**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:**







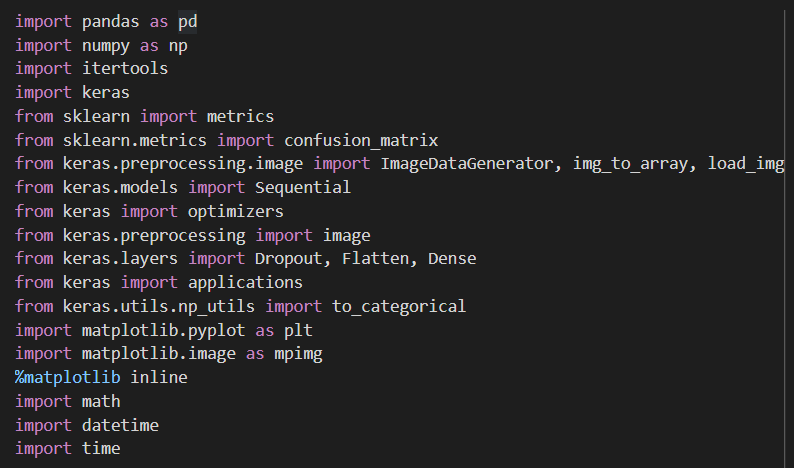


**Test data**



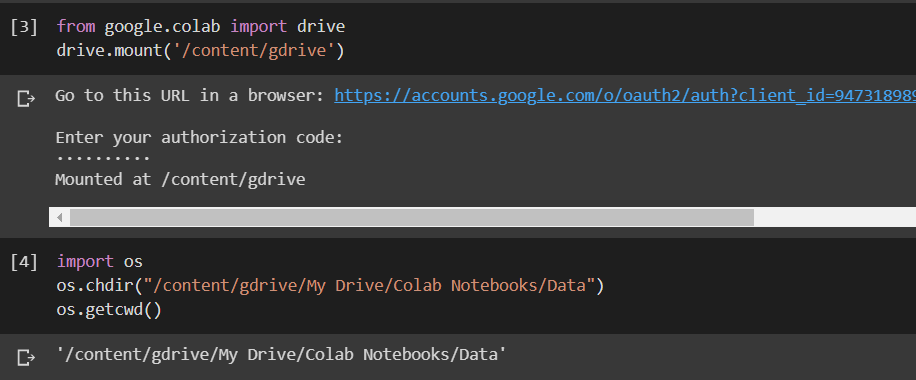
**Step Two: Import all the necessary libraries.**

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



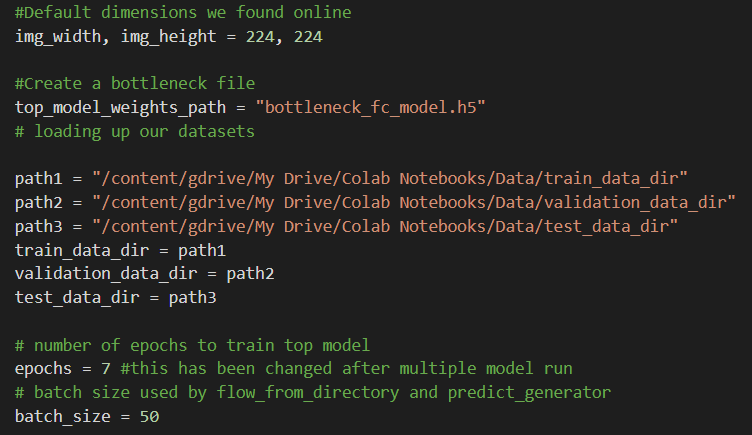
**Step Three: Image Dimensions and Locating images dataset on Google Drive.**

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



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.

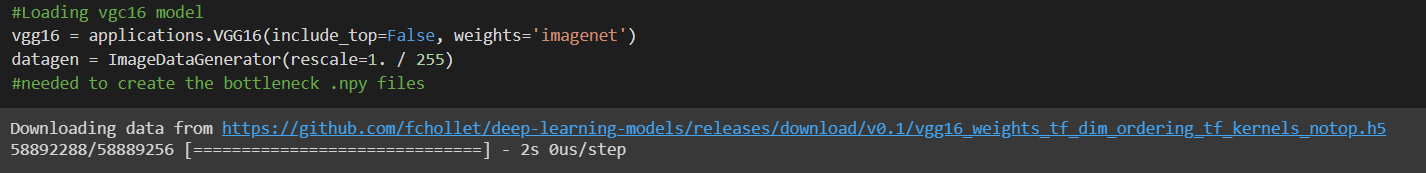
For image dimension, I am using 224\*224 dimension as 224\*224 works best in our case. Along with the image dimension we also create bottleneck file system.



