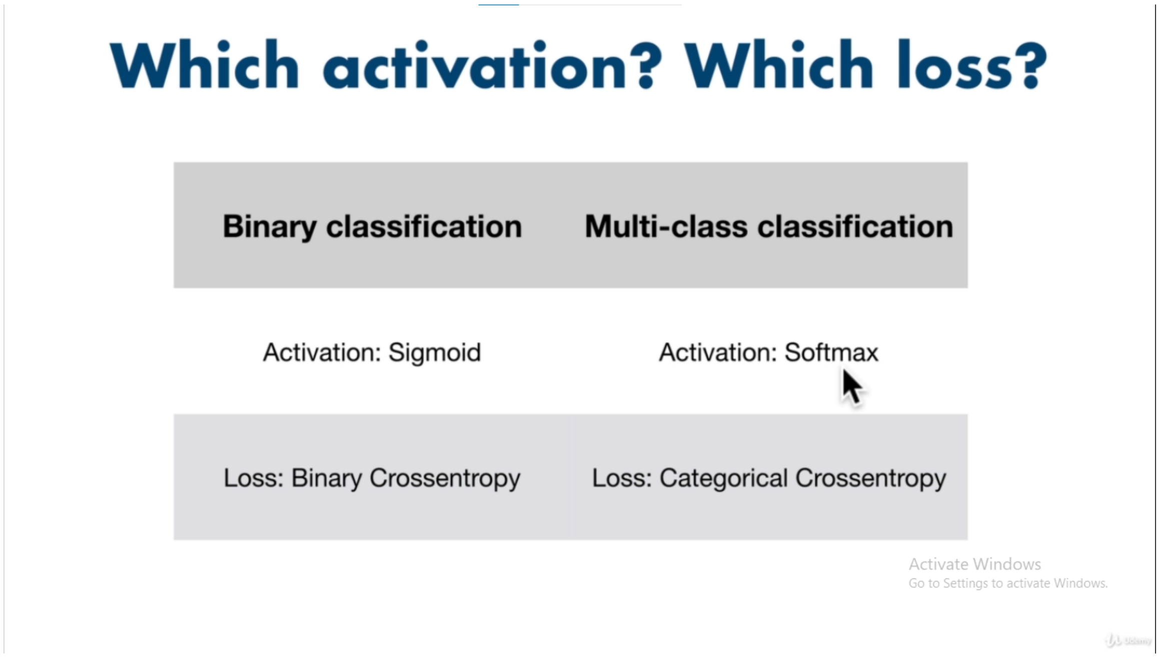
```
model = create_model()
model.summary()
```

```
Building moddel with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/4
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer_1 (KerasLayer)   (None, 1001)              5432713
_________________________________________________________________
dense_1 (Dense)              (None, 120)               120240
=================================================================
Total params: 5,552,953
Trainable params: 120,240
Non-trainable params: 5,432,713
_________________________________________________________________
```



What we're focused on

A TensorFlow workflow
TensorFlow Hub
1. Get data ready (turn into Tensors)
2. Pick a model (to suit your problem)
3. Fit the model to the data and make a prediction
4. Evaluate the model
5. Improve through experimentation
6. Save and reload your trained model
Key
What you'll be focused on
Already implemented
Inputs
Machine Learning Algorithm
Outputs



Which activation? Which loss?

Binary classification	Multi-class classification
Activation: Sigmoid	Activation: Softmax
Loss: Binary Crossentropy	Loss: Categorical Crossentropy