

**A Project Report
on
FACE TO EMOJI USING OPEN CV AND HAAR CASCADE
CLASSIFIER**

**submitted in partial fulfillment of the requirements for the award of the degree
of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING**

by

17WH1A0531	Ms. M.SREE HARIKA
17WH1A0541	Ms. T.ADHI LAKSHMI
18WH5A0501	Ms. G.SOWMYA

under the esteemed guidance of

**Ms.B.NAGAVENI
Assistant Professor**



**Department of Computer Science and Engineering
BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090**

May, 2021

DECLARATION

We hereby declare that the work presented in this project entitled "**FACE TO EMOJI USING OPEN CV AND HAAR CASCADE CLASSIFIER**" submitted towards completion of Project Work in IV year of B.Tech., CSE at 'BVRIT HYDERABAD College of Engineering For Women', Hyderabad is an authentic record of our original work carried out under the guidance of Ms.B.Nagaveni, Assistant Professor, Department of CSE.

Sign. with date:

Ms. M.SREE HARIKA
(17WH1A0531)

Sign. with date:

Ms. T.ADHI LAKSHMI
(17WH1A0541)

Sign. with date:

Ms. G.SOWMYA
(18WH5A0501)

BVRIT HYDERABAD
College of Engineering for Women
(NBA Accredited – EEE, ECE, CSE and IT)
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Bachupally, Hyderabad – 500090

Department of Computer Science and Engineering



Certificate

This is to certify that the Project Work report on "**FACE TO EMOJI USING OPEN CV AND HAAR CASCADE CLASSIFIER**" is a bonafide work carried out by Ms. M.SREE HARIKA (17WH1A0531) ; Ms. T.ADHI LAKSHMI (17WH1A0541) ; Ms. G.SOWMYA (18WH5A0501) in the partial fulfillment for the award of B.Tech. degree in **Computer Science and Engineering, BVRIT HYDERABAD College of Engineering for Women, Bachupally, Hyderabad**, affiliated to Jawaharlal Nehru Technological University Hyderabad, Hyderabad under my guidance and supervision.

The results embodied in the project work have not been submitted to any other University or Institute for the award of any degree or diploma.

Head of the Department
Dr. K.Srinivasa Reddy
Professor and HoD,
Department of CSE

Guide
Ms. B.Nagaveni
Assistant Professor

External Examiner

Acknowledgements

We would like to express our sincere thanks to **Dr. K V N Sunitha, Principal, BVRIT HYDERABAD College of Engineering for Women**, for providing the working facilities in the college.

Our sincere thanks and gratitude to our **Dr. K.Srinivasa Reddy, Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for all the timely support and valuable suggestions during the period of our project.

We are extremely thankful and indebted to our internal guide, **Ms. B.Nagaveni, Assistant Professor, Department of CSE, BVRIT HYDERABAD College of Engineering for Women** for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank our Project Coordinator, all the faculty and staff of **CSE Department** who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

**Ms. M.SREE HARIKA
(17WH1A0531)**

**Ms. T.ADHI LAKSHMI
(17WH1A0541)**

**Ms. G.SOWMYA
(18WH5A0501)**

Contents

S.No.	Topic	Page No.
	Abstract	i
	List of Figures	ii
1.	Introduction	1
	1.1 Objectives	1
	1.2 Methodology	1
	1.2.1 Dataset	1
	1.2.2 The proposed CNN model	3
	1.2.3 Haar Cascade Classifier	7
	1.3 Organization of Project	9
2.	Theoretical Analysis of the proposed project	10
	2.1 Requirements Gathering	10
	2.1.1 Software Requirements	10
	2.1.2 Hardware Requirements	10
	2.2 Technologies Description	10
3.	Design	15
	3.1 Architecture Diagram	15
	3.2 UML Diagrams	16
	3.2.1 Use Case Diagram	17
	3.2.2 Sequence Diagram	17
	3.2.3 Activity Diagram	18
	3.2.4 Collaboration Diagram	19
	3.2.5 Class Diagram	20
4.	Implementation	21
	4.1 Data Preparation	21
	4.2 Model Creation	25
	4.3 Model Building	26
	4.4 Image Capturing	35
	4.5 Facial Expression to Emoji	37

4.6	Test Cases	38
4.7	Dataset Training Screenshots	39
4.8	Input Screenshots	40
4.9	Output Screenshots	42
5.	Conclusion	49
6.	References	50

ABSTRACT

Facial emoji recognizer is an end user application which detects the expression of the person in the video being captured by the camera. The emoji relevant to the expression of the person in the video is shown on the screen which changes with the change in the expressions. Facial expressions are important in human communication and interactions. Also, they are used as an important tool in studies about behavior and in medical fields. Facial emoji recognizer provides a fast and practical approach for non-meddlesome emotion detection. The purpose was to develop an intelligent system for facial based expression classification using CNN algorithm. Haar classifier is used for face detection and CNN algorithm is utilized for the expression detection and giving the emoticon relevant to the expression as the output. The emojis used in this project include Happiness, sadness, fear, anger, disgust, neutral, surprise.

