**Image Recognition using Natural Language Processing**

## **1.Pre-trained image**

TensorFlow is an open-source library developed by Google, which supports machine learning and deep learning. TensorFlow Hub has a variety of pre-trained, ready-to-deploy models such as image, text, video, and audio  classification.

tf.keras.applications.Xception(

    include\_top=True,

    weights="imagenet",

    input\_tensor=None,

    input\_shape=None,

    pooling=None,

    classes=1000,

    classifier\_activation="softmax",

)

**Example 1:**

model = tf.keras.Sequential([

    embed,

    tf.keras.layers.Dense(16, activation="relu"),

    tf.keras.layers.Dense(1, activation="sigmoid"),

])

## **2.Fine-tuning**

 The second strategy is Fine-tuning, or what I like to call network surgery. We spoke about Fine-tuning above, where it is built on making “fine” adjustments to a process to obtain the desired output to further improve performance. Fine-tuning is also seen as a further step to Feature Extraction. In fine-tuning we freeze certain layers and selectively re-train some to improve its accuracy, gaining a higher performance at a lower learning rate and requiring less training time.

Fine-tuning is vital if you want to be able to obtain efficient feature representation from the base model, thus being more catered to your task at hand.

**3.Load the data**

So let's first make sure we import the libraries that we need and load the Data.

