DATA 586 Project

Application of Deep Learning and Convolutional Neural Network to classify and identify Animals

Group Member: Shreeram Murali, Harry Sun

Main Objective:

1. Getting familiar with Image Dataset and Using this dataset with CNN.
2. Training CNN for image classification and animal detection.

Step One: Dataset of Images.

The first step is to gather the data. This in my opinion, will be the most difficult and annoying aspect of the project. Remember that the data must be labeled. If the dataset is not labeled, this can be be time consuming as you would have to manually create new labels for each categories of images. We are making use of Google Colab with mounted Google Drive with all the 4500 images approximately. Now that we have our datasets stored safely in our computer or cloud, let's make sure we have a training data set, a validation data set, and a testing data set.

Training data set contains about 85-90% of our image data with labels. This data would be used to train our machine about the different types of images we have. Validation data set contains about 15-10% of our image data with labels This will test how well our machine performs against known labeled data. Testing data set contains random image data without labels. This testing data will be used to test how well our machine can classify data it has never seen. The testing data can also just contain images from Google that you have downloaded, if it makes sense to the topic you are classifying.

The Dataset consists of training set, validation set and testing set. The training set contains 4000 images divided into 10 folders with animal names to distinguish them, with 400 images in each folder of the animal. The validation set consists of 400 images in 10 different folders with animal names and each folder contain 40 images. The testing data contains 10 images of animal which are not part of our classification labels and are selected at random.

Insight into Data:

The images are divided into 10 classes of different species of animals.

The classes are as follows:

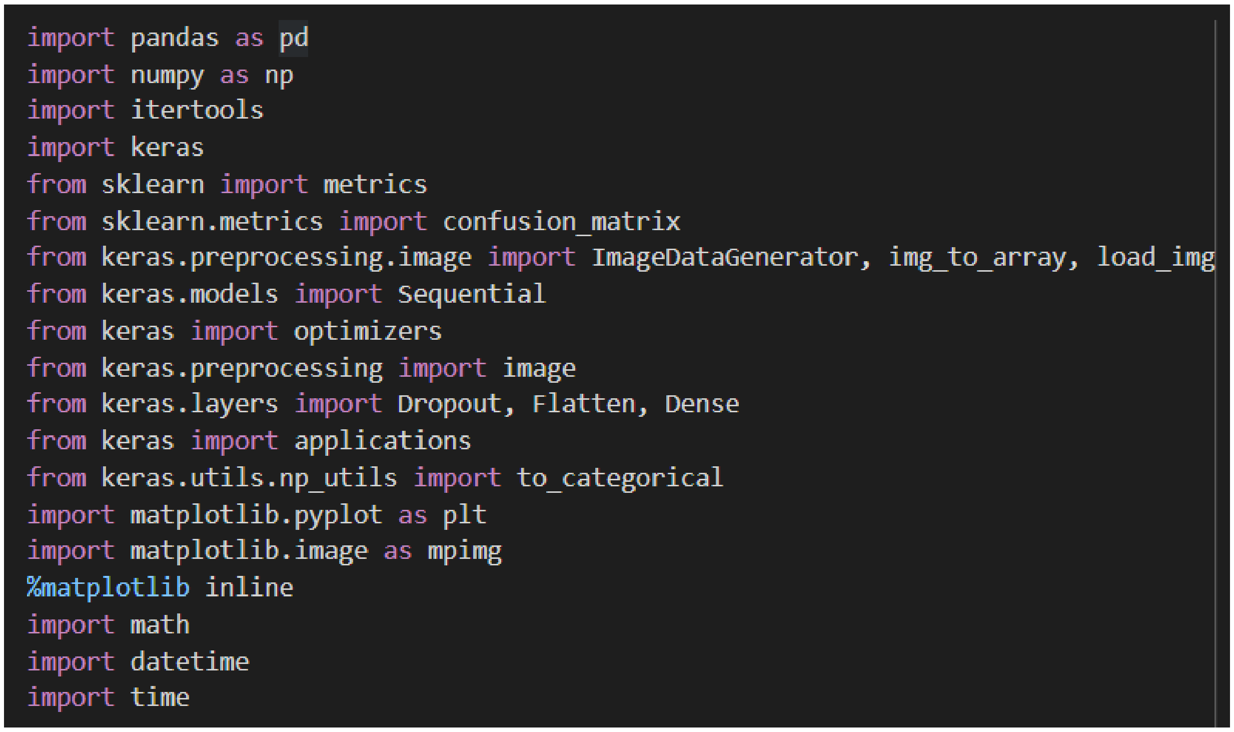
- Butterfly
- Cat
- Chicken
- Cow
- Dog
- Elephant
- Hose
- Sheep
- Spider
- Squirrel

Data Sample:

Test data

Step Two: Import all the necessary libraries.