LIST OF FIGURES

S.No.	Fig No.	Fig Name	Page No.
1.	1.2.1.1	Dataset	2
2.	1.2.1.2	Images in the Dataset	3
3.	1.2.2	Convolution Neural Network	4
4.	1.2.2.1	Convolution in CNN	5
5.	1.2.2.2	Max pooling in CNN	5
6.	1.2.2.3	Flattening in CNN	6
7.	1.2.2.4	Full Connection in CNN	6
8.	1.2.3	Calculating Haar features	7
9.	1.2.3.1	Object detection using Haar cascade classifier	8
10.	3.1	Architecture diagram	15
11.	3.2.1	Use Case Diagram	16
12.	3.2.2	Sequence Diagram	17
13.	3.2.3	Activity Diagram	18
14.	3.2.4	Collaboration Diagram	19
15.	3.2.5	Class Diagram	20
16.	4.1.1	Importing Libraries and Reading the Dataset	21
17.	4.1.2	Splitting the data	22
18.	4.1.3	Data Augmentation	23
19.	4.1.4	Displaying Reshaped Images	24
20.	4.2.1	Creating the model	25
21.	4.3.1	Building the model	26
22.	4.3.2	Flattering the model	27
23.	4.3.3	Model Summary	28
24.	4.3.4	Describing the model layers	29
25.	4.3.5	Model Compilation	30
26.	4.3.6	Model training	31
27.	4.3.7	Model Accuracy	32
28.	4.3.8	Plotting Accuracy and Loss graph	33

Face to Emoji using opencv and haar cascade classifier

29.	4.3.9	Plot between accuracy and val accuracy	33
30.	4.3.10	Plot between loss and val loss	34
31.	4.4.1	Starting the Camera	35
32.	4.4.2	Face detection	36
33.	4.5.1	Displaying the emoji	37
34.	4.6.1	Test Cases	38
35.	4.7.1	Training dataset	39
36.	4.8.1	Command to run project	40
37.	4.8.2	Camera Starting	40
38.	4.8.3	Image detection	41
39.	4.9.1	Neutral	42
40.	4.9.2	Happiness	43
41.	4.9.3	Surprise	44
42.	4.9.4	Sadness	45
43.	4.9.5	Anger	46
44.	4.9.6	Disgust	47
45.	4.9.7	Fear	48

1. INTRODUCTION

Emoji's are the one that are used to identify the emotions of the people. Today is the era of fast and dynamic internet and communication technologies. Hence Communication is so easy compared to the past. To enhance the communication emoji's were developed. Emoji's are the pictorial depiction of the facial expression of human beings. This project identifies the expression on human face and converts that expression into emoji using Convolutional neural network and Haar cascade classifier. It includes seven human expressions, that is neutral, fear, anger, happy, sad, disgust and surprise emotions. These expressions are the actual expressions which are being conveyed in human beings.

1.1 Objectives

The main goal of this project is to convert the recognized facial emotion in real time to corresponding emoji. The implementation of this project identifies the facial expression and represents that expression using emoji indicator. Such expressions are neutral, fear, anger, happy, sad, and surprise and disgust. Furthermore, facial expressions are related with the person's behavior and personality. If expressions are good, then it means the person is in a good mood or If the person's expressions show anger or disgust it means that he or she is not feeling well. The expressions are actual predictor of human behavior. Understanding expressions are important because they give a lot of information regarding behavior and moods of people. With this one can know what is going inside the person's mind and how it can be handled.

1.2 Methodology

To Convert the facial expression to emoji a large collection of the different expressions images are required. The images are taken from the FER2013 dataset. In this section the methodology followed is discussed in detail.

1.2.1 Dataset

Proper and large dataset is required for identification and conversion of facial expressions to emoji's during the training and the testing phase. The dataset for the experiment is FER+ 2013 which is downloaded from the Kaggle which contains around 33,000 images. It has total 33,000 images. The Train images are 28,709 images and the Test images are 3,589 images. For every emotion in the train images has around 5000 images and for every emotion

Face to Emoji using opencv and haar cascade classifier

in the test images has around 1000 images.. Here the emotions included are Happiness, Sadness, Angry, Disgust, Fearful, Surprise, Neutral.

