FACIAL EXPRESSION RECOGNITION

Submitted in partial fulfilment of the requirements for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

MINI PROJECT REPORT

Submitted By

SOUMABHA CHAKRABORTY

(Registration Number: 20352058)

Under the Guidance of

Dr. V. UMA

Assistant Professor.

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF ENGINEERING AND TECHNOLOGY

PONDICHERRY UNIVERSITY

DECEMBER 2021

SCHOOL OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Master of Computer Applications

BONAFIDE CERTIFICATE

This is to certify that the Project Work titled **"FACIAL EXPRESSION RECOGNITION"** is a bonafide work of Mr**. SOUMABHA CHAKRABORTY**, Registration No. **20352058** carried out in partial fulfilment for the award of degree of **MASTER OF COMPUTER APPLICATIONS** of **Pondicherry University** during the academic year 2020-21**.** This mini project work is original and not submitted earlier for the award of any degree/diploma in the University to the best of our knowledge.

Signature of the Guide / Supervisor

Dr. V.UMA

Assistant Professor,

Department of Computer Science,

School of Engineering & Technology,

Pondicherry University.

ACKNOWLEDGEMENT

I would like to express my special thanks of gratitude to my guide respected Dr. V. Uma, for her continuous guidance, motivation and support. I am grateful to respected Head of the Department (Computer Science), Prof. S. Sivasathya who gave me the golden opportunity to do this wonderful project on the topic FACIAL EXPRESSION RECOGNITION which also helped me in doing a lot of Research and I came to know about so many new things I am really thankful to them. Secondly I would also like to thank my Professors and staff-members who helped me a lot in finalizing this project within the limited time frame.

SOUMABHA CHAKRABORTY

ABSTRACT

In the fast pacing world of technology, emotions and expressions play a huge role for employees and the AI is all about understanding the human way of thinking. Companies spend billions of dollars to know how the emotions, sentiments and expressions help in the optimum growth of company and high productivity of enterprises. A system that can read the expression of a person based on biometric markers can actually help the person sitting on the opposite side of the screen to know much about the person whom he is talking to. This project is developed to reduce the tedious process of recognising facial expression over electronic media like video calls and audio visual meetings. This is a Deep learning based project which is implemented using python and related libraries that perform deep neural networks. It first detects the face, and scribes a green rectangle around the face based on the detection as an indication of object (face) and according to the expression of the person inside the green box it predicts and gives the expression based on the 7 expressions categorised in it according to the dataset.

Facial Expression Recognizer provides a detail study of all the 7 universal expressions (sadness, happiness, disgust, anger, surprise, fear and neutral). And return its prediction in real time streaming data. The main goal of this project is to determine the expression and show it along with the detected face in real time.

TABLE OF CONTENT

CHAPTER 1: INTRODUCTION

1.1 ABOUT THE PROJECT

Communication plays a huge part in our daily life. And facial expressions are one of the most important aspects of communication. This is because facial expressions not only help in expressing thoughts and ideas but also emotions in a substantial way.

In our daily lives, we can easily identify the personal emotion of the person standing nearby just by observing their facial expression. There is an amazing fact about facial expressions that are unknown to many. Although they are considered to be products of social learning as culture, Darwin mentions that there are expressions that are products of human evolution.

Facial expression recognition is a technology that will complement our communication to a great extent. It is used widely in many industries, it uses biometric markers to detect emotions in human faces. (Biometric markers explained at the end of the document)

More precisely, this technology is a sentiment analysis tool and is able to automatically detect six basic universal expressions happiness, sadness, anger, surprise, fear and disgust. Neutral has been also added later as a categorical class.

CHAPTER 2: PROBLEM DEFINITION AND FEASIBILITY STUDY

2.1 PROBLEM DEFINITION

Understanding facial expressions is very important because facial expressions can be nonverbal communication clues that play an important role in interpersonal relations. These cues complement speech by helping the listener to interpret the intended meaning of the spoken words.

(Where we can use)

. Market Research.

. Gaming industry (emotion experienced in gameplay).

. Feedback from people without voice.

. Meetings in office

. Behavioural systems testing

. Corporate interviews

The purpose of this project is to build an effective expression analysis that would reduce the above problems.

BIOMETRIC MARKERS: - Biometric markers are like eyes and facial patterns which are different for each person. Each face has some signature which if summed together brings out a technique of uniquely recognizing a pattern. (The system plots some coordinate points in our faces which makes a geometrical figure of polygons. The area, edges are unique for every individual. And this recognized the pattern it makes.)

2.2 OBJECTIVE OF THE PROJECT

The objective of the project is to accumulate different images with facial expressions from different datasets and make changes into it with image augmentation techniques like scaling, zooming, rotation. Splitting the data into training and test set and changing it from image to vectors while doing it. Then using different deep learning methods find the

suitable algorithm and train the model in two phases for the amount of data that is being used here. Perform comparison between different accuracy and loss with different functions and then utilise it in real time streaming data from the webcam or embedded camera in laptop. We are going to deal with image datasets, performing data processing creating and training a convolutional neural network using tensorflow 2.X. The idea is to show the correct expression of the person.

2.3 EXISTING SYSTEM

The existing system only detects the faces of the individuals and using different models and algorithms classifies it between human or different objects. The existing system works with camera drivers that trigger the webcam and detect the faces. This proposed system will not only detect faces but will also show the expression of the person in it with the help of Deep learning methods and machine learning algorithms.

2.4 FEASIBILITY STUDY

The feasibility is the test conducted to its workability, meeting user requirements, effective use of system and the cost. The objective of the feasibility is not the solve problems but acquire an idea of its scope also. The proposed system considered as a feasible if only it is useful and it is determined with preliminary stage. Here the project is economically and legally feasible but the technically there are many constraints however with the available resources we can still make a good model out of it.

2.5 KNOWLEDGE OF DATASET

- Dataset available in (https://www.kaggle.com/mahmoudima/mma-facial-expression)
- Here is the description for the data.
- The images provided in this dataset are MMA facial expression MMAFEDB is collected from different facial expression datasets.
- All images are cropped to only face region and resized to 48x48 pixels.
- The dataset contains 22,485,444 images including train and test data in different classes.
- All the images are coloured images.
- Images are augmented for better prediction in the model.
- **Some of the images are taken from other sources.

CHAPTER 3: SYSTEM REQUIREMENTS AND SPECIFICATIONS

The domain of this project is Deep Learning including CNN, ANN and Computer vision. A brief research on this domain has already been done with a scope for future implementation of this along with sentiment analysis.

3.1 SYSTEM REQUIREMENTS

- An operating system.
- Google account for accessing google Colab.
- A browser that supports google Colab Web servers.
- Or a high end computer that can process Deep learning methods with ease.