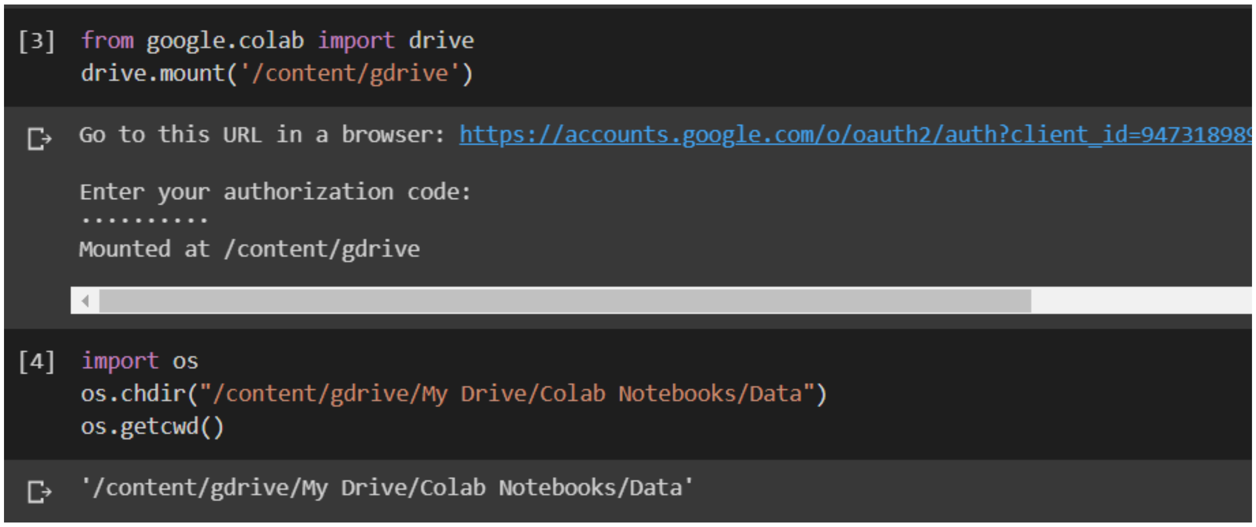
The picture below shows all the necessary packages we have used in this project.



```python
import pandas as pd
import numpy as np
import itertools
import keras
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from keras.models import Sequential
from keras import optimizers
from keras.preprocessing import image
from keras.layers import Dropout, Flatten, Dense
from keras import applications
from keras.utils.np_utils import to_categorical
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
import math
import datetime
import time
```

Step Three: Image Dimensions and Locating images dataset on Google Drive.

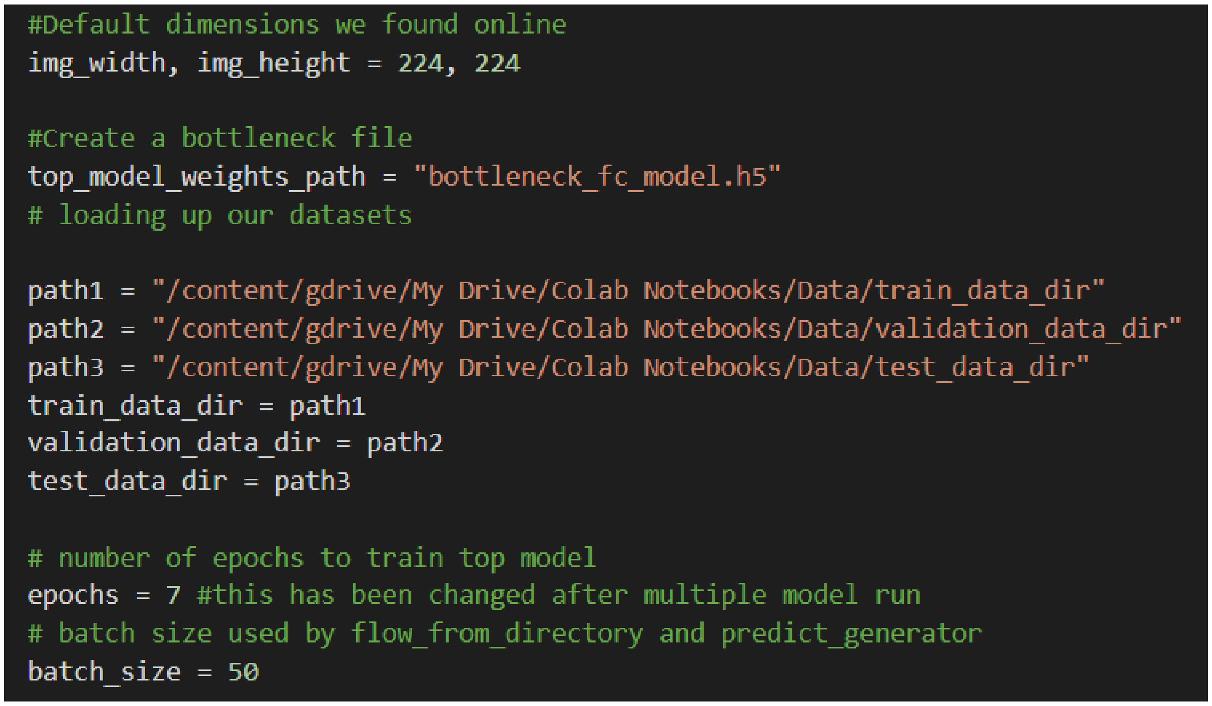
We have the dataset stored in Google Drive and we can mount Google Drive in Google Colab and make use of it.



```
[3]  from google.colab import drive
     drive.mount('/content/gdrive')

     Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=94731898

     Enter your authorization code:
     ··········
     Mounted at /content/gdrive

[4]  import os
     os.chdir("/content/gdrive/My Drive/Colab Notebooks/Data")
     os.getcwd()

     '/content/gdrive/My Drive/Colab Notebooks/Data'
```

The above picture shows the code we have used to mount the drive on to colab and get the folder path we want; we use this path to retrieve out data set from drive.