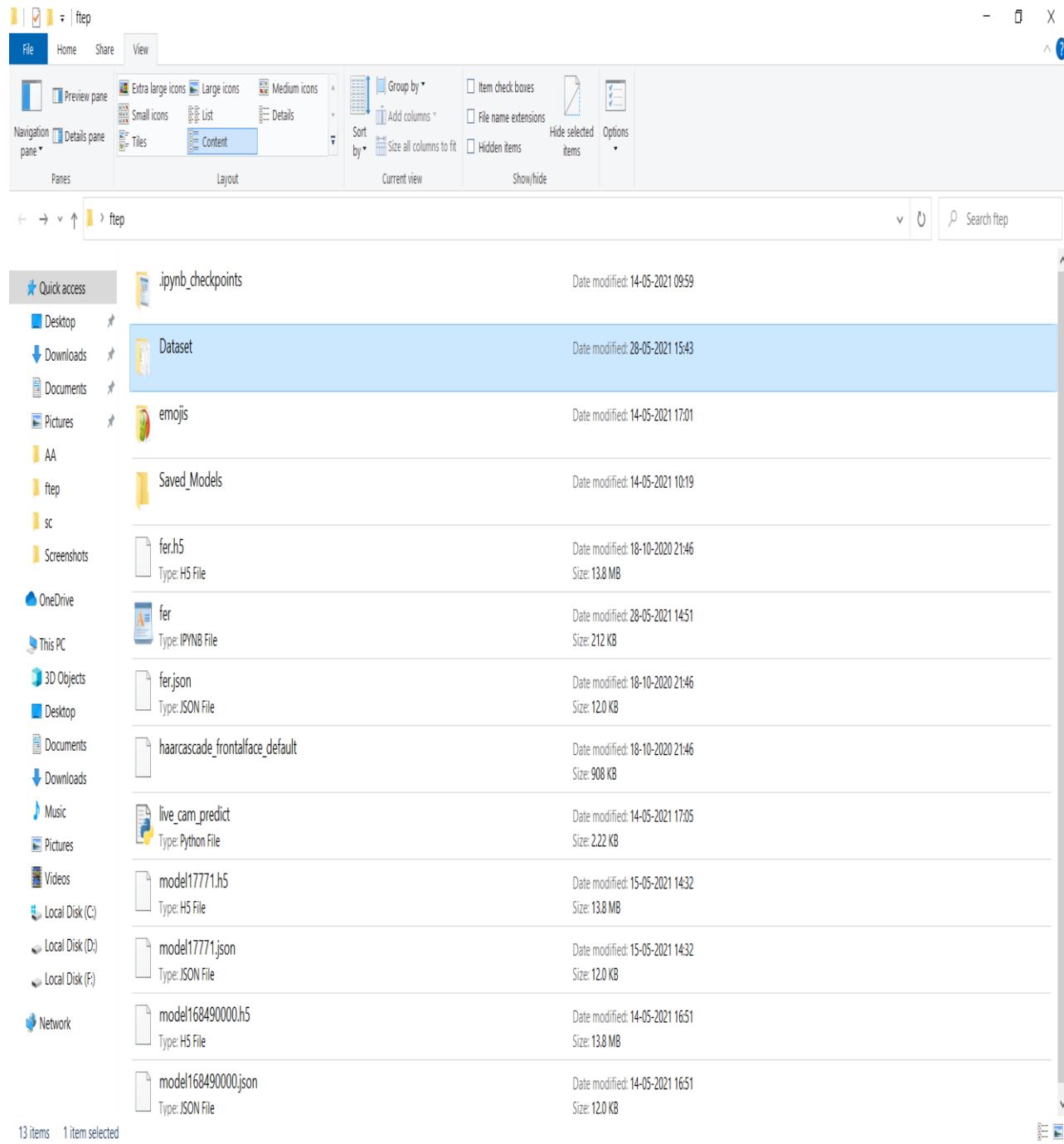


Fig 1.2.1.1: Dataset



Fig 1.2.1.2 : Images in the Dataset

1.2.2 The proposed CNN model

CNN architectures vary with the type of the problem at hand. The proposed model consists of three convolutional layers each followed by a maxpooling layer. The final layer is fully connected MLP. ReLu activation function is applied to the output of every convolutional layer and fully connected layer. The first convolutional layer filters the input image with 32 kernels of size 3x3. After max pooling is applied, the output is given as an input for the second convolutional layer with 64 kernels of size 4x4. The last convolutional layer has 128 kernels of size 1x1 followed by a fully connected layer of 512 neurons. The output of this layer is given to softmax function which produces a probability distribution of the four output class. The model is trained using adaptive moment estimation (Adam) with batch size of 100 for 1000 epochs.

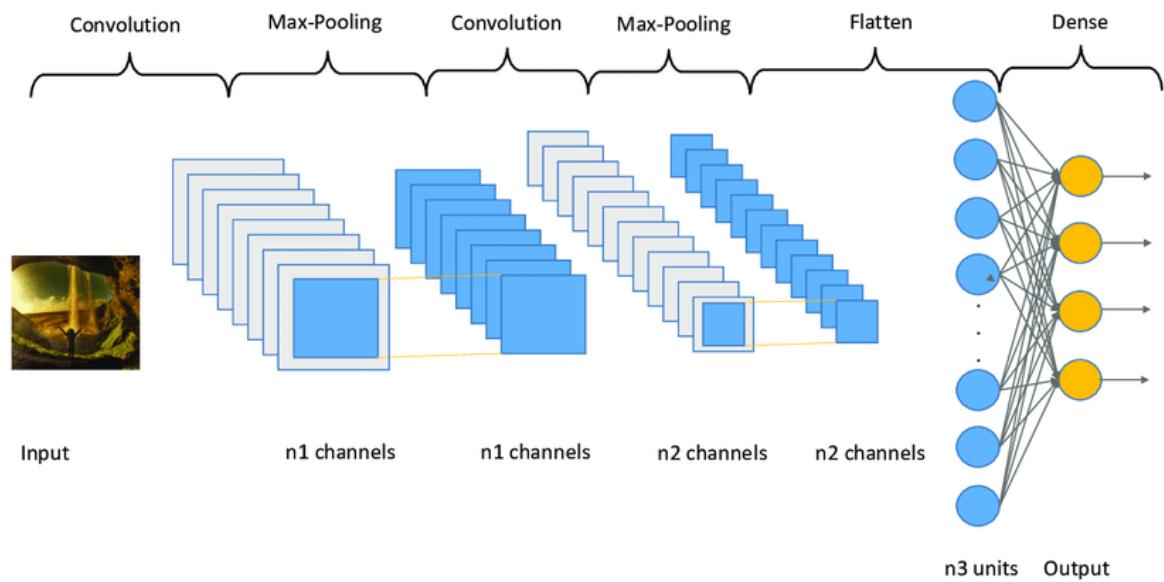


Fig 1.2.2: Convolution Neural Network

There are four CNN algorithm steps,

Convolution: The term convolution refers to the mathematical combination of two functions to produce a third function. It merges two sets of information. In the case of a CNN, the convolution is performed on the input data with the use of a filter or kernel to then produce a feature map.

Here are the three elements that enter into the convolution operation:

- Input image
- Feature detector
- Feature map

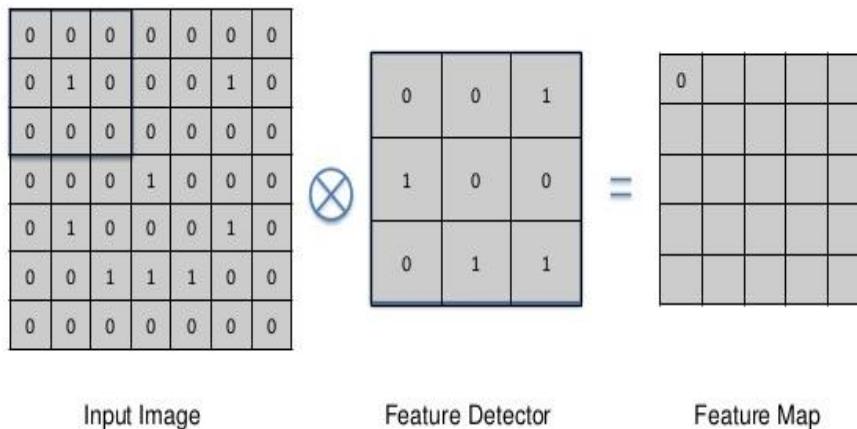


Fig 1.2.2.1: Convolution in CNN

Max pooling: Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

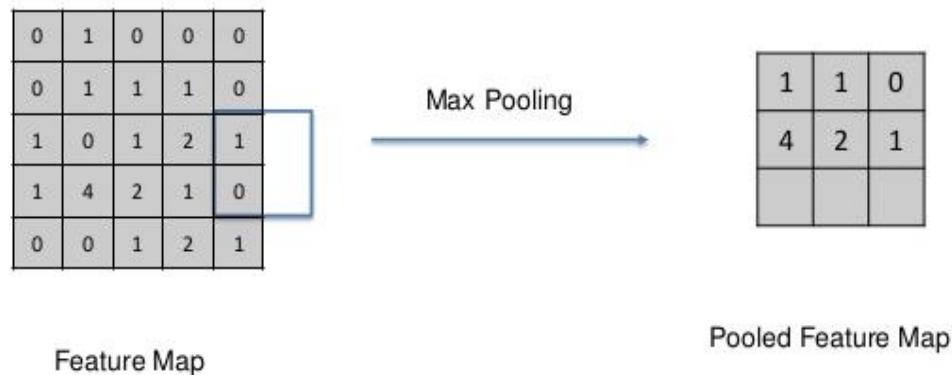


Fig 1.2.2.2: Max Pooling in CNN

Flattening: Flattening is the process of converting all the resultant 2 dimensional arrays into a single long continuous linear vector.

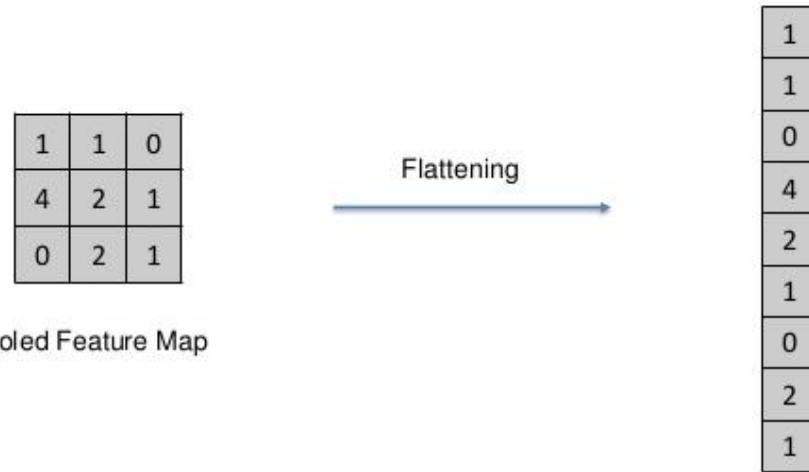


Fig 1.2.2.3: Flattening in CNN

Full Connection: At the end of a CNN, the output of the last Pooling Layer acts as a input to the so called Fully Connected Layer. There can be one or more of these layers (“fully connected” means that every node in the first layer is connected to every node in the second layer).

As you see from the image below, we have three layers in the full connection step:

- Input layer
- Fully-connected layer
- Output layer

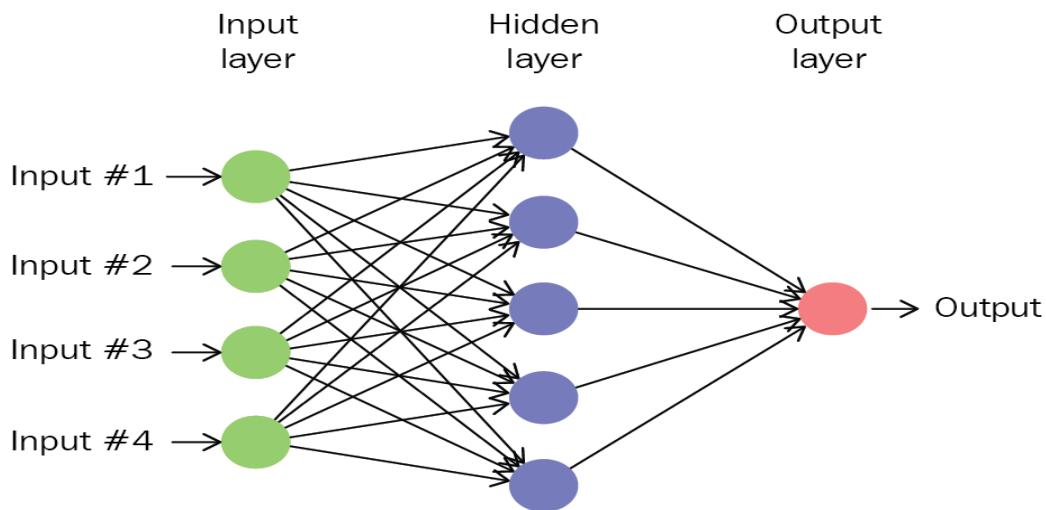


Fig 1.2.2.4: Full Connection in CNN

1.2.3 Haar Cascade Classifier

A Haar classifier, or a Haar cascade classifier, is a machine learning object detection program that identifies objects in an image and video.

The algorithm can be explained in four stages:

1. Calculating Haar Features
2. Creating Integral Images
3. Using Adaboost
4. Implementing Cascading Classifiers

This algorithm requires a lot of positive images of faces and negative images of non-faces to train the classifier, similar to other machine learning models.

