FRUIT CLASSIFIER

Contribution Report

E/16/267

PARACKRAMA G.T.W.

GROUP NO: 06

OVERVIEW

From this mini-report, I am going to mention my work contribution to the CO542 research project continued throughout second half Of the semester.I am from Group 6 and our project was “Fruit Classifier”. Here, All the members are new to these concepts most of the work was done together by all 4 group members communicating via zoom meetings and Group chats. The research and testing parts were done individually to achieve the best fair output for our project. Here, I have included evidence of work with self-assessment on the project.

1.Introduction to the project

Fruit recognition using Deep Convolutional Neural Network (CNN) is one of the most promising applications in computer vision. In recent times, deep learning based classifications are making it possible to recognize fruits from images. However, fruit recognition is still a problem for the stacked fruits on weighing scale because of the complexityand similarity.

With the lively improvement of our human society, additional attention has been paid to the superiority of our life, particularly the food we eat. Over the last few years, computer visions have been widely used in fruit recognition methods. In the field of image recognition and classification, Deep Neural Network (DNN) is used to identify fruits from images. DNN performs better than other machine learning algorithms. Convolutional Neural Networks (CNNs) are classified as a deep learning algorithm. In deep learning, CNN are the most commonly used type of Artificial Neural Networks (ANNs). It is being used several visual recognition analyzing which includes video and image recognition, face recognition, handwritten digit recognition , and fruit recognition etc. The accuraciesin these fields including fruit recognition using CNN have reached human-level perfection.

CNN has a very similar architecture as ANN. There are several neurons in each layer in ANN. Hence, the weighted sum of all the neurons of a layer becomes the input of a neuron of the next layer adding a biased value. In CNN the layer has three dimensions. Here all the neurons are not fully connected instead they are connected to the local receptive field. A cost function is generated in order to train the network. It compares the network's output with the desired output.

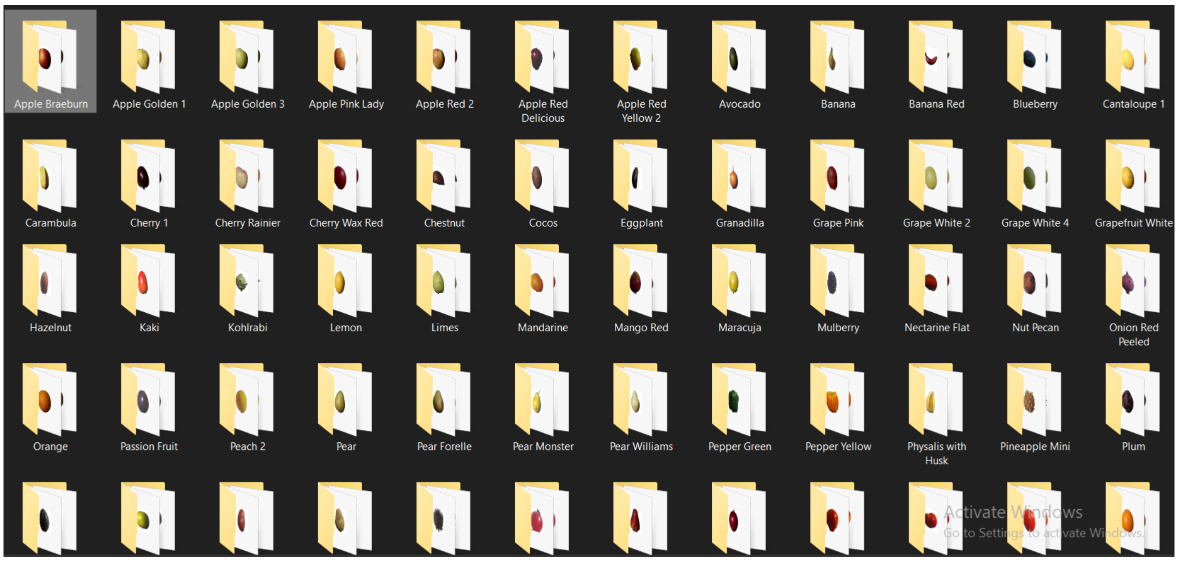
Accurate and efficient fruit recognition is of great importance in the field of robotic harvesting and yield mapping. An ideal fruit recognition system is accurate that can be trained on an easily available dataset, shows real-timepredictions and acclimates various types of fruits. Therefore, in our research, we implement a fruit recognition classifierusing CNN. The input image is taken as 100×100 pixels ofRGB image. For the networks best performance, we used various combinations of hidden layers for nine cases and observe the accuracies. The final experiment result shows the much-improved fruit recognition rate. The mathematical model of the network is executed in python with tensorflow.