For image dimension, I am using 224*224 dimension as 224*224 works best in our case. Along with the image dimension we also create bottleneck file system.

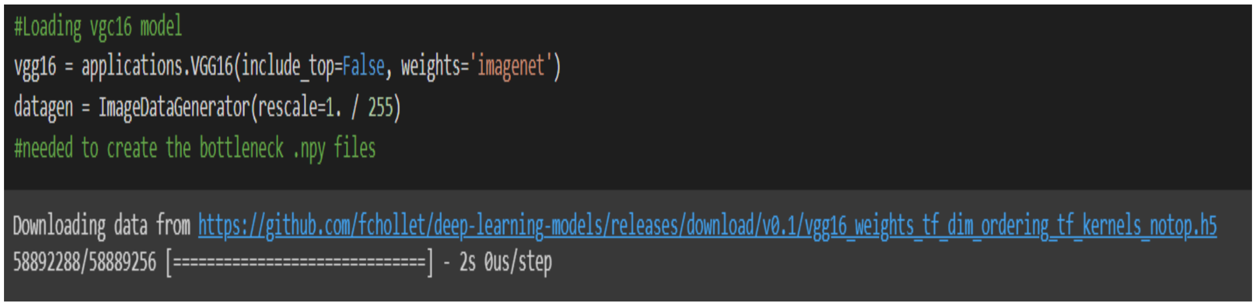


```
#Default dimensions we found online
img_width, img_height = 224, 224

#Create a bottleneck file
top_model_weights_path = "bottleneck_fc_model.h5"
# loading up our datasets

path1 = "/content/gdrive/My Drive/Colab Notebooks/Data/train_data_dir"
path2 = "/content/gdrive/My Drive/Colab Notebooks/Data/validation_data_dir"
path3 = "/content/gdrive/My Drive/Colab Notebooks/Data/test_data_dir"
train_data_dir = path1
validation_data_dir = path2
test_data_dir = path3

# number of epochs to train top model
epochs = 7 #this has been changed after multiple model run
# batch size used by flow_from_directory and predict_generator
batch_size = 50
```

Here we also give the path of training data folder, validation data folder and testing data folder. We specify the epochs and batch size parameters as well to train out model.

Step Four: Importing transfer learning model VGG16.

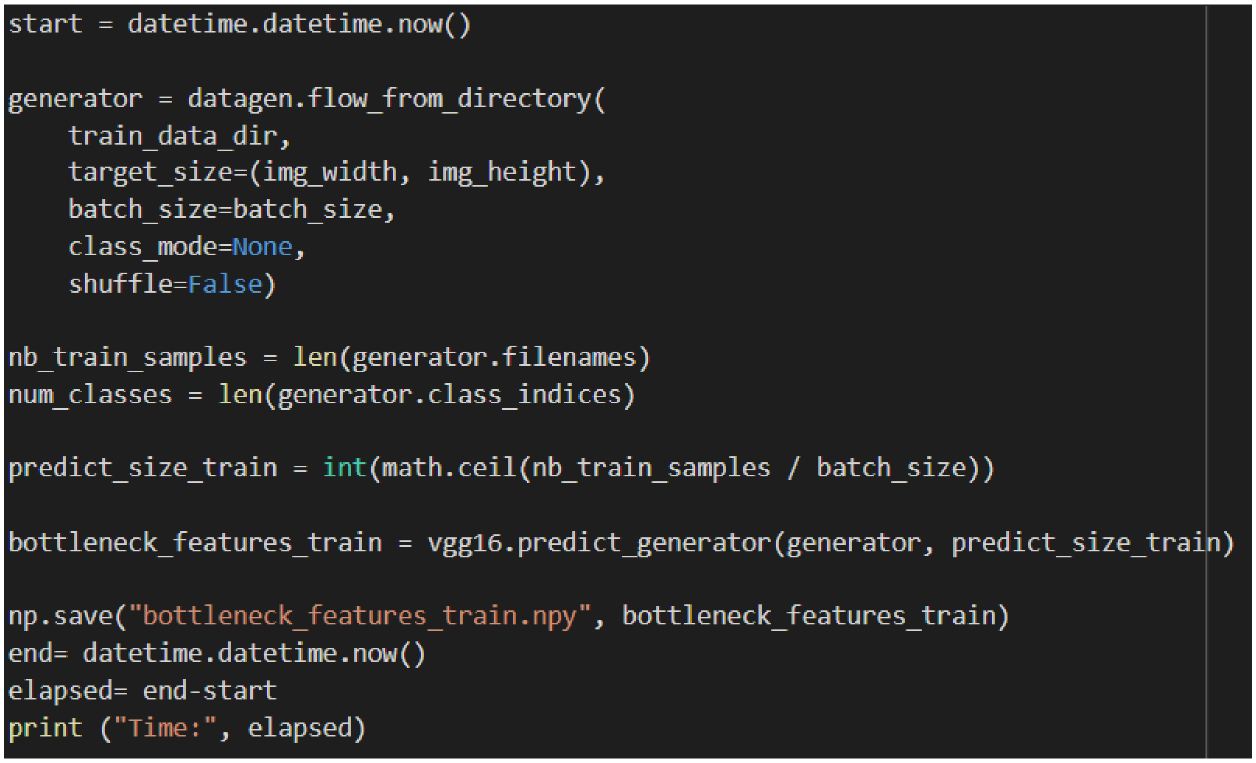
This is a very important step because transferring learning is very handy because it comes with already pre-made neural networks and other important components. The VGG16 model comes with 11 convolutional layers and it’s very easy to work with. There are many other transfer learning models which can be used such as resnet50 which comes with 50 neural layers.



```
#Loading vgc16 model
vgg16 = applications.VGG16(include_top=False, weights='imagenet')
datagen = ImageDataGenerator(rescale=1. / 255)
#needed to create the bottleneck .npy files
```