3.2 SOFTWARE REQUIREMENTS

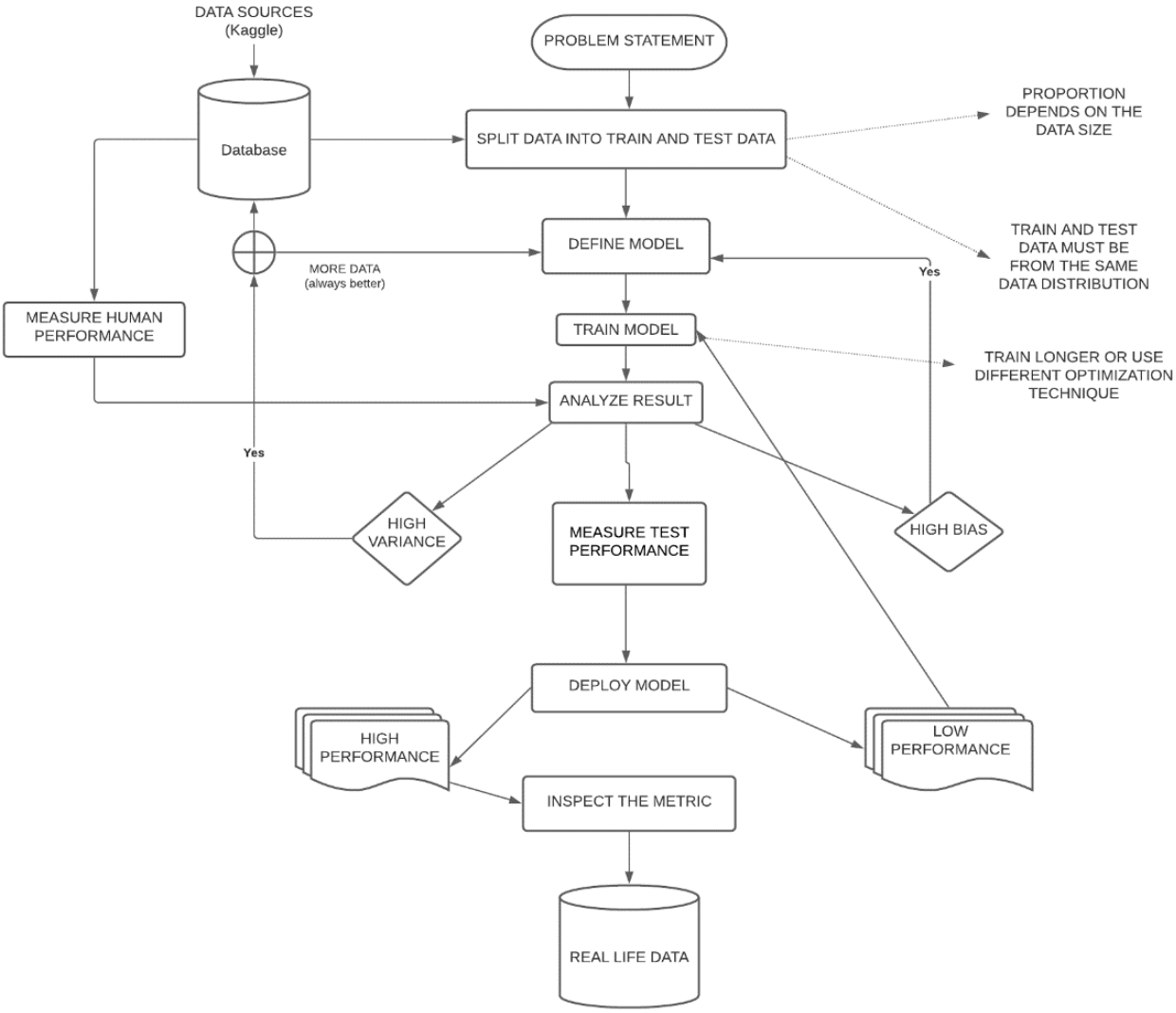
- Python interpreter and IDE.
- Jupyter Notebook.
- Necessary python library functions.
- Required Dataset.

3.3 HARDWARE REQUIREMENTS

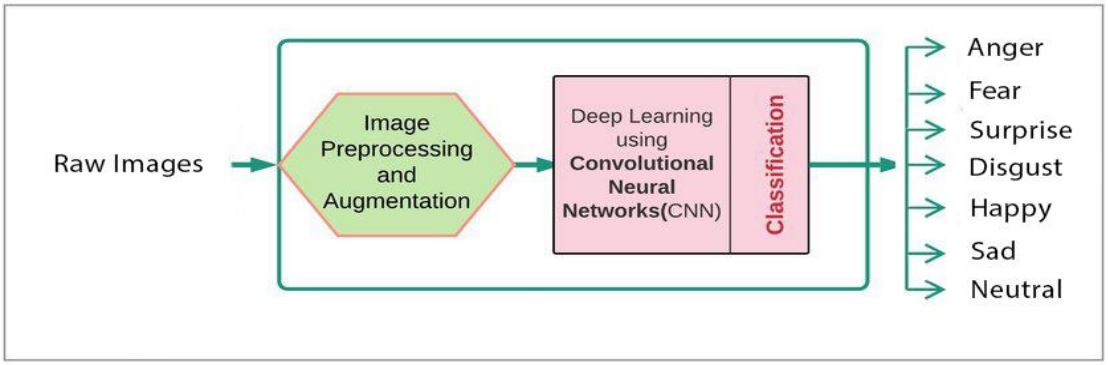
- Web Camera.
- A computer/ laptop with basic configuration as most of the work is in cloud.
- A graphics driver will be an add-on.