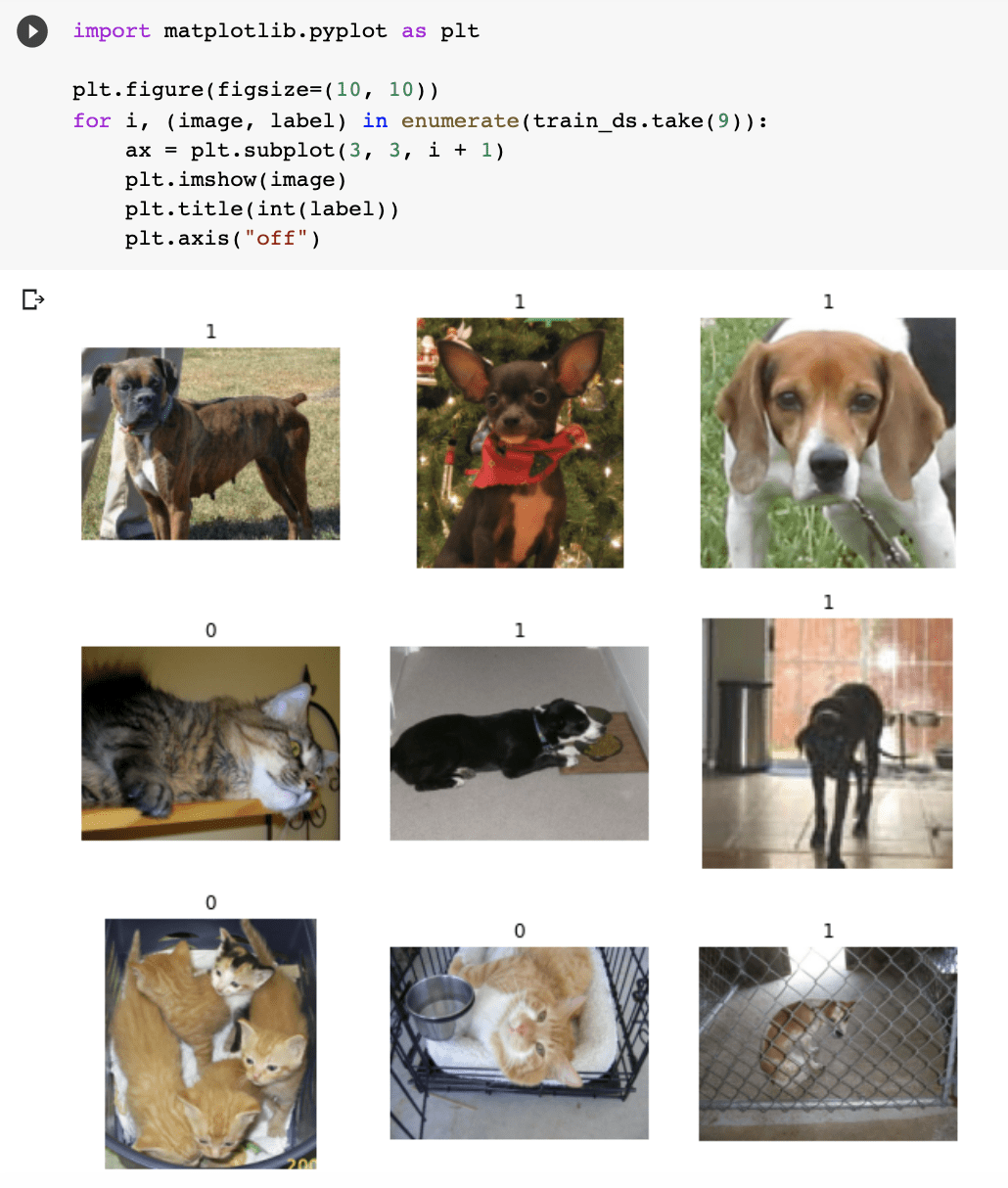
import numpy as np

import tensorflow as tf

from tensorflow import keras

import tensorflow\_datasets as tfds

It is always good to print out your sample size for your training and test to understand how much data you are working with and have a gander with the images, so you can have a look at what data you’re working with.



Due to images varying in size, it is a good approach to standardise to fixed image size, as it's a good consistent input for the neural network.

## **4.Data pre-processing**

## Now let’s go into Data Augmentation. When working with a smaller dataset, it is good practice to apply random transformation to the training images, such as horizontal flipping. Flipping means rotating an image on a vertical or horizontal axis. This will help expose the model to different angles and aspects of the training data, reducing overfitting.

from tensorflow import keras

from tensorflow.keras import layers

data\_augmentation = keras.Sequential(

  [layers.RandomFlip("horizontal"), layers.RandomRotation(0.1),]

)

## **5.base\_model**

base\_model = keras.applications.Xception(

  weights="imagenet",  # Weights pre-trained on ImageNet.

  input\_shape=(150, 150, 3),

  include\_top=False,

)

base\_model.trainable = False

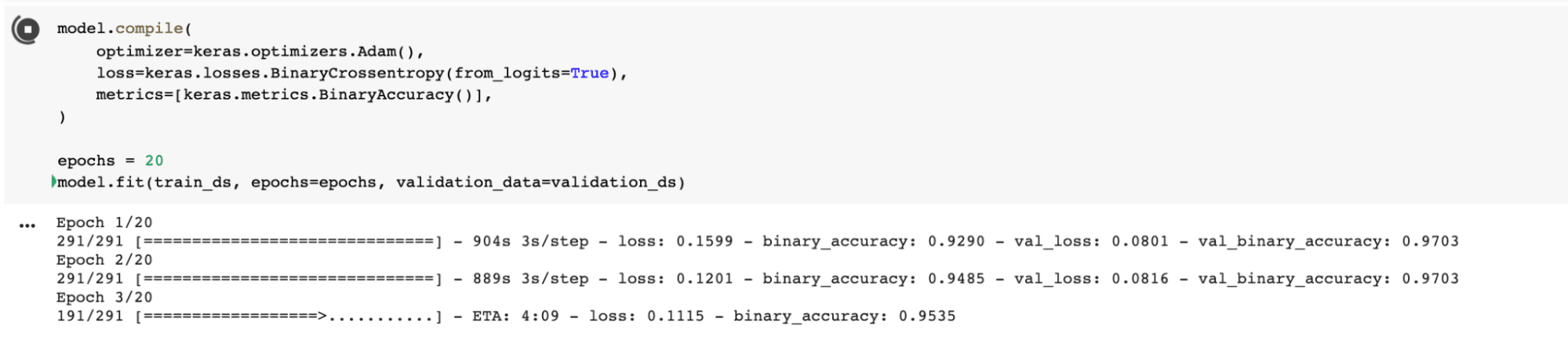
## **6.Train the top layer**

The next step is creating a new layer on top of the frozen layers, which will learn its knowledge of the old features and use that to determine predictions on the new dataset. This is beneficial as explained further up in the steps of transfer learning, the likelihood that the current output on the pre-trained model and the output you want from your model will be different is high, therefore adding new layers will improve the model overall.

# Create a new model on top

inputs = keras.Input(shape=(150, 150, 3))

x = data\_augmentation(inputs)  # Apply random data augmentation



## **7.Fine-tuning**

## So after we have run the model with frozen layers, we need to run the model with the base models unfrozen, which essentially improves the model, with a lower learning rate. You want to reduce the amount of overfitting, so let’s take this step slowly and unfreeze the base model.