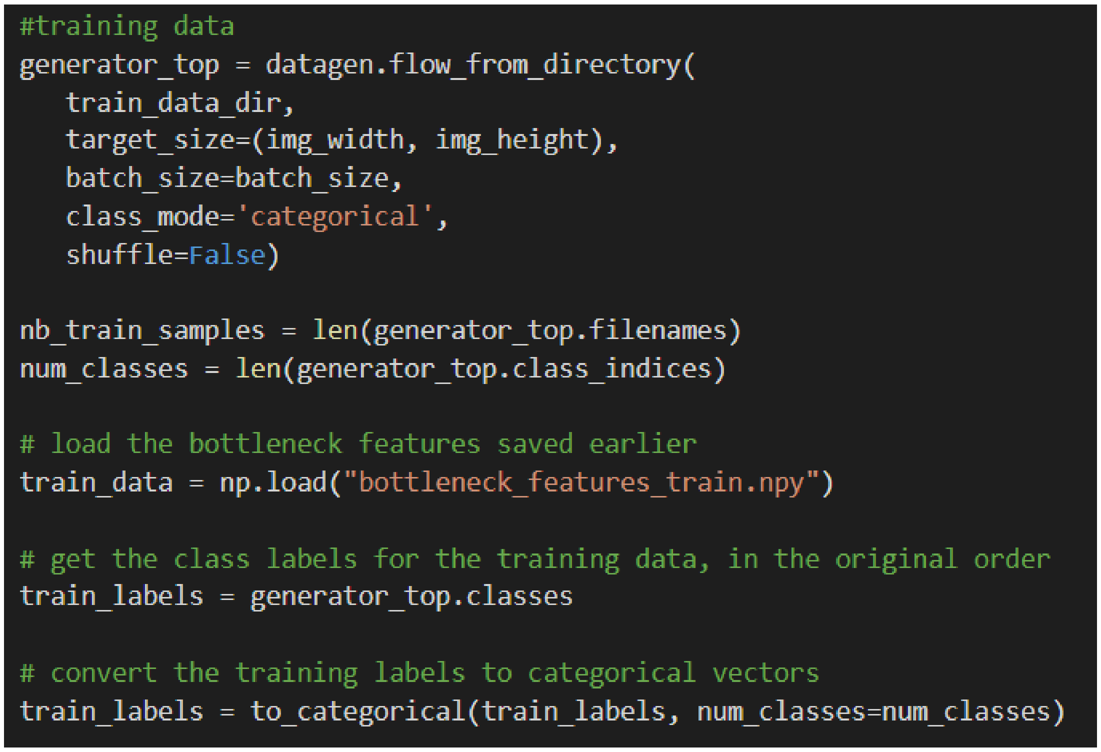
```
Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 2s 0us/step
```