CHAPTER 4: SYSTEM DESIGN AND IMPLEMENTATION

4.1 FLOW CHART DIAGRAM



4.2 DESIGN



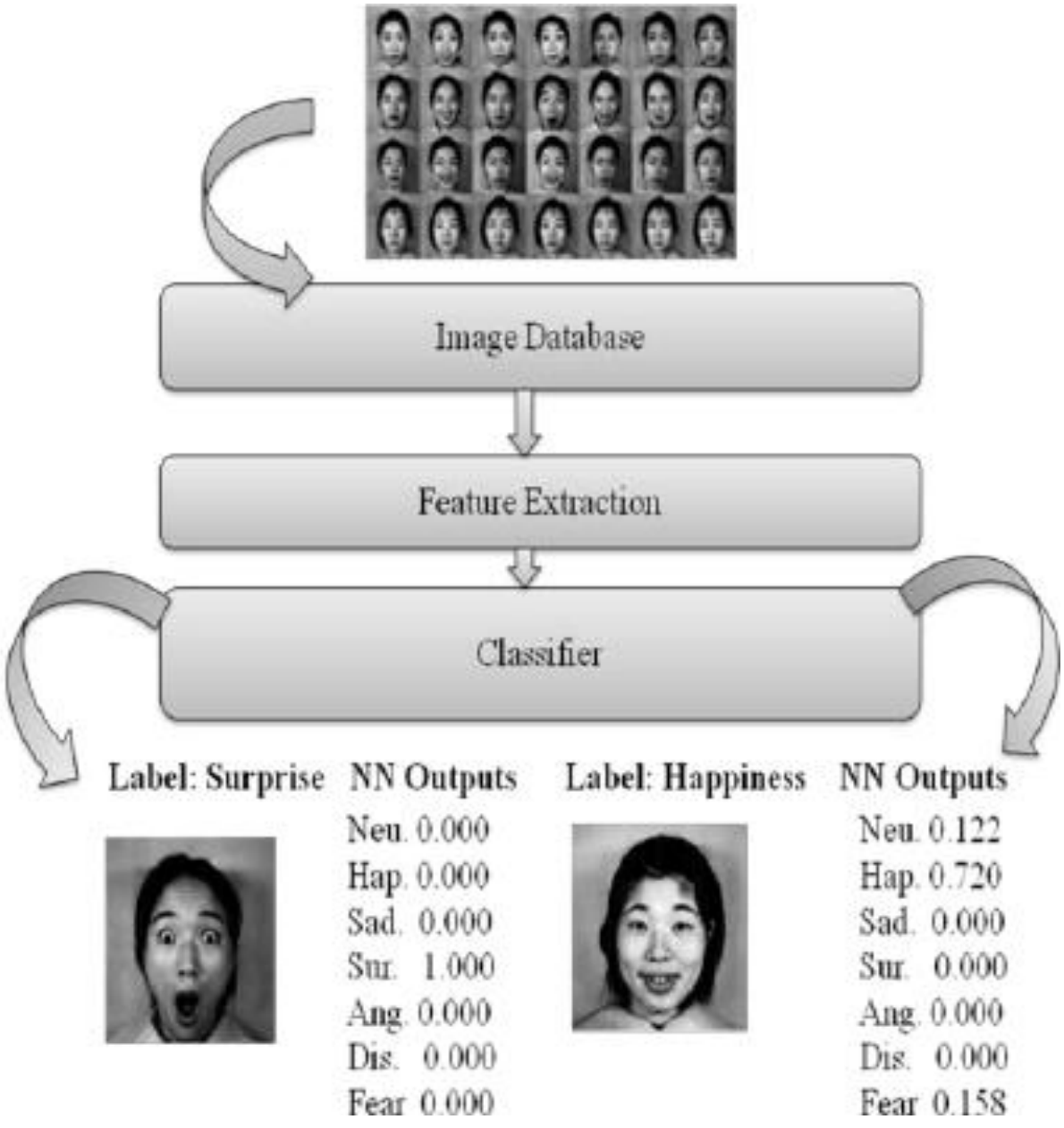
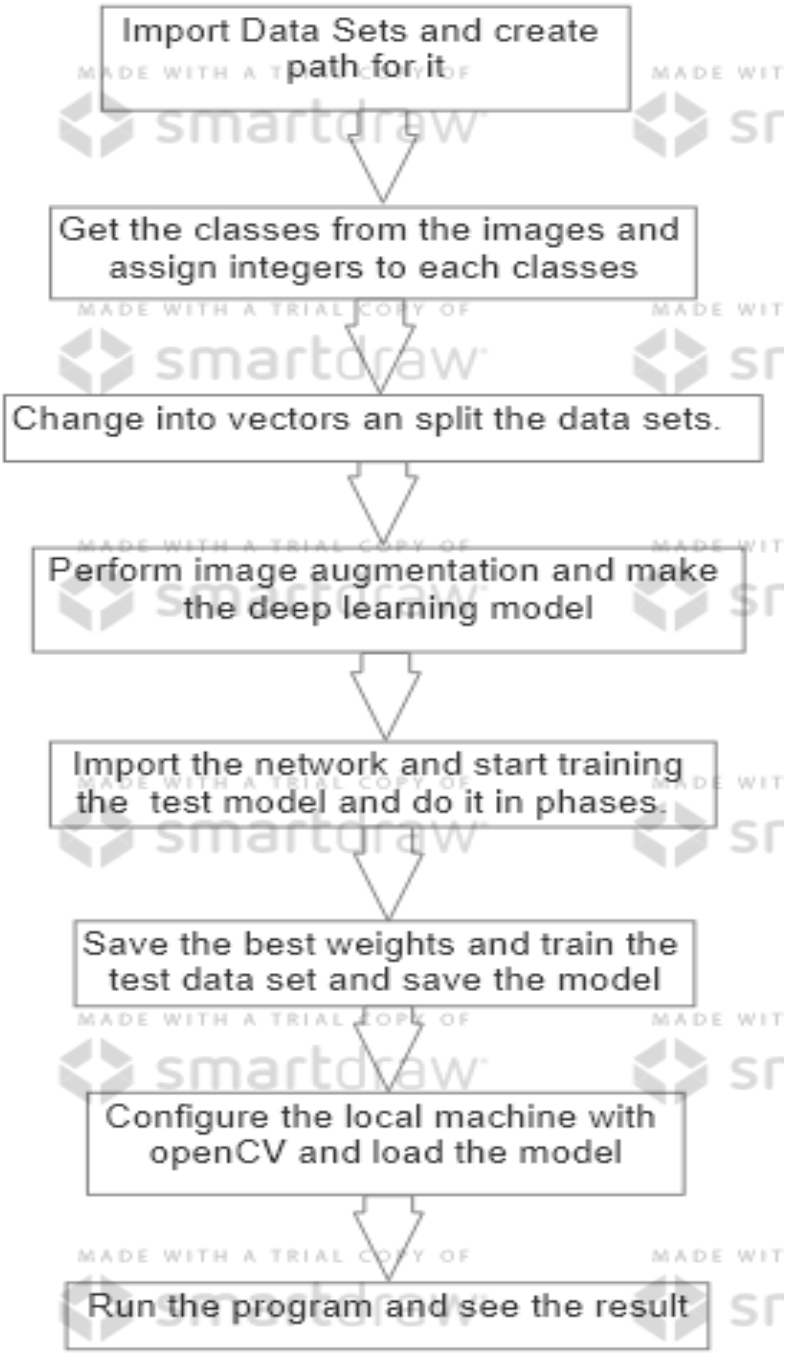


Image Database
Feature Extraction
Classifier
Label: Surprise
NN Outputs
Neu. 0.000
Hap. 0.000
Sad. 0.000
Sur. 1.000
Ang. 0.000
Dis. 0.000
Fear 0.000
Label: Happiness
NN Outputs
Neu. 0.122
Hap. 0.720
Sad. 0.000
Sur. 0.000
Ang. 0.000
Dis. 0.000
Fear 0.158

4.3 BLOCK DIAGRAM



4.4 PYTHON AND LIBRARY

The prerequisites for the project is mentioned here and the theoretical knowledge for making this project a success. This project will identify the expressions based on real time images using different libraries like tensorflow, numpy, pathlib, matplotlib, dlib, opencv and random

module. The whole code is done using python programming knowledge and deep learning concepts. The methods used in deep learning have mathematical significance and statistical relevance.

Python is a General purpose programming language used in all fields. Mainly most of the work of AI and related fields are done using python and its different functional modules.

- Tensorflow- A library used for building tensors in deep learning projects.
- Numpy- It's the numerical python usually used for fast calculations and computations.
- Matplotlib- The plotting function module used for all visual representations.
- Pathlib- Its main work is to work with paths of files and its locations.
- Random-For generating random numbers and shuffling.
- Dlib- It's a super-fast modern C++ toolkit containing ML algorithms.
- OpenCV- This is used for accessing webcam and other functionalities.
- Make sure to work with a version of python that supports OpenCV and tensorflow as not all versions support it. The developer had to downgrade to 3.6.8 version of python to have a feasible test case of this project.
- Due to change in the version of library modules in Colab and Jupyter some of the codes might show error.