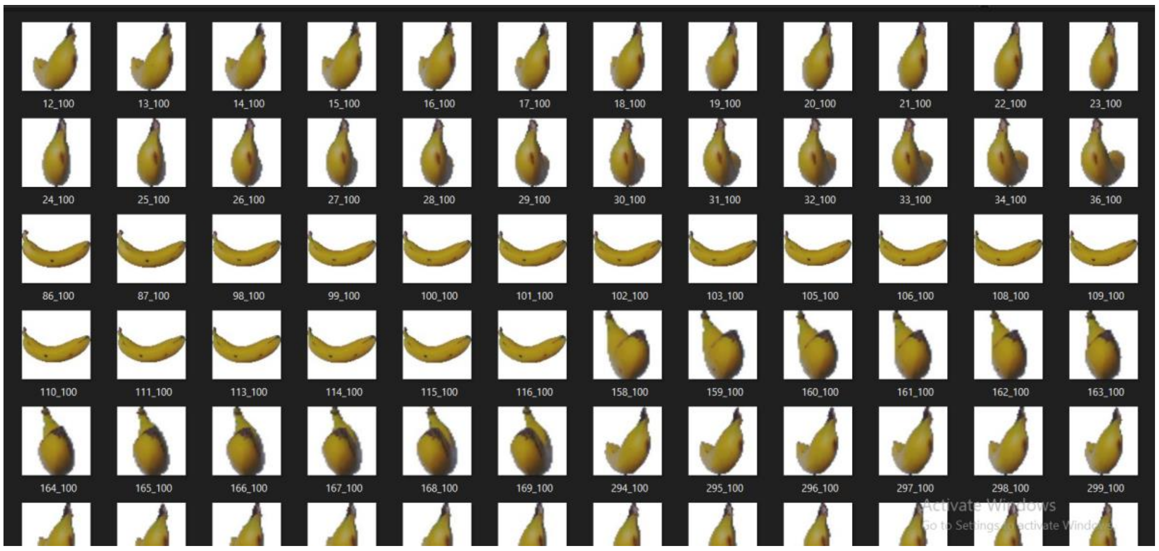
By researches done individually and by the team, we decided to use CNN technologies in this process since it shows high accuracy and efficiency in mage classification and pixel data processing. Also, it uses a system much like a multilayer perceptron that has been designed for reduced processing requirements.

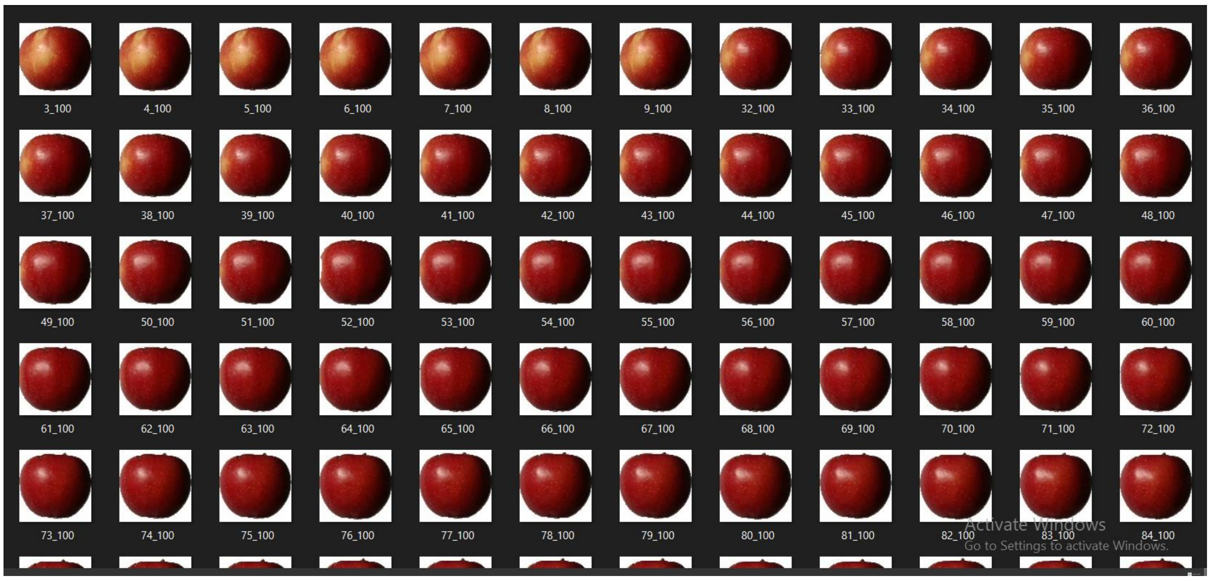
2.OVERALL ARCHITECTURE OF THE PROPOSED CNN MODEL

Our task is to implement a deep learning classifier of fruit images. The dataset contains segmented images of 60 different fruits. We'll have to implement a convolutional network.

The dataset we used was Fruits-small dataset (http://vegesm.web.elte.hu/). Even though it was not much large image dataset (it contained only 40800 images, 100 x 100 pixel, RGB), it contained image data of different types of fruits and also different species of the same fruit captured in different directions.







First We download the data and extract it:

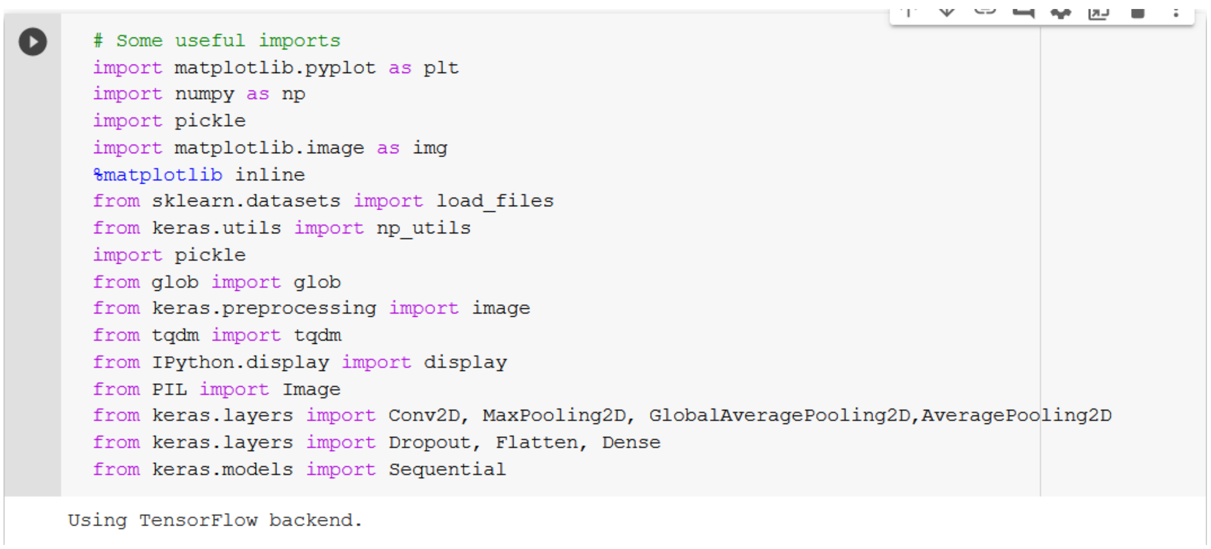
```
!wget http://vegesm.web.elte.hu/fruits_small.zip

!unzip fruits_small.zip > /dev/null
```

Dataset Characteristics:

Total number of images: 40800 images
Training set size: 30468 images
Test set size: 10332 images
Number of classes: 60 different fruits
Image size: 100 x 100 pixels, RGB.