Calculating Haar Features

The first step is to collect the Haar features. A Haar feature is essentially calculations that are performed on adjacent rectangular regions at a specific location in a detection window. The calculation involves summing the pixel intensities in each region and calculating the differences between the sums. Here are some examples of Haar features below.

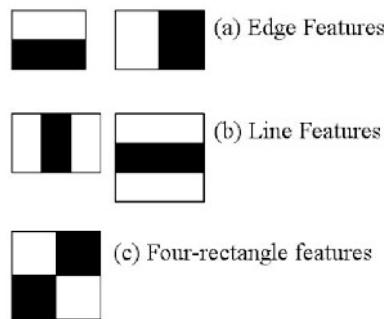


Fig.1.2.3:Calculating Haar features

Creating Integral Images

Without going into too much of the mathematics behind it (check out the paper if you're interested in that), integral images essentially speed up the calculation of these Haar features. Instead of computing at every pixel, it instead creates sub-rectangles and creates array references for each of those sub-rectangles. These are then used to compute the Haar features.

Adaboost Training

Adaboost essentially chooses the best features and trains the classifiers to use them. It uses a combination of “weak classifiers” to create a “strong classifier” that the algorithm can use to detect objects.

Implementing Cascading Classifiers

The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners. The cascade classifier is made up of a series of stages, where each stage is a collection of weak learners. Weak learners are trained using boosting, which allows for a highly accurate classifier from the mean prediction of all weak learners.

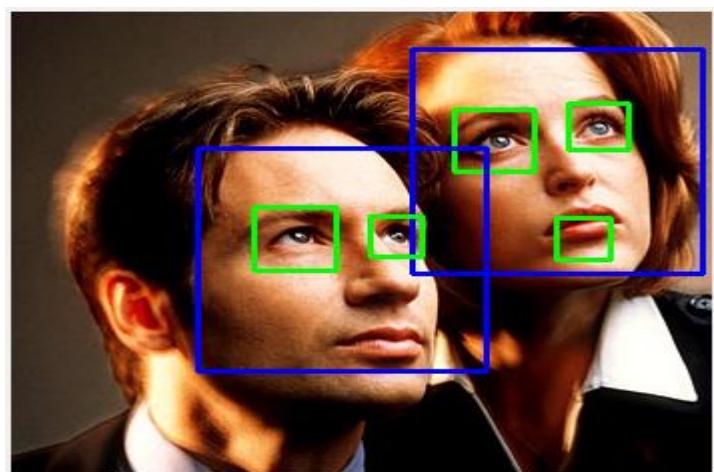


Fig.1.2.3: Object detection using Haar cascade classifier

Applications of Haar Cascades

Facial Recognition: Similar to how iPhone X uses facial recognition, other electronic devices and security protocols can use Haar cascades to determine the validity of the user for secure login.

Agriculture: Haar classifiers can be used to determine whether harmful bugs are flying onto plants, reducing food shortages caused by pests.

Industrial Use: Haar classifiers can be used to allow machines to pick up and recognize certain objects, automating many of the tasks that humans could previously only do.

1.3 Organization of Project

The project which is developed captures image from live video and converts the facial expression detected in captured image to the corresponding emoji using convolutional neural network. We have three modules in our project.

- Capturing the image from live video
- Detecting the expression from the captured image
- Converting the detected expression to the corresponding emoji

2. THEORETICAL ANALYSIS OF THE PROPOSED PROJECT

2.1 Requirements Gathering

2.1.1 Software Requirements

Programming Language : Python 3.6

Dataset : FER+ 2013 Dataset

Packages : Tensorflow, Numpy, Pandas, Matplotlib, Scikit-learn,open cv

Tool : Jupyter Notebook, Pycharm

2.1.2 Hardware Requirements

Operating System: Windows 10/ubuntu 16.04

Processor : Intel Core i3-2348M

CPU Speed : 2.30 GHz

Memory : 8 GB (RAM)

2.2 Technologies Description

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics created by Guido van Rossum. It first released in 1991. Python supports modules and packages, which encourages program modularity and code reuse. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library. Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Biopython and Astropy providing domain-specific functionality.

Python is commonly used in artificial intelligence projects and machine learning projects with the help of libraries like TensorFlow, Keras, Pytorch and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

An empirical study found that scripting languages, such as Python, are more productive than conventional languages, such as C and Java, for programming problems involving string

manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and not much worse than C or C++". Python also acknowledges that speed of development is important. Readable and terse code is part of this, and so is access to powerful constructs that avoid tedious repetition of code. The speed of development, the ease with which a programmer of other languages can pick up basic Python skills and the huge standard library is key to another area where Python excels. All its tools have been quick to implement, saved a lot of time, and several of them have later been patched and updated by people with no Python background - without breaking.

Tensorflow

Tensorflow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs) and is also used for machine learning applications such as neural networks. Tensorflow computations are expressed as stateful dataflow graphs.

Tensorflow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Numpy

NumPy was developed by Travis Oliphant in 2005. It is an open source project and you can use it freely. It is a general-purpose array-processing package. Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows Numpy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is the most powerful and flexible open-source Python Library. It provides high-performance data manipulation and analysis tool using its powerful data structures. Pandas is used to accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc. Few things that pandas performs better:

- It Easily handles the missing data.
- It is fast, powerful and flexible.
- extensively used in production in financial applications.

Matplotlib

Matplotlib is an open source library and python 2D plotting library for creating static, animated, and interactive visualizations in Python. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. One can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code.

One of the best advantage is that it allows us visual access to huge amounts of data in easily digestible visuals. It consists of several plots like line, bar, scatter, histogram etc.

The pyplot module provides a MATLAB-like interface for simple plotting, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn is largely written in Python, and uses NumPy extensively for high-performance linear algebra and array operations. integrates well with many other Python libraries, such as NumPy for array vectorization, Pandas dataframes, SciPy, and many more.