Step Five: Weight and Feature using VGG16 Model

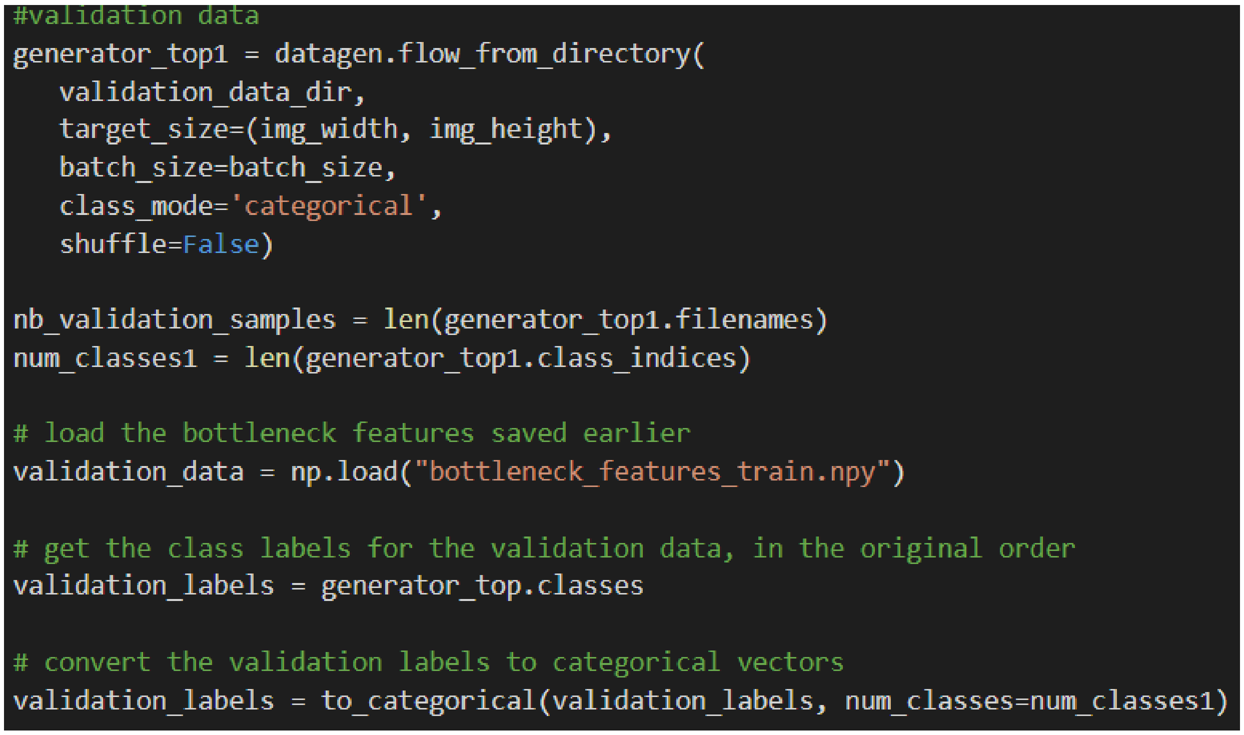


```
start = datetime.datetime.now()

generator = datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode=None,
    shuffle=False)

nb_train_samples = len(generator.filenames)
num_classes = len(generator.class_indices)

predict_size_train = int(math.ceil(nb_train_samples / batch_size))

bottleneck_features_train = vgg16.predict_generator(generator, predict_size_train)

np.save("bottleneck_features_train.npy", bottleneck_features_train)
end= datetime.datetime.now()
elapsed= end-start
print ("Time:", elapsed)
```

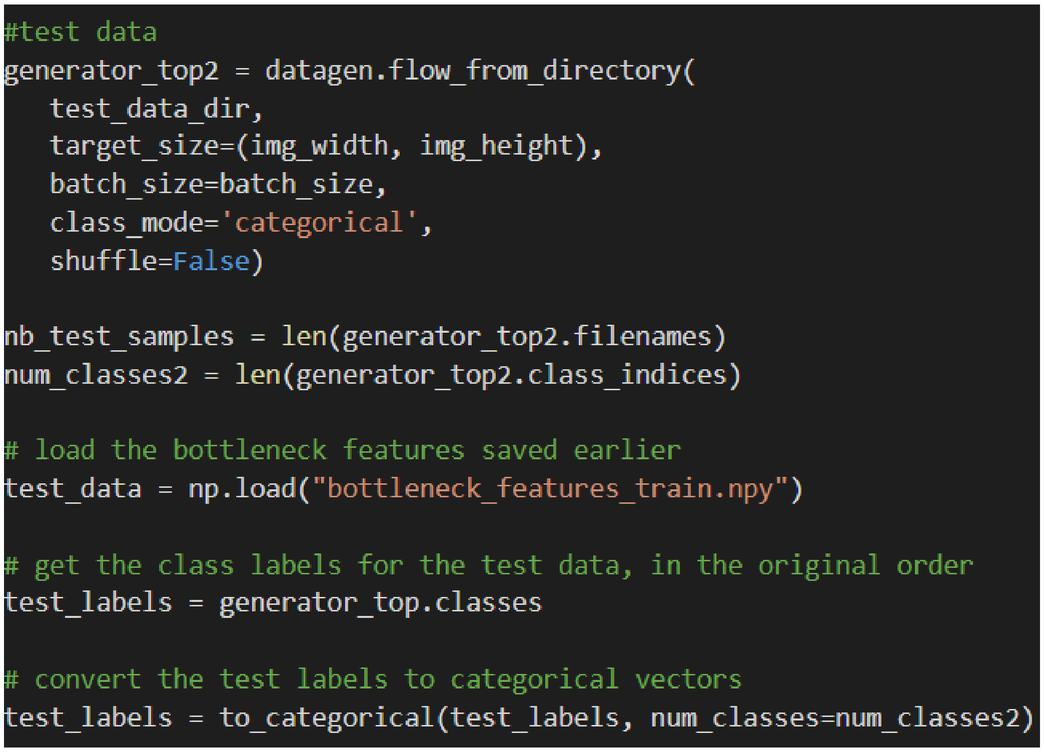
Here we are crating a simple image classifier and we perform this to all 3 datasets. Once this is done, we can this bottleneck file for our convolution neural network.



```
#training data
generator_top = datagen.flow_from_directory(
   train_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_train_samples = len(generator_top.filenames)
num_classes = len(generator_top.class_indices)

# load the bottleneck features saved earlier
train_data = np.load("bottleneck_features_train.npy")

# get the class labels for the training data, in the original order
train_labels = generator_top.classes

# convert the training labels to categorical vectors
train_labels = to_categorical(train_labels, num_classes=num_classes)
```