The process steps were completed as follows.

1. Dataset preparations & testing the dataset set



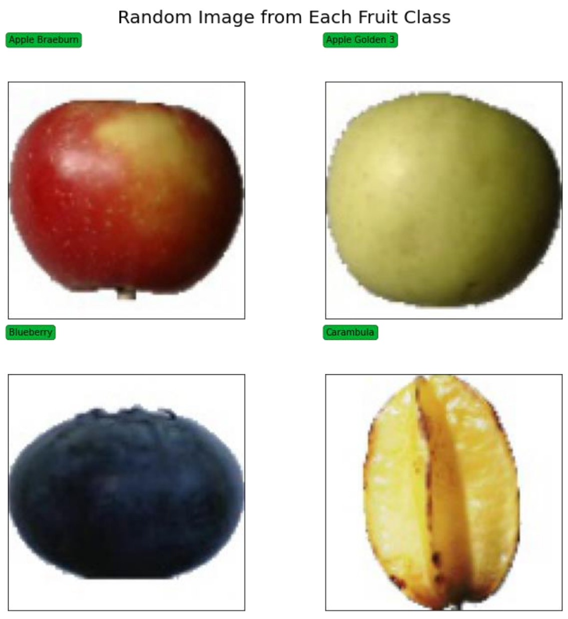
```
# Some useful imports
import matplotlib.pyplot as plt
import numpy as np
import pickle
import matplotlib.image as img
%matplotlib inline
from sklearn.datasets import load_files
from keras.utils import np_utils
import pickle
from glob import glob
from keras.preprocessing import image
from tqdm import tqdm
from IPython.display import display
from PIL import Image
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D,AveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential
```

```
Using TensorFlow backend.
```

After set up the dataset, I used a code just to check the dataset setup. The function randomly selects 4 images and prints them with the class names.

```
import os

dataset = '/content/fruits-small'

train_dir = '/content/fruits-small/train'

test_dir = '/content/fruits-small/test'

rows = 2

cols = 2

fig, ax = plt.subplots(rows, cols, frameon=False, figsize=(10, 10))

fig.suptitle('Random Image from Each Fruit Class', fontsize=20)

sorted_food_dirs = sorted(os.listdir(train_dir))

for i in range(rows):

    for j in range(cols):

        try:
            food_dir = sorted_food_dirs[5*i*cols + 2*j]
        except:
            break
        all_files = os.listdir(os.path.join(train_dir, food_dir))
        rand_img = np.random.choice(all_files)
        img = plt.imread(os.path.join(train_dir, food_dir, rand_img))
        ax[i][j].imshow(img)
        ec = (0, .6, .1)
        fc = (0, .7, .2)
        ax[i][j].text(0, -20, food_dir, size=10, rotation=0,
                ha="left", va="top",
                bbox=dict(boxstyle="round", ec=ec, fc=fc))
plt.setp(ax, xticks=[], yticks=[])
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

The results were as follows:

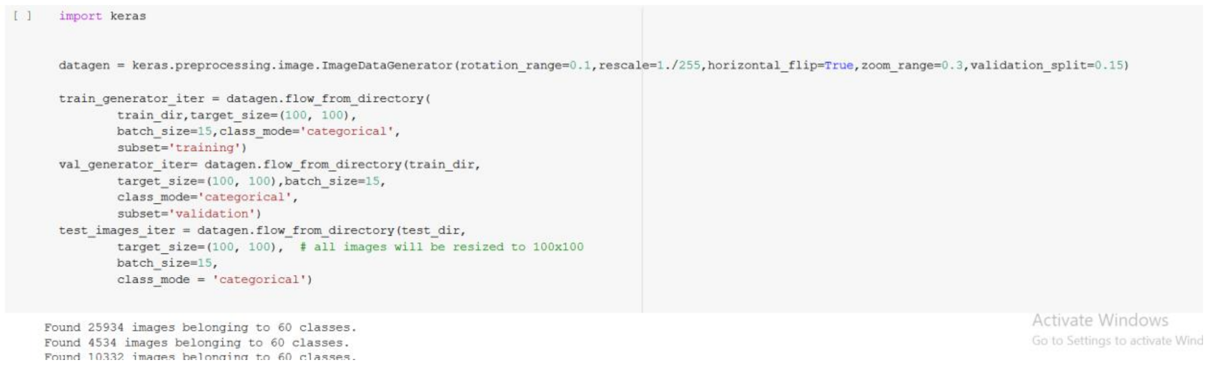


2.Splitting the dataset -

Here, our fruits small dataset had not defined a validation set. Therefore, we had to split it into validation and training sets ourselves.

3.**Preprocess the dataset**
Here ,We need to augment the data, since we do not have many images per classes. Create an augmentation mechanism, data automatically does the following transformations during training:.

- flip images horizontally
- rotates them
- performs zooming