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn.

Scikit-learn is community effort and anyone can contribute to it. It (scikit- learn) is most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python

Open Cv

It is a python library which is used to develop real-time computer vision applications. It helps in detecting the faces and objects from the video. It is a open source library which was built to provide a common infrastructure for computer vision applications. It supports windows, linux, amdriod and mac-os operating systems. It leads mostly towards real-time applications.

Open cv now supports a multitude of algorithms related to computer vision and machine learning. It also supports a variety of programming languages like cpp, python, java etc. Its application areas are facial recognition, human computer interaction object detection and etc.

Jupyter Notebook

The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter Notebooks are a spin-off project from the IPython project, which used to have an IPython Notebook project itself. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

Jupyter Notebooks are an open source web application that can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, etc.

PyCharm

PyCharm is an integrated development environment (IDE) used in computer programming supports data science with Anaconda and web development with django or flask.

It is developed by the czech company JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems. It is an integrated python debugger which supports scientific tools like matplotlib, numpy and scipy.

3.DESIGN

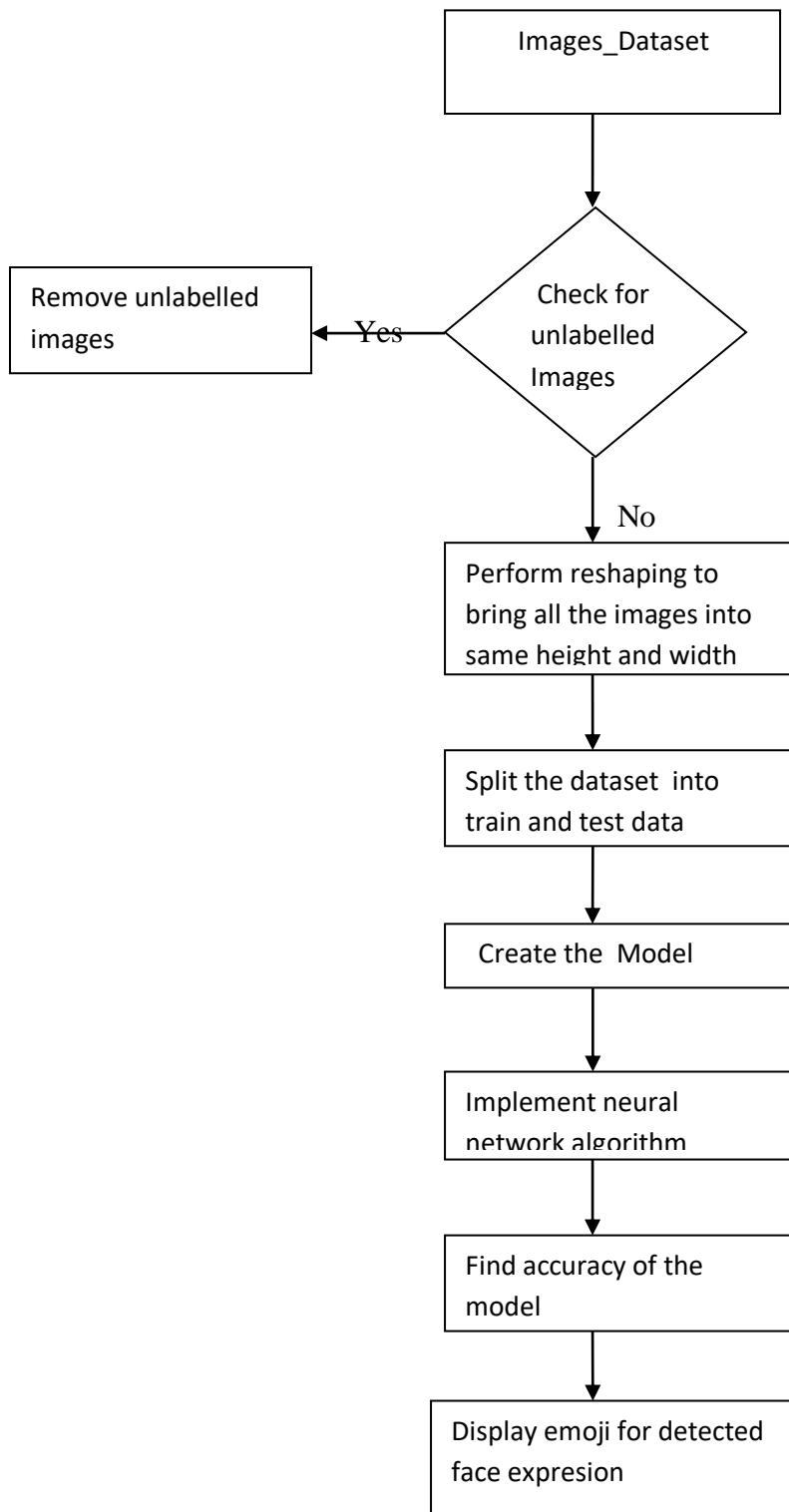


Fig 3.1: Architecture diagram

3.2 UML Diagrams

3.2.1 Use Case Diagram

In UML, there are five diagrams available to model the dynamic nature and use case diagram is one of them. Use case diagrams consist of actors, use cases and their relationships. A single use case diagram captures a particular functionality of a system. To model the entire system, a number of use case diagrams are used.