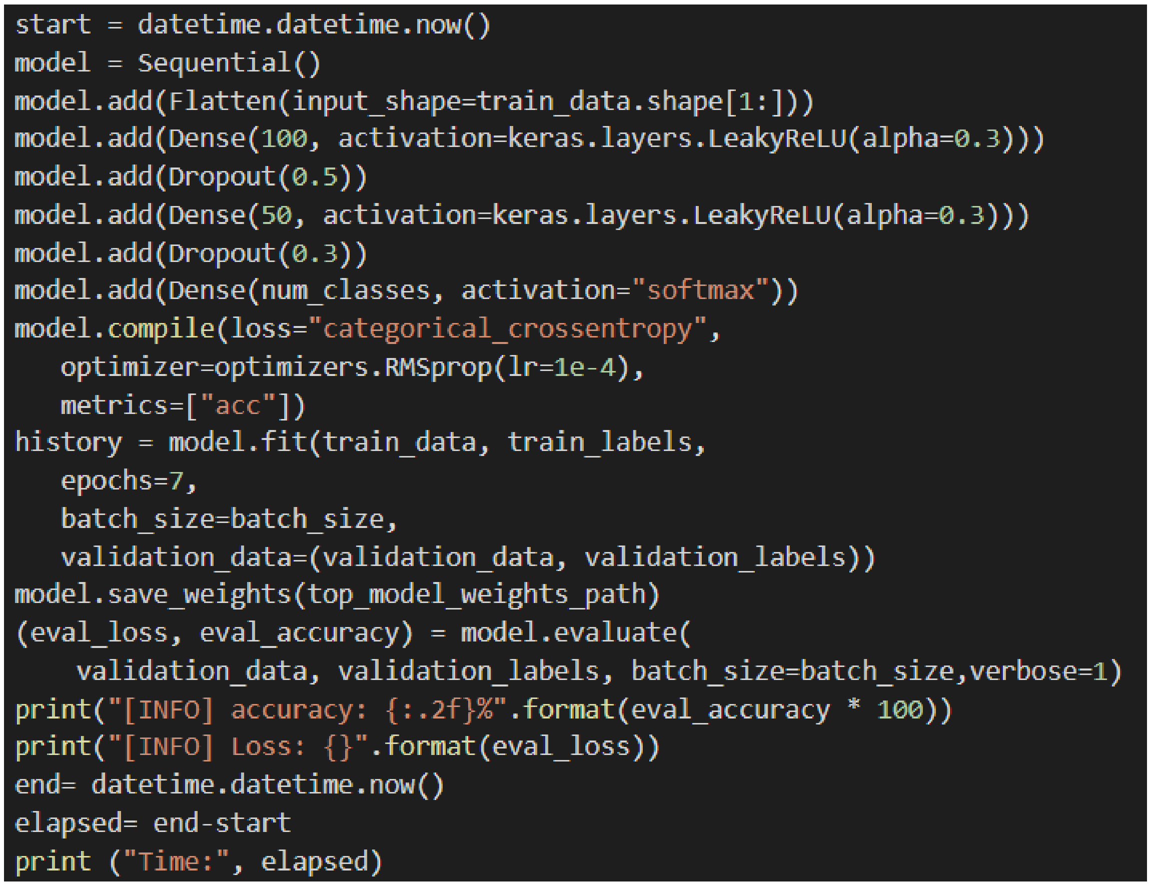


```
#validation data
generator_top1 = datagen.flow_from_directory(
   validation_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_validation_samples = len(generator_top1.filenames)
num_classes1 = len(generator_top1.class_indices)

# load the bottleneck features saved earlier
validation_data = np.load("bottleneck_features_train.npy")

# get the class labels for the validation data, in the original order
validation_labels = generator_top.classes

# convert the validation labels to categorical vectors
validation_labels = to_categorical(validation_labels, num_classes=num_classes1)
```



```
#test data
generator_top2 = datagen.flow_from_directory(
   test_data_dir,
   target_size=(img_width, img_height),
   batch_size=batch_size,
   class_mode='categorical',
   shuffle=False)

nb_test_samples = len(generator_top2.filenames)
num_classes2 = len(generator_top2.class_indices)

# load the bottleneck features saved earlier
test_data = np.load("bottleneck_features_train.npy")

# get the class labels for the test data, in the original order
test_labels = generator_top.classes

# convert the test labels to categorical vectors
test_labels = to_categorical(test_labels, num_classes=num_classes2)
```

Step Six: Creating our Convolutional Neural Network.