```
[ ]  import keras


     datagen = keras.preprocessing.image.ImageDataGenerator(rotation_range=0.1,rescale=1./255,horizontal_flip=True,zoom_range=0.3,validation_split=0.15)

     train_generator_iter = datagen.flow_from_directory(
             train_dir,target_size=(100, 100),
             batch_size=15,class_mode='categorical',
             subset='training')
     val_generator_iter= datagen.flow_from_directory(train_dir,
             target_size=(100, 100),batch_size=15,
             class_mode='categorical',
             subset='validation')
     test_images_iter = datagen.flow_from_directory(test_dir,
             target_size=(100, 100),  # all images will be resized to 100x100
             batch_size=15,
             class_mode = 'categorical')
```

```
Found 25934 images belonging to 60 classes.
Found 4534 images belonging to 60 classes.
Found 10332 images belonging to 60 classes.
```

```
Found 25934 images belonging to 60 classes.
Found 4534 images belonging to 60 classes.
Found 10332 images belonging to 60 classes.
```

Here when using activation Function we tried three activation function

Relu
Tanh
Simsiod

There we notice Relu gives better result than other two with a high accuray

We also found that,

ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.ReLu does not have the vanishing gradient problem. Vanishing gradients lead to very small changes in the weights *proportional to the partial derivative* of the error function.

So for the activation functions we used RELU

4. Training the network

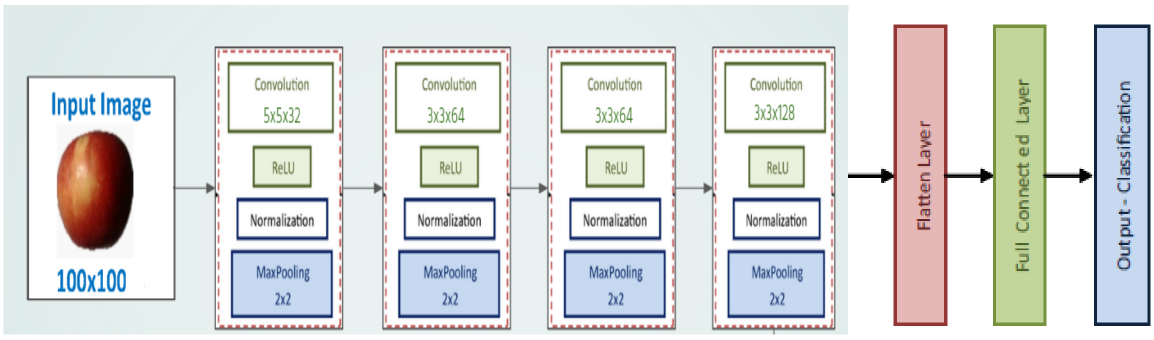
Next we had to Implement and train the architecture. It contained the following layers:

(Here one of my teammate go through deep in layered architecture and finally, we come with this final idea)

Implement and train the following architecture. It has the following layers:

- A convolutional layer with 5x5 kernel and 32 filters
- A 2x2 MaxPooling layer
- Two convolutional layers with 3x3 kernels and 64 filters each
- A MaxPooling layer
- Another 3x3 convolutional layer with 128 filters, followed by a MaxPooling layer
- A fully connected layer of 512 units
- A final softmax layer

All layers have ReLU activations. Train the network for 15 epochs.



This is our layered architecture

So Implementing and testing the above parts i have worked together with my team members. because without Understanding those parts the project cannot be continued further as I am new to this CNN models

So let's talk about my individual Contributin ;

When compiling and training the model I have put a much effective effect than other members in finding and testing optimizers and training the model.

Optimization Algorithms

Most of the neural network-based techniques including CNN utilize gradient descent to lower the error rate for the training process and for reforming the internal parameters. Gradient descent is a first-order optimization algorithm, and its derivatives provides direction and increasing or decreasing error function. Information guides the error function, altering it downward to the local minimum. The orthodox batch gradient descent technique computes a gradient of the whole training data, which makes its process computationally slow.

Therefore when Compiling the models normally Optimizers are used
Optimizers are **methods used to change the attributes of your neural network** such as weights and learning rate in order to reduce the losses. Optimizers help to get results faster
So there are Lot of Optimizers which is one of the two arguments required for compiling a Keras model

So i checked Our model with following three Optimizers

- Adagrad
- RMSprop
- Adam

Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

RMSprop Optimizer that implements the RMSprop algorithm.

The gist of RMSprop is to:

- Maintain a moving (discounted) average of the square of gradients
- Divide the gradient by the root of this average

This implementation of RMSprop uses plain momentum, not Nesterov momentum.The centered version additionally maintains a moving average of the gradients, and uses that average to estimate the variance.

These three optimizers are gradient desent based optimizers and when I traning model I noticed these three had given nearly same accuracy but I notice that Adam is little bit fast and lot of reasearchers stated that Adam is a good optimizer for image classification.

And also;

The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

- ***Adaptive Gradient Algorithm*** *(AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).*
- ***Root Mean Square Propagation*** *(RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).*

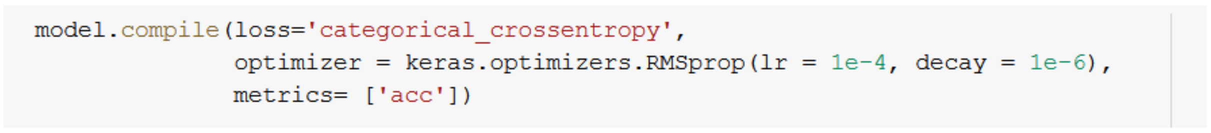
Adam realizes the benefits of both AdaGrad and RMSProp.

Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance).

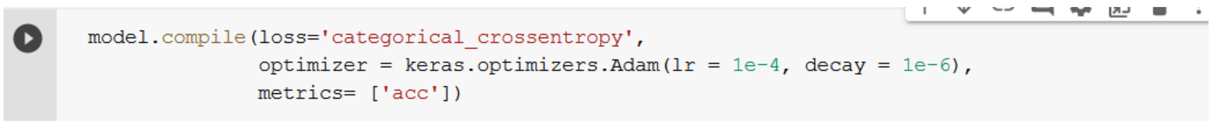
Because of these reasons I decided for obtain the final result we used adam as our optimizer

Adam is

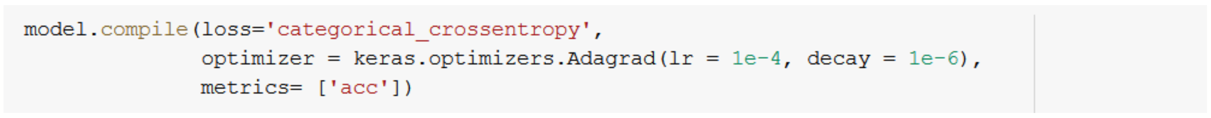
The Adaptive Momentum (Adam) technique estimates the adaptive learning rate for all parameters involved in the training of gradients. It is a computationally efficient and very simple technique that includes first-order gradients with a small memory requirement for stochastic optimization. The proposed technique is utilized in the case of machine learning issues with high-dimensional parameter spaces and huge data sets that calculate learning rates individually for various parameters from approximations that include first- and 2nd-order moment