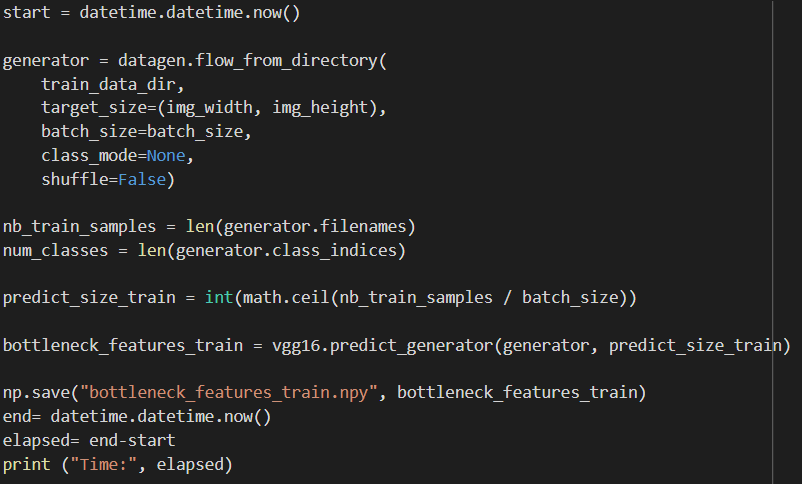
Here we also give the path of training data folder, validation data folder and testing data folder. We specify the epochs and batch size parameters as well to train out model.

**Step Four: Importing transfer learning model VGG16.**

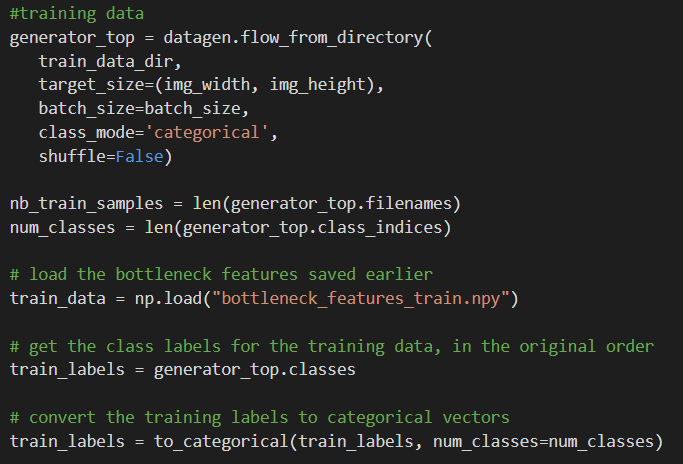
This is a very important step because transferring learning is very handy because it comes with already pre-made neural networks and other important components. The VGG16 model comes with 11 convolutional layers and it’s very easy to work with. There are many other transfer learning models which can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

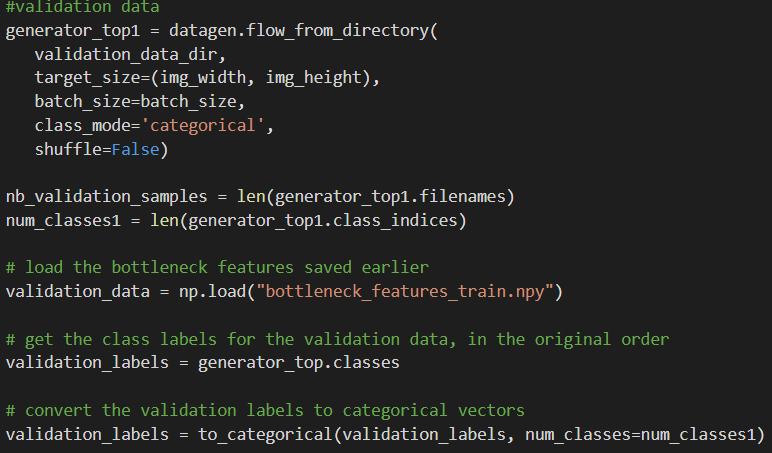


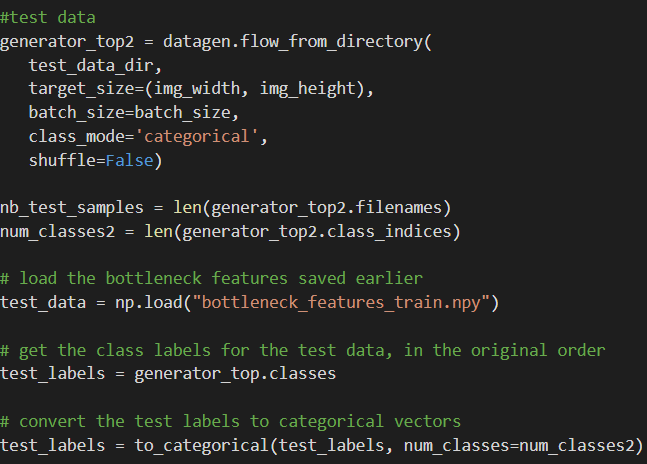
**Step Five: Weight and Feature using VGG16 Model**