4.5 DEEP LEARNING AND CNN

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called **artificial neural networks.** Deep learning is usually subdivided into CNN, ANN, RNN and GAN. In this project we will only use the modules of CNN and ANN. Deep learning is used to solve complex problems which simulates real world and imitates the way human gains knowledge. Deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction while traditional ML models are linear.

CNN, A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data. CNNs are powerful image processing, artificial intelligence (AI) that use deep learning to perform both generative and descriptive tasks, often using machine vison that includes image and video recognition,

4.6 GOOGLE COLAB AND JUPYTER NOTEBOOK

Google Colaboratory is a cloud platform for ML and DL enthusiasts for training their model and get results. It works on the high end systems provided by google. It by defaults updates to the most recent modules and libraries of python for better working with the project.

Jupyter notebook is the local version of google Colab. It works on local system using local hardware and software. It simulates the Colab environment and work in a similar way.

CHAPTER 5: TESTING

5.1 CONCEPT OF USING PHASES

Training a large pre-trained network on large dataset with more classes is a very time consuming activity and there are very high chances that the models will overfit. Neurons tend to interdepend on each other with time so we perform dropouts and better functions. So to avoid these issues, we train our model in two phases. In phase one, we train our whole network for 13 epochs. Here we tune the weights of all the layers of our model. In this phase, we expect our convolutional layers to learn the patterns in the current data. In phase two, we freeze all the layers of our model and only train the last layers where we do the classification.

This phase gives more importance to the classification layer, thus making it more accurate.

PHASE-1

```
Epoch 1/13
689/689 [==============================] - 536s 739ms/step - loss: 3.3714 - accuracy: 0.4441 - precision: 0.7436 - recall: 0.2013 - val_loss: 1.2104 - val_accuracy: 0.5773 -
Epoch 2/13
689/689 [==============================] - 512s 740ms/step - loss: 2.9394 - accuracy: 0.5339 - precision: 0.7699 - recall: 0.3004 - val_loss: 1.2157 - val_accuracy: 0.5593 -
Epoch 3/13
689/689 [==============================] - 510s 740ms/step - loss: 2.8009 - accuracy: 0.5549 - precision: 0.7690 - recall: 0.3367 - val_loss: 1.1409 - val_accuracy: 0.5728 -
Epoch 4/13
689/689 [==============================] - 510s 740ms/step - loss: 2.7211 - accuracy: 0.5641 - precision: 0.7639 - recall: 0.3605 - val_loss: 1.2452 - val_accuracy: 0.5431 -
Epoch 5/13
689/689 [==============================] - 509s 739ms/step - loss: 2.6608 - accuracy: 0.5734 - precision: 0.7635 - recall: 0.3776 - val_loss: 1.1842 - val_accuracy: 0.5450 -
Epoch 6/13
689/689 [==============================] - 503s 730ms/step - loss: 2.5966 - accuracy: 0.5808 - precision: 0.7625 - recall: 0.3910 - val_loss: 1.1879 - val_accuracy: 0.5698 -
Epoch 7/13
689/689 [==============================] - 505s 733ms/step - loss: 2.5439 - accuracy: 0.5856 - precision: 0.7595 - recall: 0.4054 - val_loss: 1.1466 - val_accuracy: 0.5771 -
Epoch 8/13
689/689 [==============================] - 506s 734ms/step - loss: 2.4930 - accuracy: 0.5927 - precision: 0.7580 - recall: 0.4178 - val_loss: 1.1423 - val_accuracy: 0.5680 -
Epoch 9/13
689/689 [==============================] - 506s 734ms/step - loss: 2.4409 - accuracy: 0.5989 - precision: 0.7499 - recall: 0.4318 - val_loss: 1.0782 - val_accuracy: 0.6043 -
Epoch 10/13
689/689 [==============================] - 505s 732ms/step - loss: 2.3869 - accuracy: 0.6049 - precision: 0.7512 - recall: 0.4429 - val_loss: 1.1228 - val_accuracy: 0.5859 -
Epoch 11/13
689/689 [==============================] - 507s 736ms/step - loss: 2.3440 - accuracy: 0.6080 - precision: 0.7500 - recall: 0.4554 - val_loss: 1.2079 - val_accuracy: 0.5484 -
Epoch 12/13
689/689 [==============================] - 507s 736ms/step - loss: 2.2993 - accuracy: 0.6119 - precision: 0.7457 - recall: 0.4623 - val_loss: 1.0885 - val_accuracy: 0.5985 -
Epoch 13/13
689/689 [==============================] - 504s 731ms/step - loss: 2.2498 - accuracy: 0.6170 - precision: 0.7437 - recall: 0.4739 - val_loss: 1.0861 - val_accuracy: 0.5935 -
```

PHASE-2

```
Epoch 1/8
689/689 [==============================] - ETA: 0s - loss: 2.1993 - accuracy: 0.6211 - precision: 0.7409 - recall: 0.4833
Epoch 00001: val_loss improved from inf to 1.11286, saving model to best_weights.h5
689/689 [==============================] - 502s 728ms/step - loss: 2.1993 - accuracy: 0.6211 - precision: 0.7409 - recall:
Epoch 2/8
689/689 [==============================] - ETA: 0s - loss: 2.1382 - accuracy: 0.6291 - precision: 0.7448 - recall: 0.4963
Epoch 00002: val_loss did not improve from 1.11286
689/689 [==============================] - 500s 726ms/step - loss: 2.1382 - accuracy: 0.6291 - precision: 0.7448 - recall:
Epoch 3/8
689/689 [==============================] - ETA: 0s - loss: 2.0937 - accuracy: 0.6318 - precision: 0.7412 - recall: 0.5043
Epoch 00003: val_loss did not improve from 1.11286
689/689 [==============================] - 502s 729ms/step - loss: 2.0937 - accuracy: 0.6318 - precision: 0.7412 - recall:
Epoch 4/8
689/689 [==============================] - ETA: 0s - loss: 2.0444 - accuracy: 0.6414 - precision: 0.7455 - recall: 0.5208
Epoch 00004: val_loss did not improve from 1.11286
689/689 [==============================] - 503s 730ms/step - loss: 2.0444 - accuracy: 0.6414 - precision: 0.7455 - recall:
Epoch 5/8
689/689 [==============================] - ETA: 0s - loss: 1.9905 - accuracy: 0.6448 - precision: 0.7454 - recall: 0.5292
Epoch 00005: val_loss did not improve from 1.11286
689/689 [==============================] - 506s 735ms/step - loss: 1.9905 - accuracy: 0.6448 - precision: 0.7454 - recall:
```

5.2 USING EPOCHS AND BATCH SIZE

Using different epochs we get different accuracy and loss. More epochs is not always good as model might overfit and dropouts are equally important

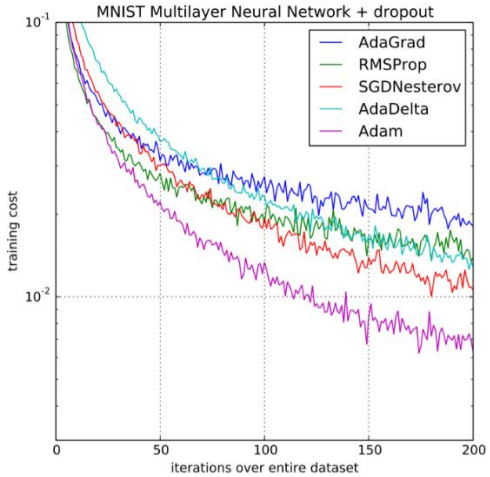
Batch size can be anything for more batch size the number of batch decreases and for less batch size the no of batches increases.