```
model.compile(loss='categorical_crossentropy',
              optimizer = keras.optimizers.RMSprop(lr = 1e-4, decay = 1e-6),
              metrics= ['acc'])
```



```
model.compile(loss='categorical_crossentropy',
              optimizer = keras.optimizers.Adam(lr = 1e-4, decay = 1e-6),
              metrics= ['acc'])
```



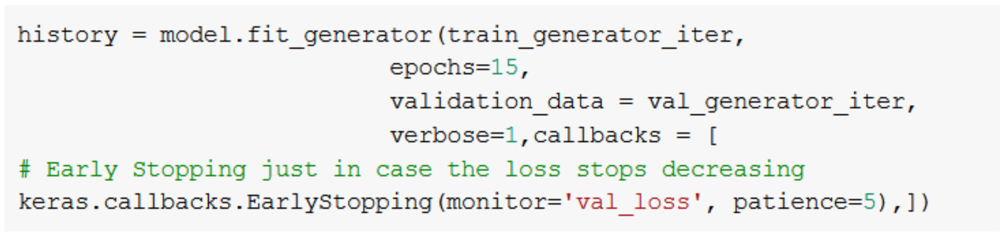
```
model.compile(loss='categorical_crossentropy',
              optimizer = keras.optimizers.Adagrad(lr = 1e-4, decay = 1e-6),
              metrics= ['acc'])
```

Accuracy of three optimizers;

Optimizer	Accuracy
Adam	98.09%
RMSProp	98.01%
AdaGrad	97.99%

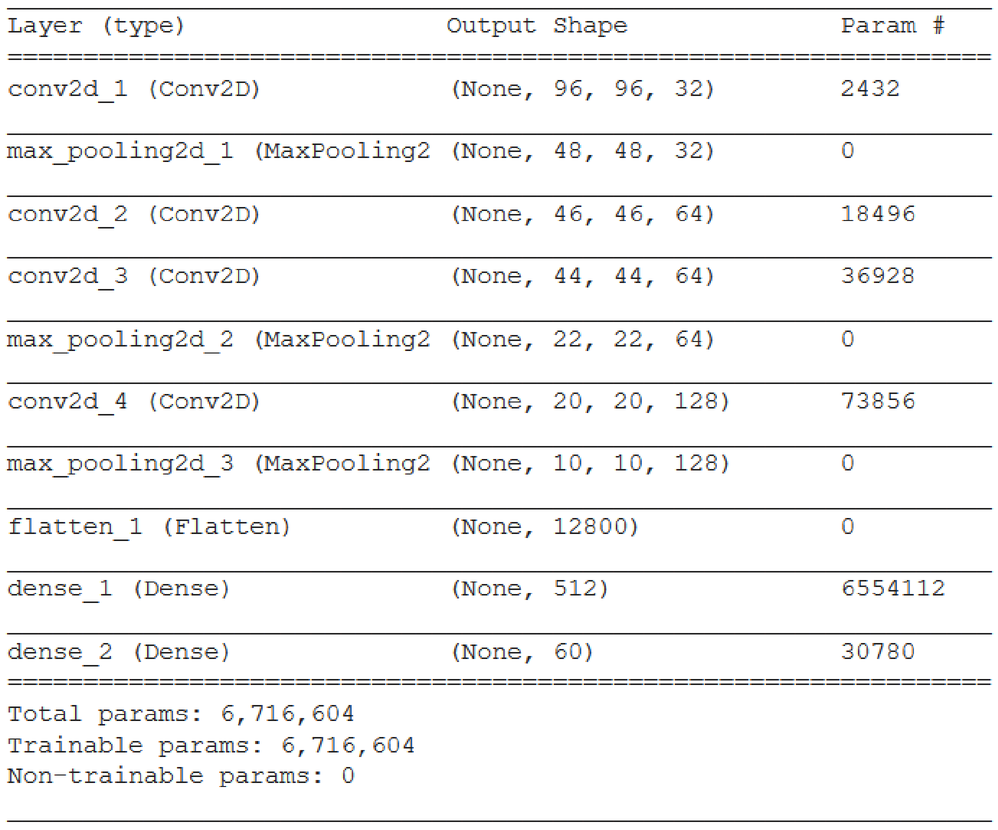
When training we used,

keras.fit_generator() in Python **which is** separate deep learning librarie which can be used to train our machine learning and deep learning models.