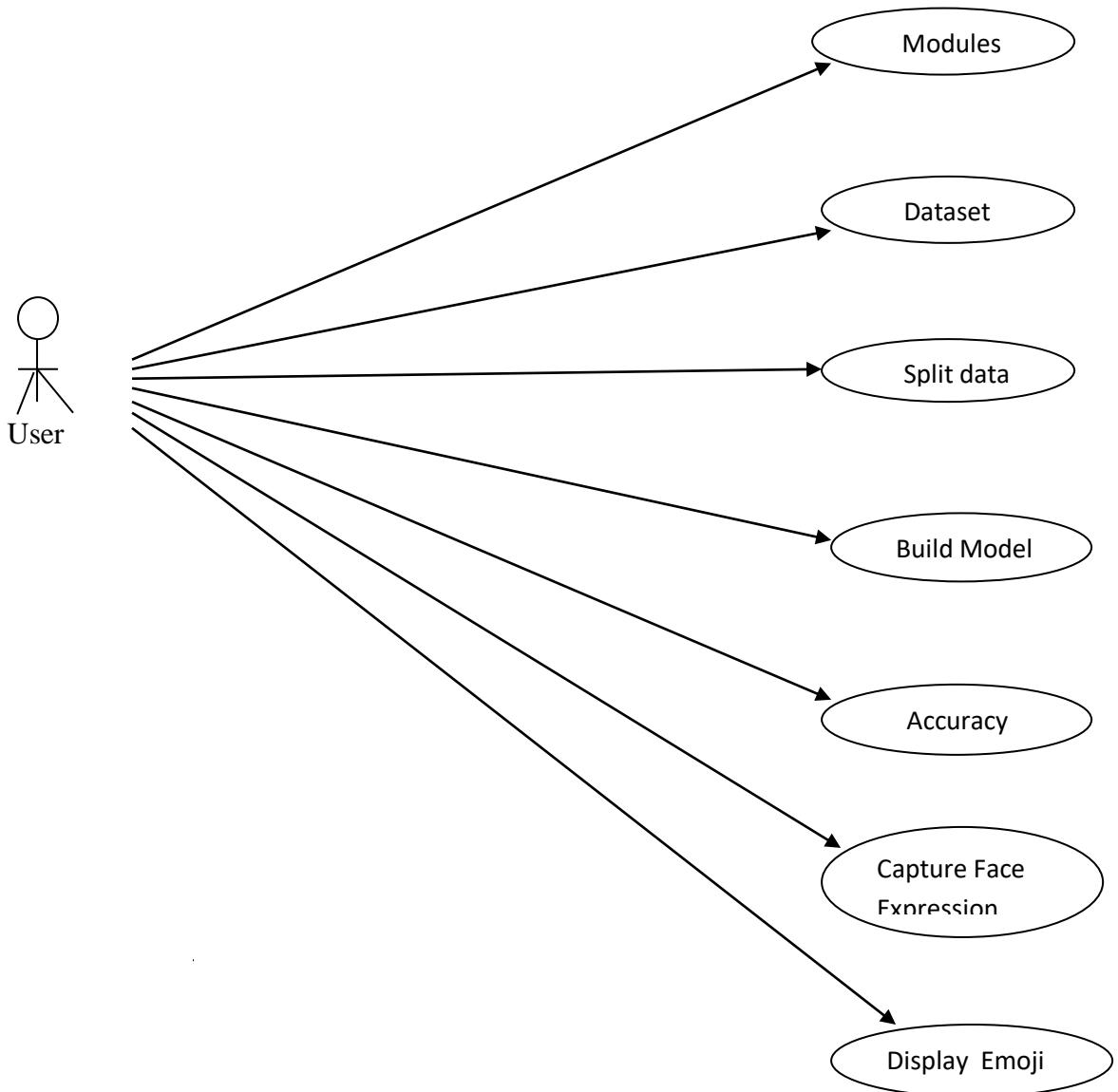


Fig 3.2.1: Use Case Diagram

3.2.2 Sequence Diagram

Sequence Diagrams represent the objects participating the interaction horizontally and time vertically. It model high-level interaction between active objects in a system. The sequence diagram is organized according to the time. The elements that are involved in interaction is shown by horizontal axis. Time proceedings down the page is represented by vertical axis.

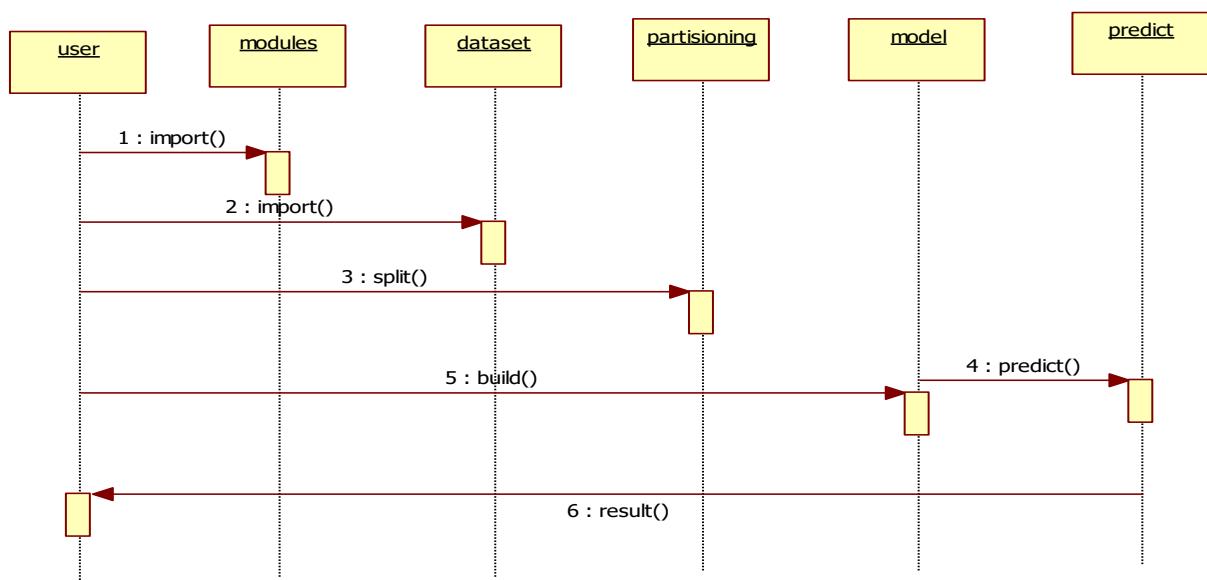


Fig 3.2.2: Sequence Diagram

3.2.3 Activity Diagram

Activity diagram is another important behavioural diagram in UML diagram to describe dynamic aspects of the system. These diagrams describe how activities are coordinated to provide a service which can be at different levels of abstraction. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

