```
# Import necessary tools
# Import TensorFlow into Colab
import tensorflow as tf
import tensorflow_hub as hub
# Checkout the labels of our data
import pandas as pd
labels_csv = pd.read_csv("drive/MyDrive/Dog Vision/labels.csv")
# Create pathnames from image ID's
filenames = ["drive/MyDrive/Dog Vision/train/"+fname+".jpg" for fname in labels_csv["id"]]
import numpy as np
labels = labels_csv["breed"].to_numpy()
# Find the unique label values
unique_breeds = np.unique(labels)
# Turn every label into a boolean array
boolean_labels= [label == unique_breeds for label in labels]
# Setup X & y variables
X=filenames
y=boolean_labels


# 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

# 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

# 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


# 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"

# 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

# Load TensorBoard notebook extension
%load_ext tensorboard

import datetime

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

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

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


# 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)]

import os
# 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"
```

```
  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

# Create a data batch with the full data set
full_data = create_data_batches(X,y)
# Create a model for full model
full_model = create_model()
# 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])
save_model(full_model,suffix="full-image-set-mobilenetv2-Adam")
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Creating training data batches...
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classific
Epoch 1/100
320/320 [==============================] - 1849s 6s/step - loss: 1.3329 - accuracy: 6
Epoch 2/100
320/320 [==============================] - 589s 2s/step - loss: 0.4017 - accuracy: 0
Epoch 3/100
320/320 [==============================] - 589s 2s/step - loss: 0.2369 - accuracy: 0
Epoch 4/100
320/320 [==============================] - 574s 2s/step - loss: 0.1535 - accuracy: 0
Epoch 5/100
320/320 [==============================] - 576s 2s/step - loss: 0.1056 - accuracy: 0
Epoch 6/100
320/320 [==============================] - 581s 2s/step - loss: 0.0779 - accuracy: 0
Epoch 7/100
320/320 [==============================] - 586s 2s/step - loss: 0.0571 - accuracy: 0
Epoch 8/100
320/320 [==============================] - 574s 2s/step - loss: 0.0462 - accuracy: 0
Epoch 9/100
320/320 [==============================] - 576s 2s/step - loss: 0.0367 - accuracy: 0
Epoch 10/100
320/320 [==============================] - 572s 2s/step - loss: 0.0312 - accuracy: 0
Epoch 11/100
320/320 [==============================] - 578s 2s/step - loss: 0.0265 - accuracy: 0
Epoch 12/100
320/320 [==============================] - 572s 2s/step - loss: 0.0222 - accuracy: 0
Epoch 13/100
320/320 [==============================] - 583s 2s/step - loss: 0.0188 - accuracy: 0
Epoch 14/100
320/320 [==============================] - 582s 2s/step - loss: 0.0180 - accuracy: 0
Epoch 15/100
320/320 [==============================] - 590s 2s/step - loss: 0.0160 - accuracy: 0
Epoch 16/100
320/320 [==============================] - 578s 2s/step - loss: 0.0159 - accuracy: 0
Saving model to: drive/MyDrive/Dog Vision/models/20230711-09531689069203-full-image-s
```