EPOCHS

```
Epoch 1/10
689/689 [==============================] - ETA: 0s - loss: 1.5933 - accuracy: 0.6954 - precision: 0.7618 - recall: 0.6178
Epoch 00001: val_loss improved from inf to 1.17626, saving model to best_weights.h5
689/689 [==============================] - 514s 745ms/step - loss: 1.5933 - accuracy: 0.6954 - precision: 0.7618 - recall: 0.6178 - val_loss: 1.1763 - val_accuracy: 0.5968 - val_prec
Epoch 2/10
689/689 [==============================] - ETA: 0s - loss: 1.5340 - accuracy: 0.7017 - precision: 0.7642 - recall: 0.6271
Epoch 00002: val_loss improved from 1.17626 to 1.16100, saving model to best_weights.h5
689/689 [==============================] - 509s 739ms/step - loss: 1.5340 - accuracy: 0.7017 - precision: 0.7642 - recall: 0.6271 - val_loss: 1.1610 - val_accuracy: 0.6102 - val_prec
Epoch 3/10
689/689 [==============================] - ETA: 0s - loss: 1.5043 - accuracy: 0.7049 - precision: 0.7690 - recall: 0.6327
Epoch 00003: val_loss did not improve from 1.16100
689/689 [==============================] - 508s 738ms/step - loss: 1.5043 - accuracy: 0.7049 - precision: 0.7690 - recall: 0.6327 - val_loss: 1.2195 - val_accuracy: 0.5846 - val_prec
Epoch 4/10
689/689 [==============================] - ETA: 0s - loss: 1.4482 - accuracy: 0.7138 - precision: 0.7715 - recall: 0.6441
Epoch 00004: val_loss did not improve from 1.16100
689/689 [==============================] - 505s 733ms/step - loss: 1.4482 - accuracy: 0.7138 - precision: 0.7715 - recall: 0.6441 - val_loss: 1.1785 - val_accuracy: 0.6111 - val_prec
Epoch 5/10
689/689 [==============================] - ETA: 0s - loss: 1.4171 - accuracy: 0.7181 - precision: 0.7748 - recall: 0.6522
Epoch 00005: val_loss did not improve from 1.16100
689/689 [==============================] - 506s 734ms/step - loss: 1.4171 - accuracy: 0.7181 - precision: 0.7748 - recall: 0.6522 - val_loss: 1.2191 - val_accuracy: 0.5976 - val_prec
Epoch 6/10
689/689 [==============================] - ETA: 0s - loss: 1.3808 - accuracy: 0.7244 - precision: 0.7773 - recall: 0.6608
Epoch 00006: val_loss did not improve from 1.16100
689/689 [==============================] - 508s 737ms/step - loss: 1.3808 - accuracy: 0.7244 - precision: 0.7773 - recall: 0.6608 - val_loss: 1.2530 - val_accuracy: 0.5920 - val_prec
```

5.3 OPTIMIZATION AND ACCURACY

The model has been optimized using ADAM optimization technique and the accuracy increases with no of epochs.



Dropouts have been taken as 30% of the total neurons. Accuracy table is given below (for 13 epochs)

No.of.Epochs	Accuracy	Loss
5	0.5734	2.6608
6	0.5808	2.5966
7	0.5856	2.5439

8	0.5927	2.4930
9	0.5989	2.4409
10	0.6049	2.3869
11	0.6080	2.3440
12	0.6119	2.2993
13	0.6170	2.2498

MODEL SUMMARY ON TESTING

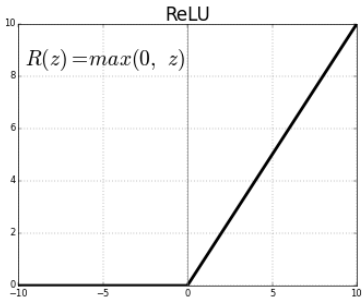
```
# Evaluating the loaded model
loss, acc, prec, rec = model.evaluate(test_dataset)

print(" Testing Loss :" , loss)
print(" Testing Acc : " , acc)
print(" Testing Precision " , prec)
print(" Testing Recall " , rec)
```

```
136/136 [==============================] - 30s 191ms/step - loss: 1.3951 - accuracy: 0.5127 - precision: 0.5645 - recall: 0.4382
 Testing Loss : 1.3950546979904175
 Testing Acc :  0.5126757025718689
 Testing Precision  0.5644946098327637
 Testing Recall  0.4382346272468567
```

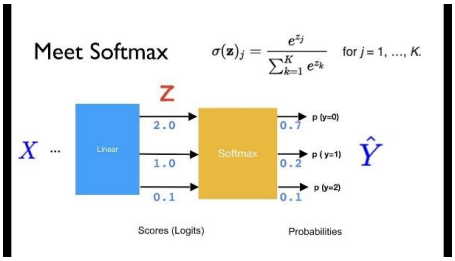
7.4 ACTIVATION TESTING

ReLu has been used for the second last neurons in CNN and softmax is used in the last layer of CNN for multiclass classification.

```
model = tf.keras.Sequential([ #to join multiple tensorflow graphs sequentially
    backbone,
    tf.keras.layers.GlobalAveragePooling2D(), #will average all the signals over final convolutional block
    tf.keras.layers.Dropout(0.3), #dropout
    tf.keras.layers.Dense(128, activation='relu'), #for generalisation
    tf.keras.layers.Dense(7, activation='softmax') #7 for 7 classes(labels) and neurons
])   #since multiclass so softmax
```



ReLu is used as it does not take negative values. It rounds up to integer values that are not negative.



CHAPTER 6: CONCLUSION

So in this project we have learned how deep learning helps us with solving high level complex computational problems consisting of images. We learned to build Convolutional Neural Networks and use that model to perform real time predictions. Used OpenCV and Dlib to connect web camera and perform face detection. Performed Data Augmentation. Performed two phased training on our model where we have frozen previous few layers

The model in future, can perform better by phase addition and cropping for better result cleaning the images and using better resolution images can yield better result when padding and pooling

Can increase number of phases and decrease no of epochs per phase while training. Trying out better versions of efficient netb7 for better result.