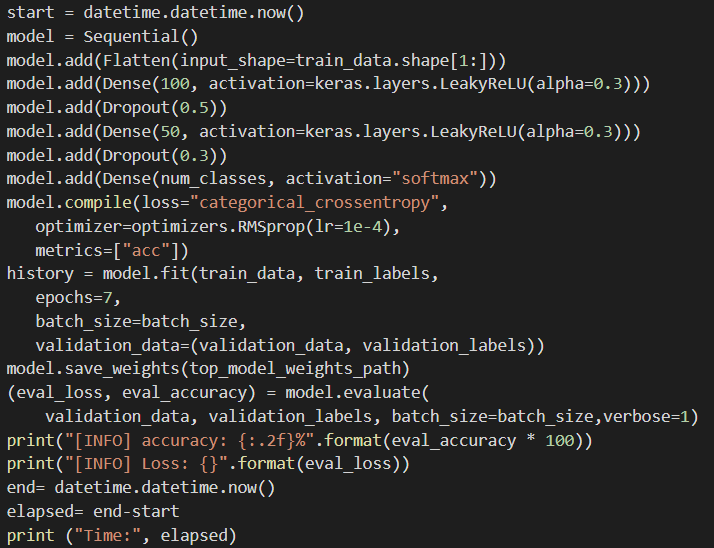
Here we are crating a simple image classifier and we perform this to all 3 datasets. Once this is done, we can this bottleneck file for our convolution neural network.







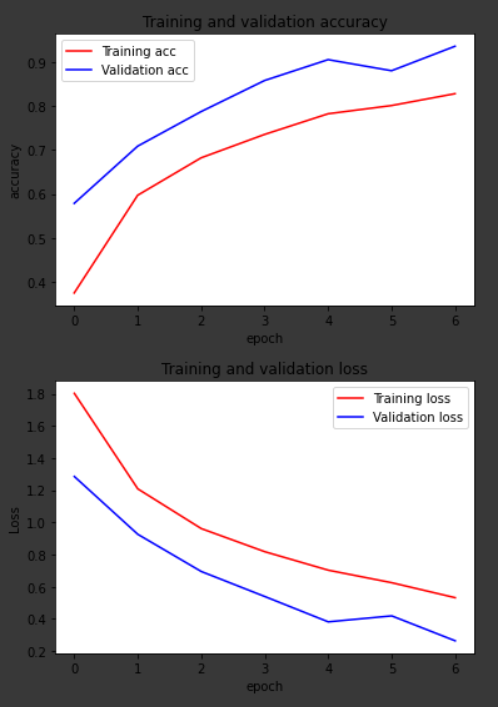
**Step Six: Creating our Convolutional Neural Network.**



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.

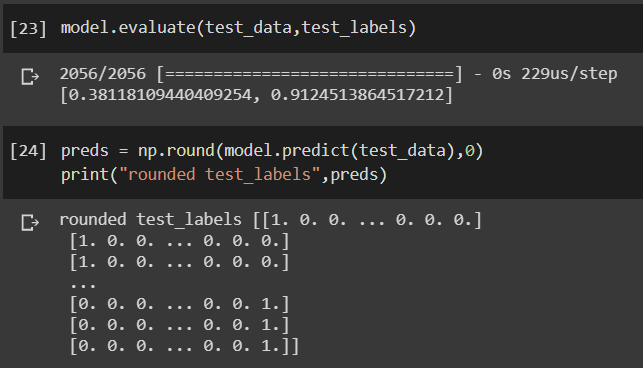
**Step Seven: Visualization of Accuracy and Loss**

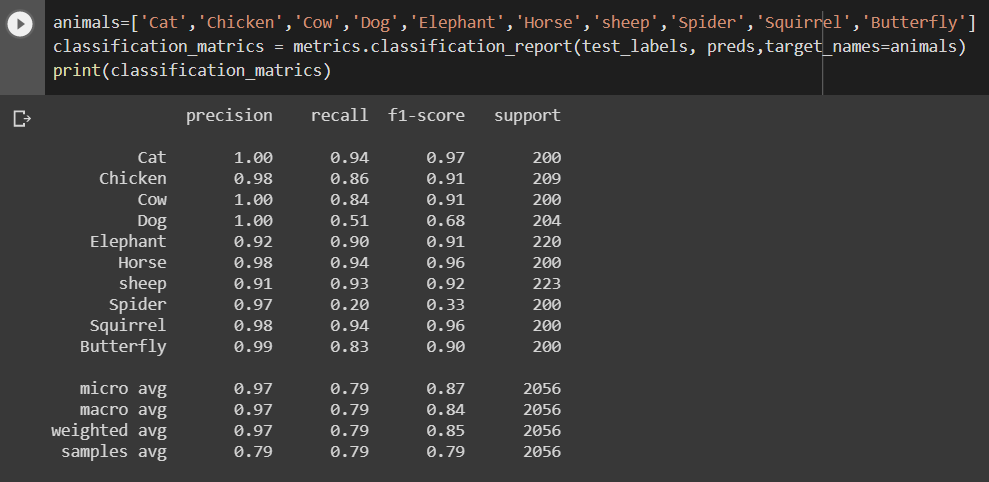


From the above figure, we can see that the accuracy for training set is lower than the validation set and in the case of loss the training loss is higher than validation loss , I my opinion this is good.

**Step Eight: Prediction.**

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





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.