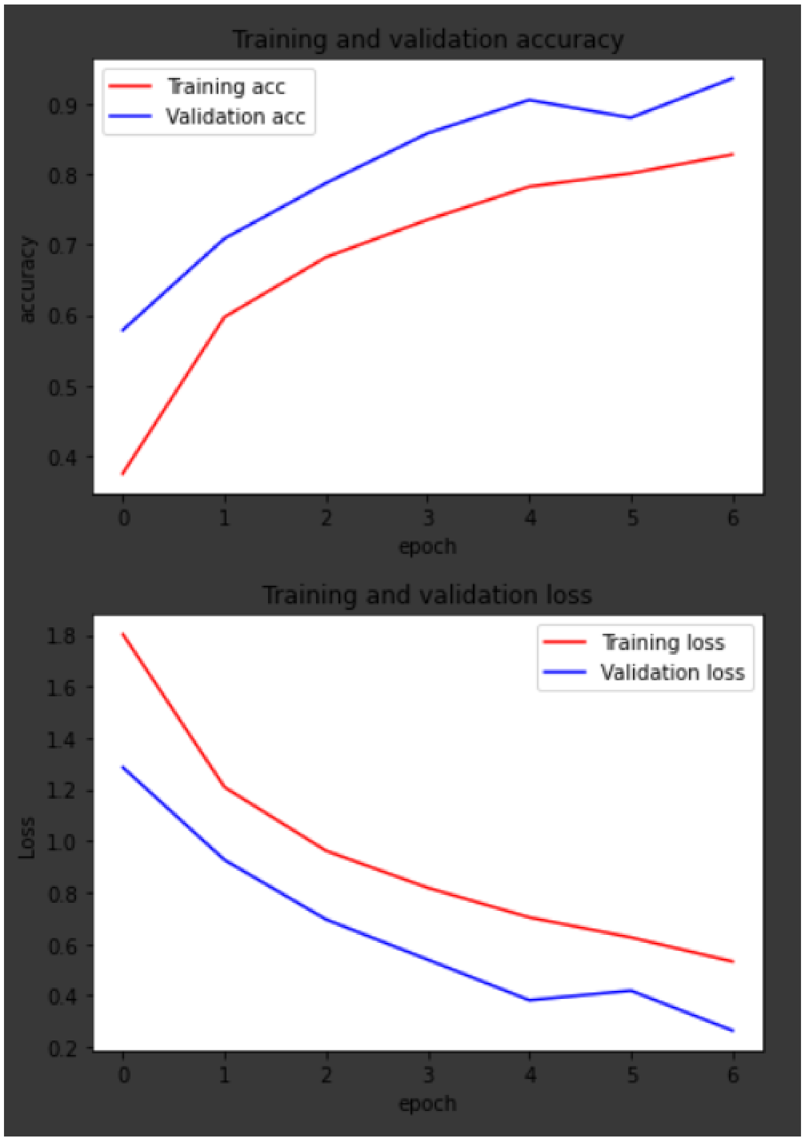


```
start = datetime.datetime.now()
model = Sequential()
model.add(Flatten(input_shape=train_data.shape[1:]))
model.add(Dense(100, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.5))
model.add(Dense(50, activation=keras.layers.LeakyReLU(alpha=0.3)))
model.add(Dropout(0.3))
model.add(Dense(num_classes, activation="softmax"))
model.compile(loss="categorical_crossentropy",
   optimizer=optimizers.RMSprop(lr=1e-4),
   metrics=["acc"])
history = model.fit(train_data, train_labels,
   epochs=7,
   batch_size=batch_size,
   validation_data=(validation_data, validation_labels))
model.save_weights(top_model_weights_path)
(eval_loss, eval_accuracy) = model.evaluate(
    validation_data, validation_labels, batch_size=batch_size,verbose=1)
print("[INFO] accuracy: {:.2f}%".format(eval_accuracy * 100))
print("[INFO] Loss: {}".format(eval_loss))
end= datetime.datetime.now()
elapsed= end-start
print ("Time:", elapsed)
```

The code above shows the CNN and how its being trained. We initialize the model as sequential() and add 3 hidden layers. This worked out the best was computationally friendly. We use softmax activation function.

An epoch is how many times the model trains on our whole data set. Batch can be explained as taking in small amounts, train and take some more. Each epoch must finish all batch before moving to the next epoch. Training with too little epoch can lead to underfitting the data and too many will lead to overfitting the data. You also want a loss that is as low as possible.

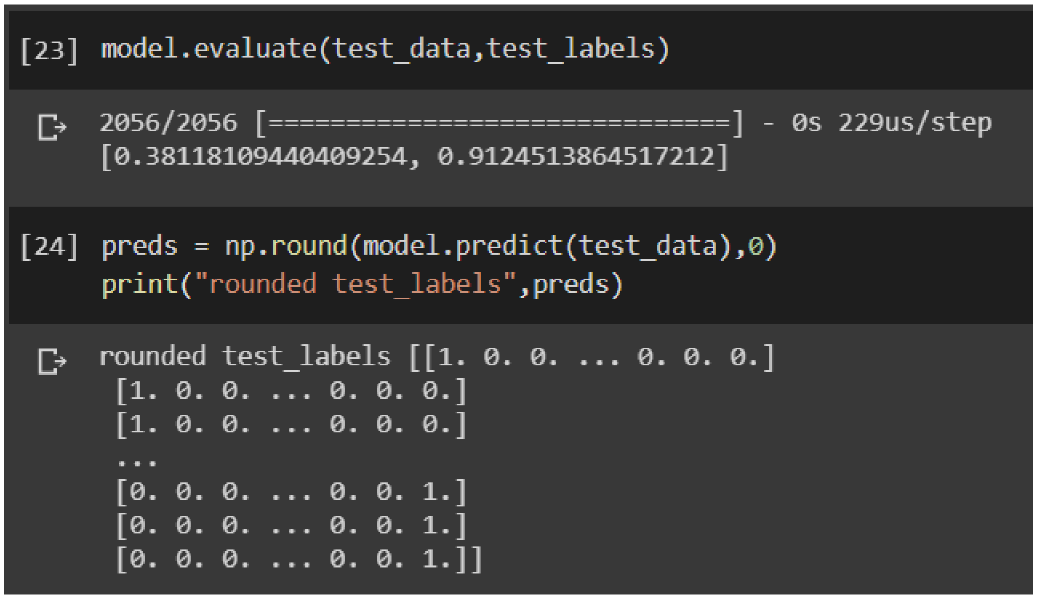
Step Seven: Visualization of Accuracy and Loss