```
history = model.fit_generator(train_generator_iter,
                              epochs=15,
                              validation_data = val_generator_iter,
                              verbose=1,callbacks = [
# Early Stopping just in case the loss stops decreasing
keras.callbacks.EarlyStopping(monitor='val_loss', patience=5),])
```

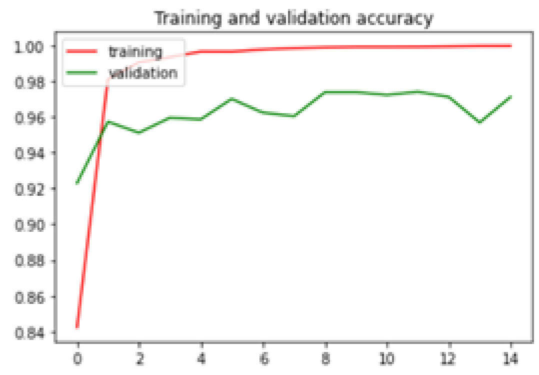
3. Results

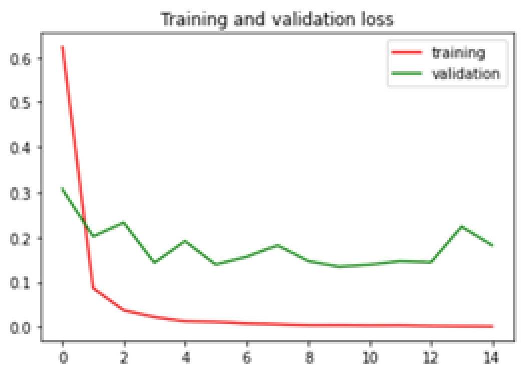


```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 96, 96, 32)        2432
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 64)        18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 44, 44, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 22, 22, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 20, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 10, 10, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               6554112
_________________________________________________________________
dense_2 (Dense)              (None, 60)                30780
=================================================================
Total params: 6,716,604
Trainable params: 6,716,604
Non-trainable params: 0
_________________________________________________________________
```

```
1729/1729 [==============================] - 130s 75ms/step - loss: 0.6235 - acc: 0.8423 - val_loss: 0.3070 - val_acc: 0.9228
Epoch 2/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0865 - acc: 0.9810 - val_loss: 0.2022 - val_acc: 0.9572
Epoch 3/15
1729/1729 [==============================] - 118s 68ms/step - loss: 0.0371 - acc: 0.9905 - val_loss: 0.2330 - val_acc: 0.9510
Epoch 4/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0219 - acc: 0.9934 - val_loss: 0.1432 - val_acc: 0.9594
Epoch 5/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0124 - acc: 0.9965 - val_loss: 0.1918 - val_acc: 0.9585
Epoch 6/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0111 - acc: 0.9965 - val_loss: 0.1394 - val_acc: 0.9700
Epoch 7/15
1729/1729 [==============================] - 118s 68ms/step - loss: 0.0076 - acc: 0.9977 - val_loss: 0.1565 - val_acc: 0.9623
Epoch 8/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0057 - acc: 0.9984 - val_loss: 0.1822 - val_acc: 0.9603
Epoch 9/15
1729/1729 [==============================] - 115s 66ms/step - loss: 0.0034 - acc: 0.9988 - val_loss: 0.1470 - val_acc: 0.9738
Epoch 10/15
1729/1729 [==============================] - 116s 67ms/step - loss: 0.0034 - acc: 0.9991 - val_loss: 0.1344 - val_acc: 0.9738
Epoch 11/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0028 - acc: 0.9991 - val_loss: 0.1390 - val_acc: 0.9722
Epoch 12/15
1729/1729 [==============================] - 116s 67ms/step - loss: 0.0030 - acc: 0.9992 - val_loss: 0.1469 - val_acc: 0.9740
Epoch 13/15
1729/1729 [==============================] - 117s 68ms/step - loss: 0.0020 - acc: 0.9994 - val_loss: 0.1443 - val_acc: 0.9711
Epoch 14/15
1729/1729 [==============================] - 118s 68ms/step - loss: 0.0016 - acc: 0.9996 - val_loss: 0.2240 - val_acc: 0.9568
Epoch 15/15
1729/1729 [==============================] - 118s 68ms/step - loss: 0.0011 - acc: 0.9997 - val_loss: 0.1823 - val_acc: 0.9711
```

So,The model has finished training, the accuracy and loss over time, both for training and validation data were plotted as follows.



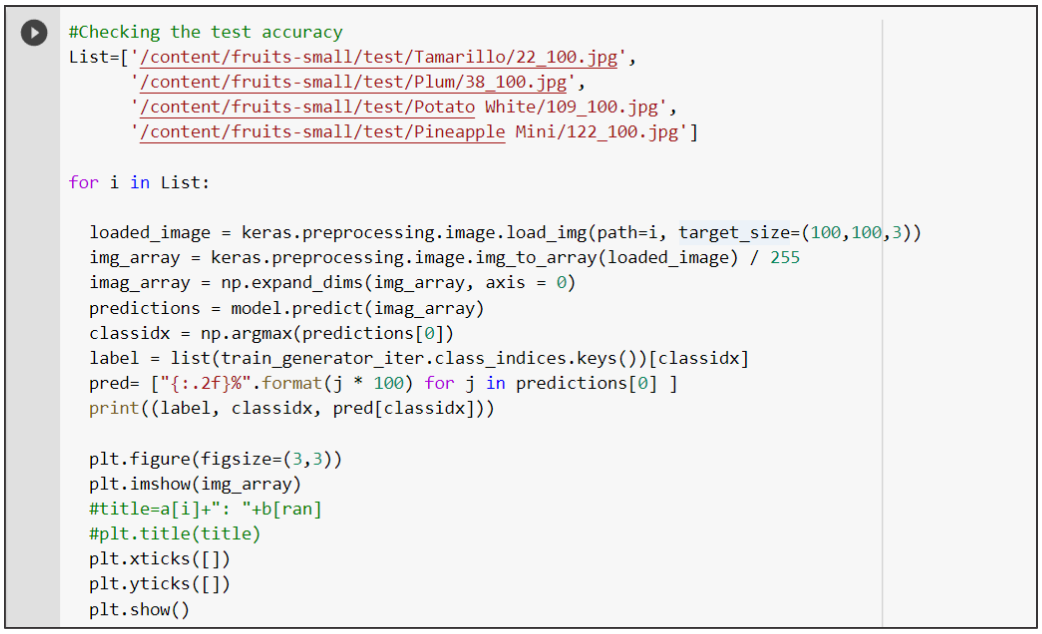


The performance of the model was able to get as follows.

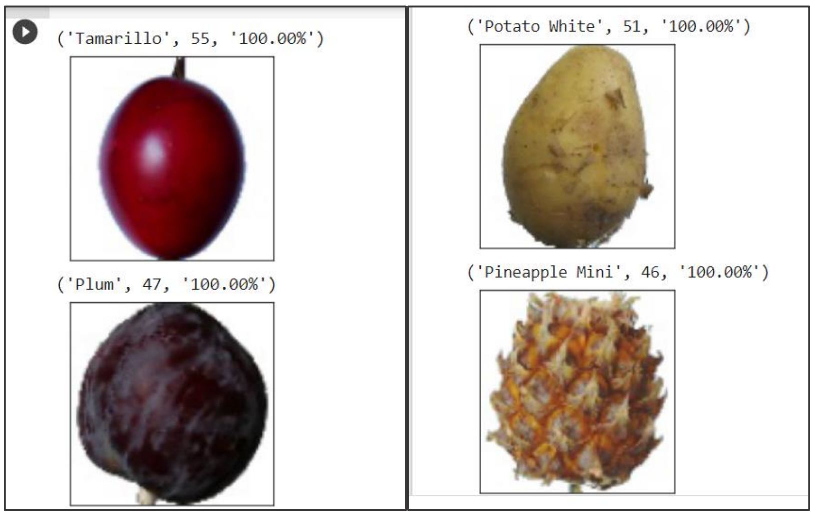
```
loss,acc = model.evaluate_generator(generator=test_images_iter, steps=500, use_multiprocessing=False)
print('test loss: {}'.format(loss))
print('test accuracy: {:.2%}'.format(acc))
```