base\_model.trainable = True

model.compile(

  optimizer=keras.optimizers.Adam(1e-5),  # Low learning rate

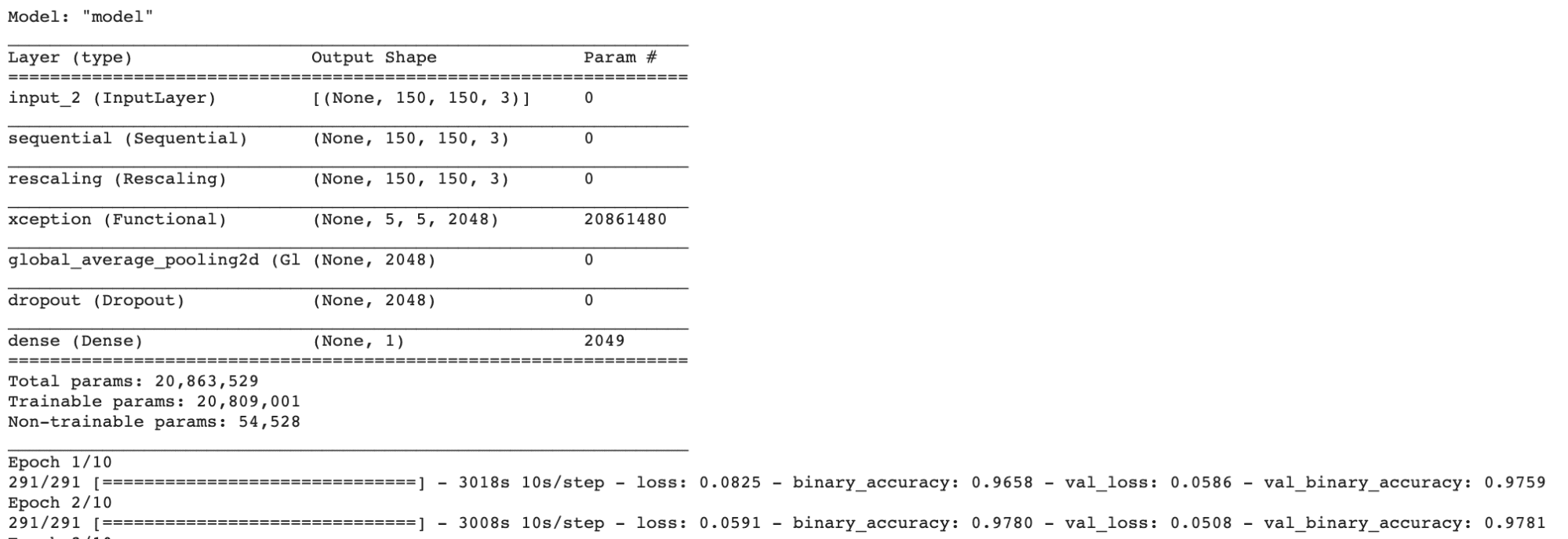
  loss=keras.losses.BinaryCrossentropy(from\_logits=True),

  metrics=[keras.metrics.BinaryAccuracy()],

)

epochs = 10

model.fit(train\_ds, epochs=epochs, validation\_data=validation\_ds)



## **8.Natural Language Processing**

NLP is the ability of a computer to be able to detect and understand human language, through speech and text just the way we humans can. The human language contains many ambiguities that cause difficulty in being able to create software that can accurately detect speech and text.

## HuggingFace

Let’s look into [HuggingFace](https://huggingface.co/). HuggingFace is an open-source provider of natural language processing (NLP) which has done an amazing job to make it user-friendly.

Their Transformers library is a python-based library that provides architectures such as BERT, that perform NLP tasks such as text classification and question answering. All you have to do is load their pre-trained models with just a few lines of code and you are ready to start experimenting. To get started with Transformers, you will need to install it.

Below is an example of using sentiment analysis, which is the ability to be able to identify an opinion expressed in a piece of text.

! pip install transformers

from transformers import pipeline

classifier = pipeline('sentiment-analysis')

classifier('I am finding the article about Transfer learning very useful.')

Output:

[{'label': 'POSITIVE', 'score': 0.9968850016593933}]

url = "<https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz>"

dataset = tf.keras.utils.get\_file("aclImdb\_v1.tar.gz", url,

                                untar=True, cache\_dir='.',

                                cache\_subdir='')

dataset\_dir = os.path.join(os.path.dirname(dataset), 'aclImdb')

os.listdir(dataset\_dir)

# Embed a 1,000-word vocabulary into 5 dimensions.

embedding\_layer = tf.keras.layers.Embedding(1000, 5)

# text vectorization layer to split, and map strings to integers.

vectorize\_layer = TextVectorization(

  standardize=custom\_standardization,

  max\_tokens=vocab\_size,

  output\_mode='int',

  output\_sequence\_length=sequence\_length)

model = Sequential([

vectorize\_layer,

Embedding(vocab\_size, embedding\_dim, name="embedding"),

GlobalAveragePooling1D(),

Dense(16, activation='relu'),

Dense(1)

])

model.compile(optimizer='adam',

            loss=tf.keras.losses.BinaryCrossentropy(from\_logits=True),

            metrics=['accuracy'])