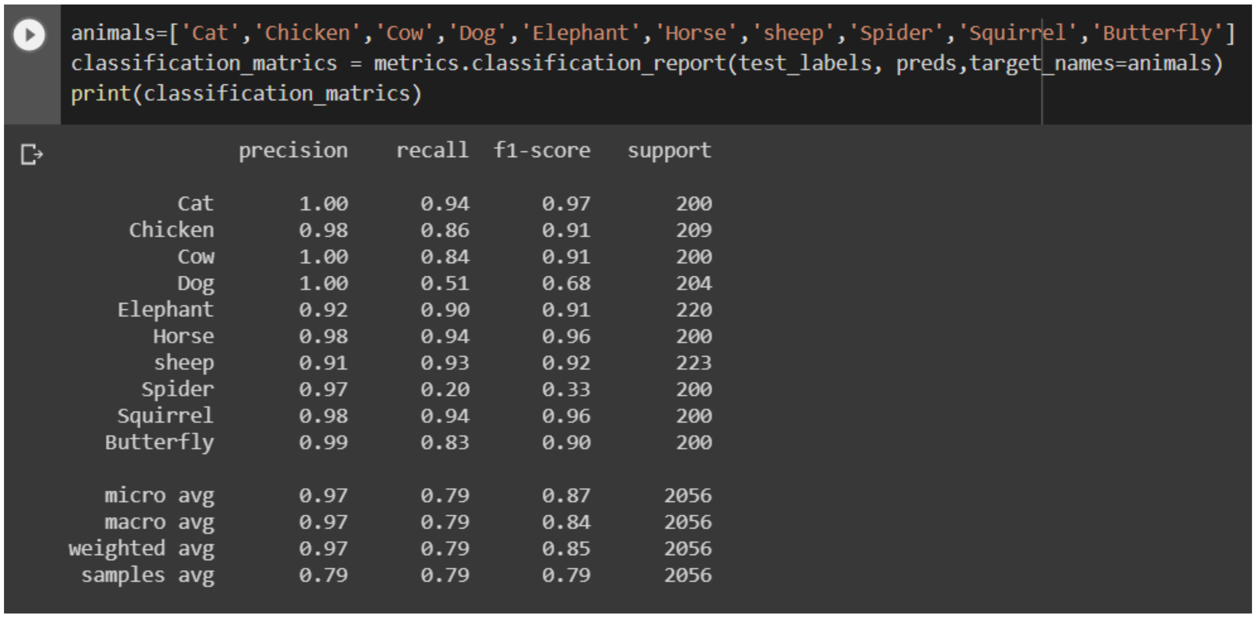
From the above figure, we can see that the accuracy for training set is lower than the validation set and in the case of loss the training loss is higher than validation loss , I my opinion this is good.

Step Eight: Prediction.

We can making prediction using model.predict(). We will also look at classification metrics.



```
[23] model.evaluate(test_data,test_labels)

2056/2056 [==============================] - 0s 229us/step
[0.38118109440409254, 0.9124513864517212]

[24] preds = np.round(model.predict(test_data),0)
     print("rounded test_labels",preds)

rounded test_labels [[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]
 [0. 0. 0. ... 0. 0. 1.]]
```



```
animals=['Cat','Chicken','Cow','Dog','Elephant','Horse','sheep','Spider','Squirrel','Butterfly']
classification_matrics = metrics.classification_report(test_labels, preds,target_names=animals)
print(classification_matrics)

              precision    recall  f1-score   support

         Cat       1.00      0.94      0.97       200
     Chicken       0.98      0.86      0.91       209
         Cow       1.00      0.84      0.91       200
         Dog       1.00      0.51      0.68       204
    Elephant       0.92      0.90      0.91       220
       Horse       0.98      0.94      0.96       200
       sheep       0.91      0.93      0.92       223
      Spider       0.97      0.20      0.33       200
    Squirrel       0.98      0.94      0.96       200
   Butterfly       0.99      0.83      0.90       200

   micro avg       0.97      0.79      0.87      2056
   macro avg       0.97      0.79      0.84      2056
weighted avg       0.97      0.79      0.85      2056
 samples avg       0.79      0.79      0.79      2056
```

From figure above, we can observe that the we try to pass our test data to make predictions with our trained model using evaluate() , as we can see the accuracy is very high. The Accuracy is at 91% which seem quite believable since the test data set contained a rhino picture which was not one of the classes we have declared in our classification.

From np.round() , we try to get our classification matrix and we can see classification matrix in the lower figure. We define our classes in animal list and print our classification matrix. We can see that precision and recall have very good values. F1 sroces are also pretty good.

To use classification metrics, we had to convert our testing data into a different numpy format, numpy array, to read. That is all the first line of code is doing. The second cell block takes in the converted code and run it through the built in classification metrics to give us a neat result. Please note that unless you manually label your classes here, you will get 0–5 as the classes instead of the animals. The important factors here are precision and f1-score. The higher the score the better your model is.