APPENDIX I: SCREENSHOTS

FIGURE 1: DATASET FROM KAGGLE

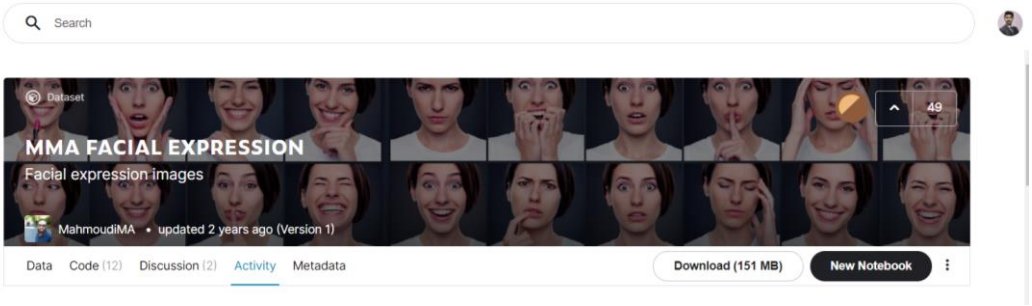
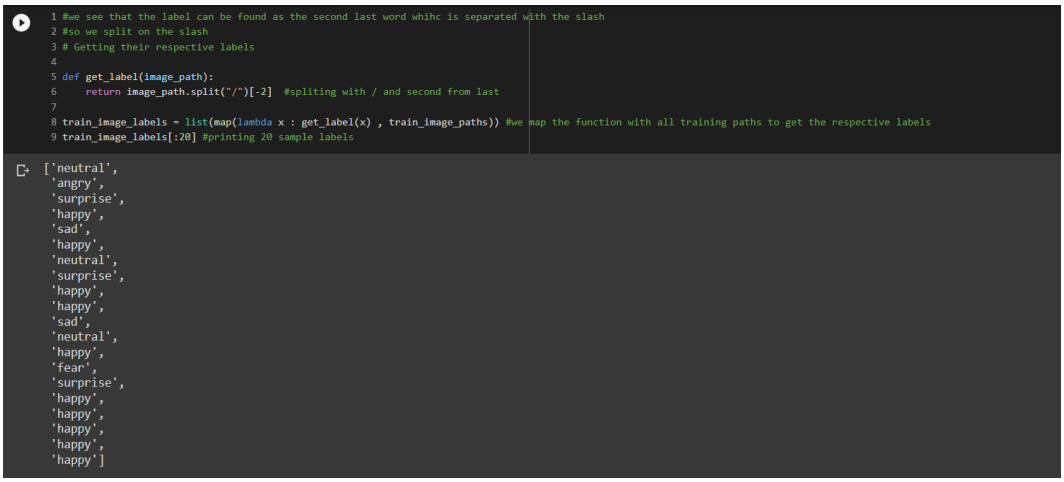


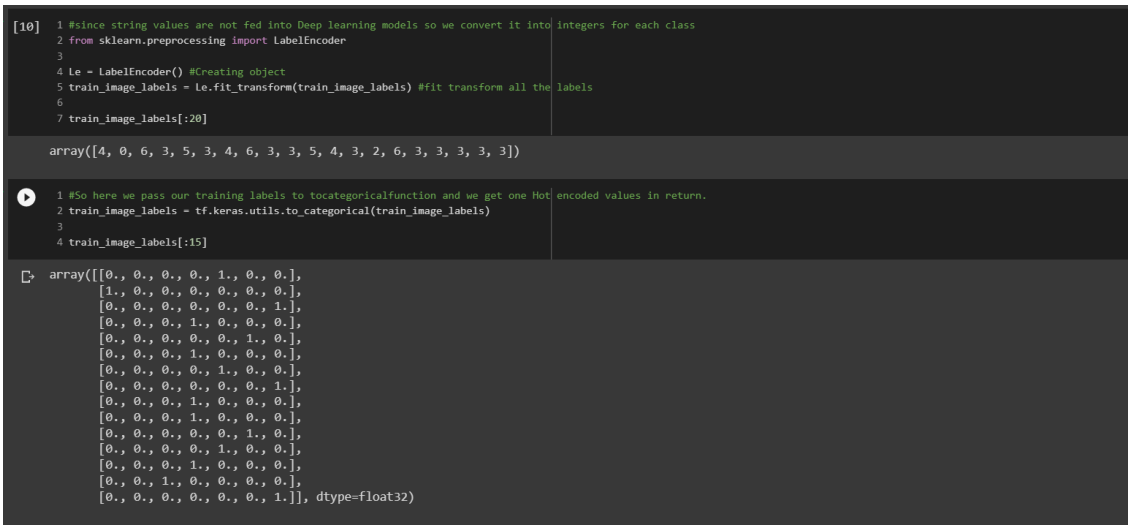
FIGURE 2: CLASSES OF IMAGES ACCORDING TO NAMES IN TRAIN SET



```
#we see that the label can be found as the second last word whihc is separated with the slash
#so we split on the slash
# Getting their respective labels

def get_label(image_path):
    return image_path.split("/")[-2]  #spliting with / and second from last

train_image_labels = list(map(lambda x : get_label(x) , train_image_paths)) #we map the function with all training paths to get the respective labels
train_image_labels[:20] #printing 20 sample labels
```

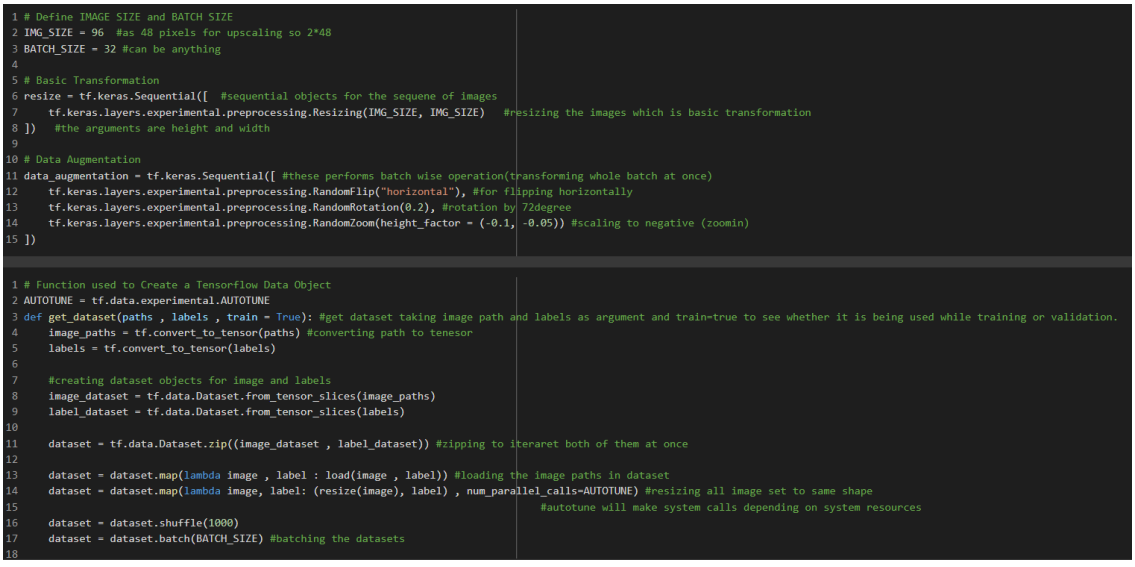
```
['neutral',
 'angry',
 'surprise',
 'happy',
 'sad',
 'happy',
 'neutral',
 'surprise',
 'happy',
 'happy',
 'sad',
 'neutral',
 'happy',
 'fear',
 'surprise',
 'happy',
 'happy',
 'happy',
 'happy',
 'happy']
```