```
test loss: 0.09727661704928148
test accuracy: 98.09%
```

So these are some test results we observed

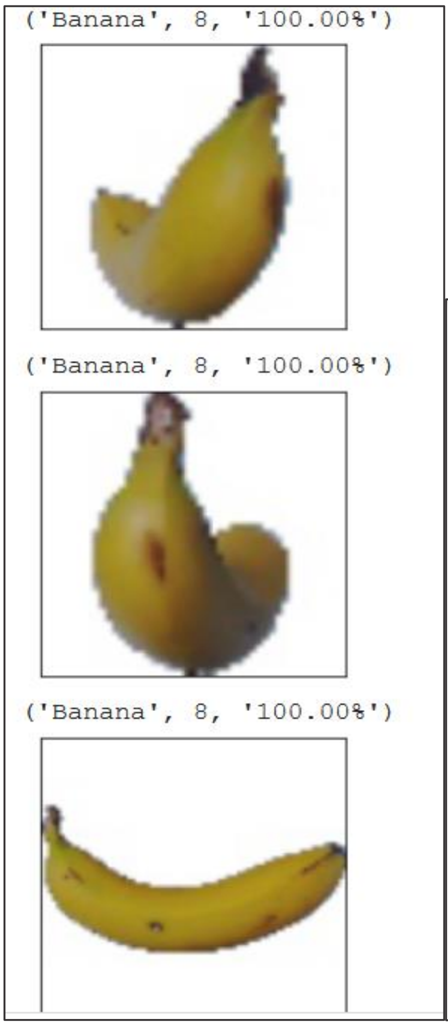


```
#Checking the test accuracy
List=['/content/fruits-small/test/Tamarillo/22_100.jpg',
      '/content/fruits-small/test/Plum/38_100.jpg',
      '/content/fruits-small/test/Potato White/109_100.jpg',
      '/content/fruits-small/test/Pineapple Mini/122_100.jpg']

for i in List:

  loaded_image = keras.preprocessing.image.load_img(path=i, target_size=(100,100,3))
  img_array = keras.preprocessing.image.img_to_array(loaded_image) / 255
  imag_array = np.expand_dims(img_array, axis = 0)
  predictions = model.predict(imag_array)
  classidx = np.argmax(predictions[0])
  label = list(train_generator_iter.class_indices.keys())[classidx]
  pred= ["{:.2f}%".format(j * 100) for j in predictions[0] ]
  print((label, classidx, pred[classidx]))

  plt.figure(figsize=(3,3))
  plt.imshow(img_array)
  #title=a[i]+": "+b[ran]
  #plt.title(title)
  plt.xticks([])
  plt.yticks([])
  plt.show()
```

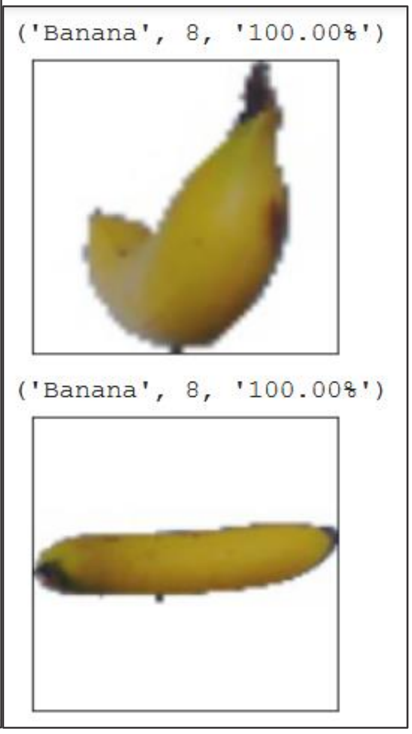