FIGURE 3: ENCODING STRINGS TO INTEGERS FOR CATEGORY AND MAPPING IT IN THE VECTOR FORM.



```
[10] 1 #since string values are not fed into Deep learning models so we convert it into integers for each class
     2 from sklearn.preprocessing import LabelEncoder
     3
     4 le = LabelEncoder() #Creating object
     5 train_image_labels = le.fit_transform(train_image_labels) #fit transform all the labels
     6
     7 train_image_labels[:20]

array([4, 0, 6, 3, 5, 3, 4, 6, 3, 3, 5, 4, 3, 2, 6, 3, 3, 3, 3, 3])

  1 #So here we pass our training labels to tocategoricalfunction and we get one Hot encoded values in return.
  2 train_image_labels = tf.keras.utils.to_categorical(train_image_labels)
  3
  4 train_image_labels[:15]

array([[0., 0., 0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1.]], dtype=float32)
```

FIGURE 4: IMAGE AUGMENTATION AND FEEDING IT AS TENSOR OBJECT DECLARING THE BATCH SIZE



```
# Define IMAGE SIZE and BATCH SIZE
IMG_SIZE = 96  #as 48 pixels for upscaling so 2*48
BATCH_SIZE = 32 #can be anything

# Basic Transformation
resize = tf.keras.Sequential([  #sequential objects for the sequene of images
    tf.keras.layers.experimental.preprocessing.Resizing(IMG_SIZE, IMG_SIZE)   #resizing the images which is basic transformation
])   #the arguments are height and width

# Data Augmentation
data_augmentation = tf.keras.Sequential([ #these performs batch wise operation(transforming whole batch at once)
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"), #for flipping horizontally
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.2), #rotation by 72degree
    tf.keras.layers.experimental.preprocessing.RandomZoom(height_factor = (-0.1, -0.05)) #scaling to negative (zoomin)
])
```

```
# Function used to Create a Tensorflow Data Object
AUTOTUNE = tf.data.experimental.AUTOTUNE
def get_dataset(paths , labels , train = True): #get dataset taking image path and labels as argument and train=true to see whether it is being used while training or validation.
    image_paths = tf.convert_to_tensor(paths) #converting path to tenesor
    labels = tf.convert_to_tensor(labels)

    #creating dataset objects for image and labels
    image_dataset = tf.data.Dataset.from_tensor_slices(image_paths)
    label_dataset = tf.data.Dataset.from_tensor_slices(labels)

    dataset = tf.data.Dataset.zip((image_dataset , label_dataset)) #zipping to iteraret both of them at once

    dataset = dataset.map(lambda image , label : load(image , label)) #loading the image paths in dataset
    dataset = dataset.map(lambda image, label: (resize(image), label) , num_parallel_calls=AUTOTUNE) #resizing all image set to same shape
                                                                        #autotune will make system calls depending on system resources
    dataset = dataset.shuffle(1000)
    dataset = dataset.batch(BATCH_SIZE) #batching the datasets
```

FIGURE 5: SAMPLE IMAGE WITH ITS CORRECTLY PREDICTED CLASS

```
[20]  1 # View a sample Validation Image
      2 print(Le.inverse_transform(np.argmax(label , axis = 1))[0])
      3 plt.imshow((image[0].numpy()/255).reshape(96 , 96 , 3))

      neutral
      <matplotlib.image.AxesImage at 0x7f8f3ad74e90>
```

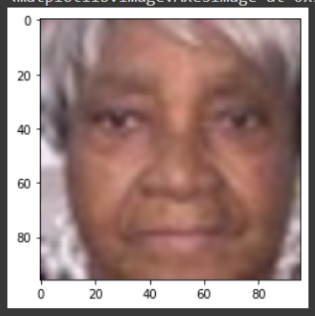


FIGURE 6: EVALUATING IN TEST DATASET

```
 1 from tensorflow.keras.applications import EfficientNetB2
 2
 3 backbone = EfficientNetB2(
 4     input_shape=(96, 96, 3),
 5     include_top=False
 6 )
 7
 8 model = tf.keras.Sequential([
 9     backbone,
10     tf.keras.layers.GlobalAveragePooling2D(),
11     tf.keras.layers.Dropout(0.3),
12     tf.keras.layers.Dense(128, activation='relu'),
13     tf.keras.layers.Dense(7, activation='softmax')
14 ])
15
16 model.compile(
17     optimizer=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07),
18     loss = 'categorical_crossentropy',
19     metrics=['accuracy' , tf.keras.metrics.Precision(name='precision'),tf.keras.metrics.Recall(name='recall')]
20 )
```

```
[29]  1 model.load_weights("best_weights.h5")
```

FIGURE 7: ACCURACY, PRECISION AND RECALL AND SAVING MODEL.

```
[33]  1 # Evaluating the loaded model
      2 loss, acc, prec, rec = model.evaluate(test_dataset)
      3
      4 print(" Testing Acc : " , acc)
      5 print(" Testing Precision " , prec)
      6 print(" Testing Recall " , rec)

      543/543 [==============================] - 39s 63ms/step - loss: 1.3369 - accuracy: 0.4696 - precision: 0.6415 - recall: 0.2490
       Testing Acc :  0.46963587403297424
       Testing Precision  0.6414785981178284
       Testing Recall  0.2489628940820694
```

```
[34]  1 # Save Model
      2 model.save("FacialExpressionModel.h5")

      /usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom mask layers require a config and must
        layer_config = serialize_layer_fn(layer)
```

```
 1 # Save Label Encoder
 2 import pickle
 3
 4 def save_object(obj , name):
 5     pickle_obj = open(f"{name}.pck","wb")
 6     pickle.dump(obj, pickle_obj)
 7     pickle_obj.close()
```

```
[36]  1 save_object(Le, "LabelEncoder")
```

FIGURE 8: UPLOADING MODEL AND WEIGHTS IN LOCAL SYSTEM

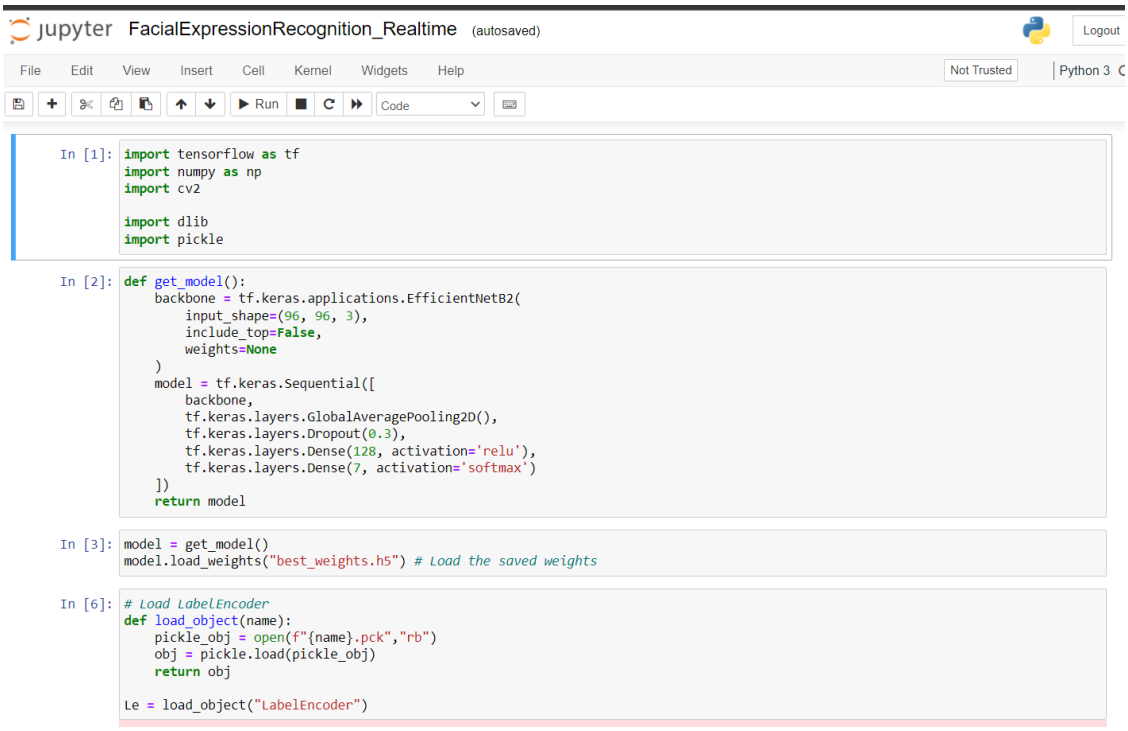
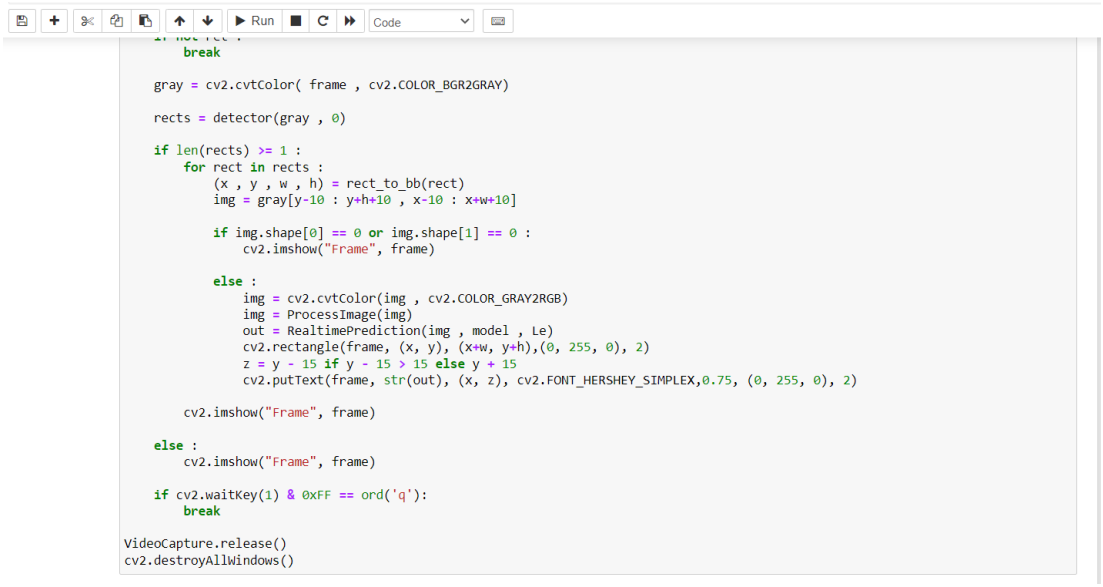
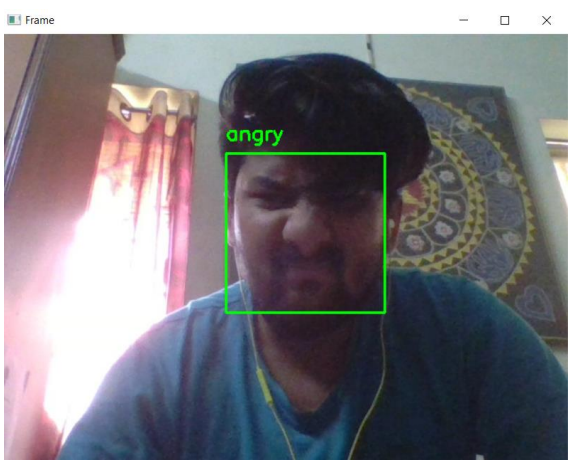
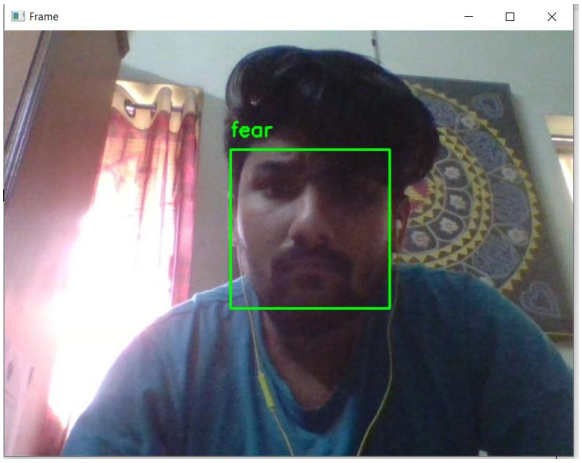


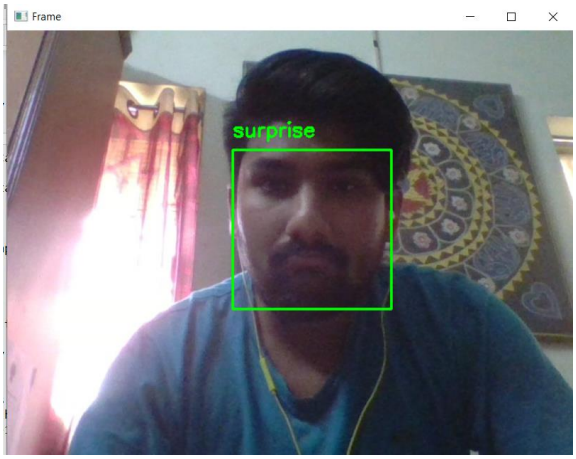
FIGURE 9: ACCESSING THE WEB CAMERA AND DETECTING EXPRESSION



OUTPUT







BIBLIOGRAPHY

A great blog with code by Taus Noor

- https://towardsdatascience.com/facial-recognition-using-deep-learning-a74e9059a150

A platform for learning and capstone projects

- https://www.dataisgood.com/

A literature survey on face detection

- https://machinelearningmastery.com/introduction-to-deep-learning-for-face-recognition/

A code for facial expression recognition with explanation

- https://www.pyimagesearch.com/2018/06/18/face-recognition-with-opencv-python-and-deep-learning/

How face reader works by Noldus

- https://www.noldus.com/facereader