```
('Tamarillo', 55, '100.00%')
('Plum', 47, '100.00%')
('Potato White', 51, '100.00%')
('Pineapple Mini', 46, '100.00%')
```

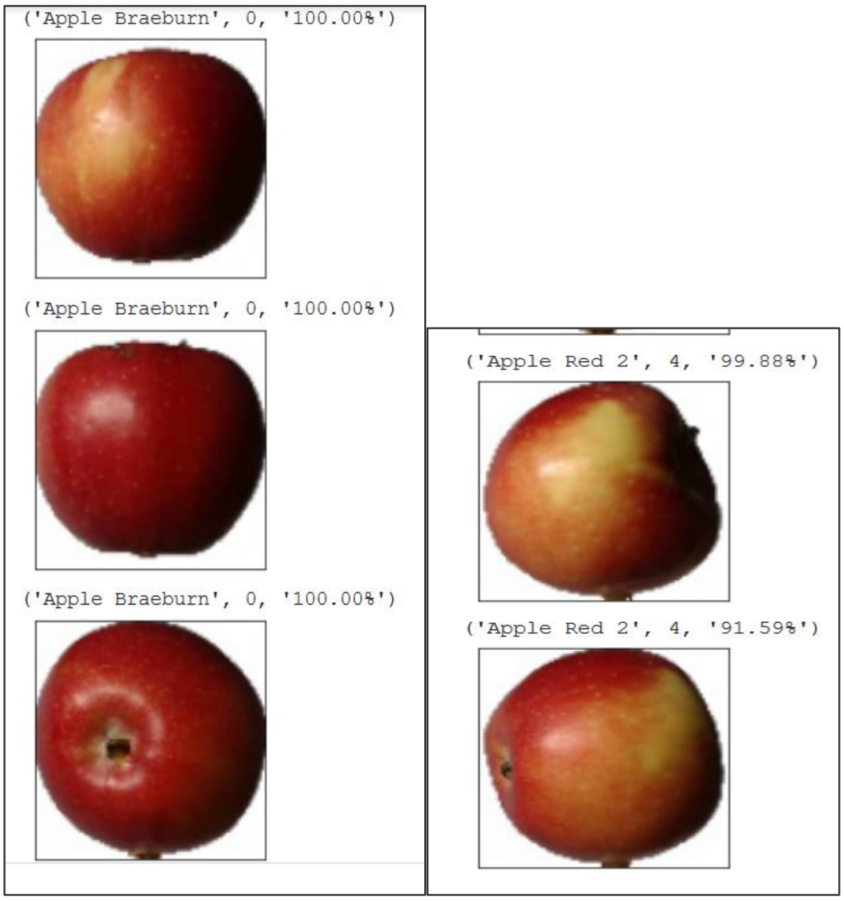
And also below are some test results in reconginzing bananas



```
('Banana', 8, '100.00%')
('Banana', 8, '100.00%')
('Banana', 8, '100.00%')
```

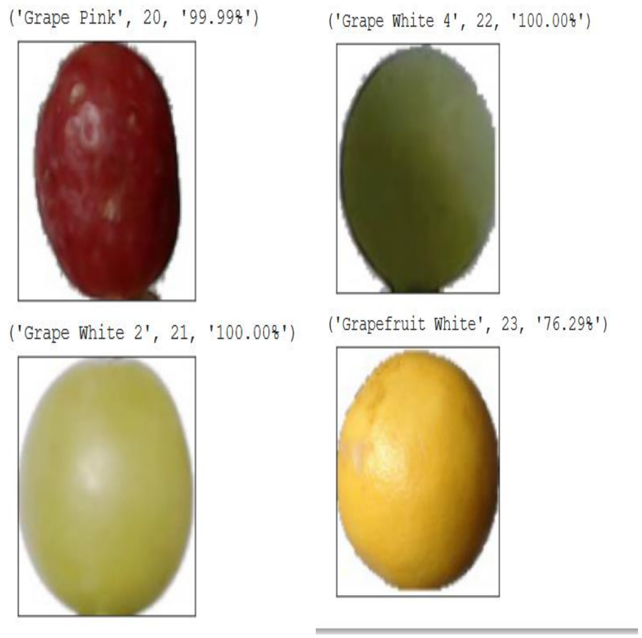


```
('Banana', 8, '100.00%')
('Banana', 8, '100.00%')
```



```
('Apple Braeburn', 0, '100.00%')
('Apple Braeburn', 0, '100.00%')
('Apple Braeburn', 0, '100.00%')
('Apple Red 2', 4, '99.88%')
('Apple Red 2', 4, '91.59%')
```

Here it Recognized raw and ripe apples which is some time humans also doubt about is this is apple or not with good accuracy



```
('Grape Pink', 20, '99.99%')
('Grape White 4', 22, '100.00%')
('Grape White 2', 21, '100.00%')
('Grapefruit White', 23, '76.29%')
```

Some results recognizing Graphs with different varies

4.Conclusion

As explained, we came up with a successful model and overall, I played a role as a codeveloper of this model and contributed and shared the knowledge with my team members in each and every part of our project.

5.References

https://evidencen.com/15-fruits-image-classification-with-computer-vision-and-tensorflow/

https://arxiv.org/abs/1904.00783

https://keras.io/api/optimizers/

https://www.mdpi.com/2313-433X/6/9/92/pdf

https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0