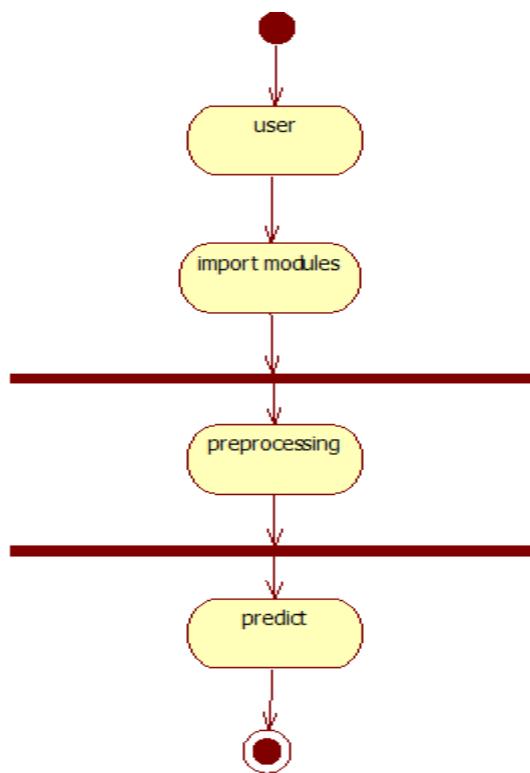


Fig 3.2.3: Activity Diagram

3.2.4 Collaboration Diagram

A Collaboration diagram resembles a flow chart that portrays the functionality, roles and behaviour of individual objects as well as the overall operation of the system in real time. It is also known as communication diagram. In these diagrams the relation between these objects shown as lines connecting the rectangles. The message between objects are shown as arrows connecting the relevant rectangles along with labels that defines the message sequencing.

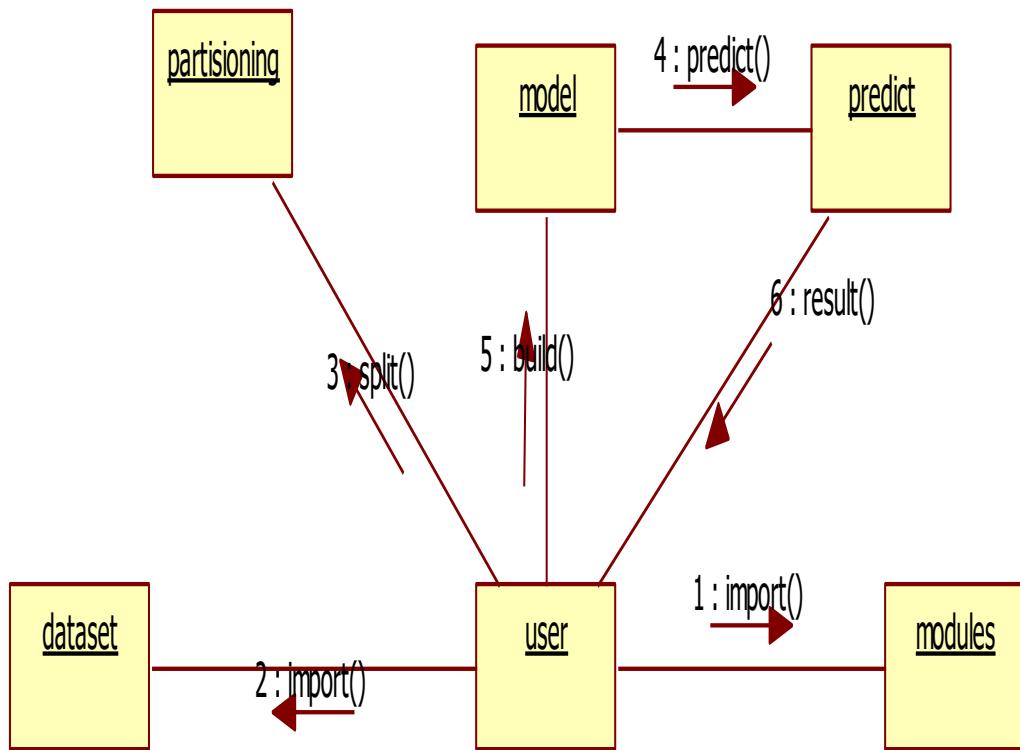


Fig 3.2.4:Collabration Diagram

3.2.5 Class Diagram

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's **classes**, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams provides a basic notation for other structure diagrams prescribed by uml.

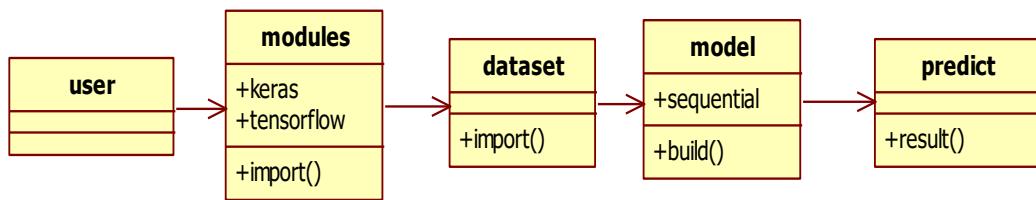
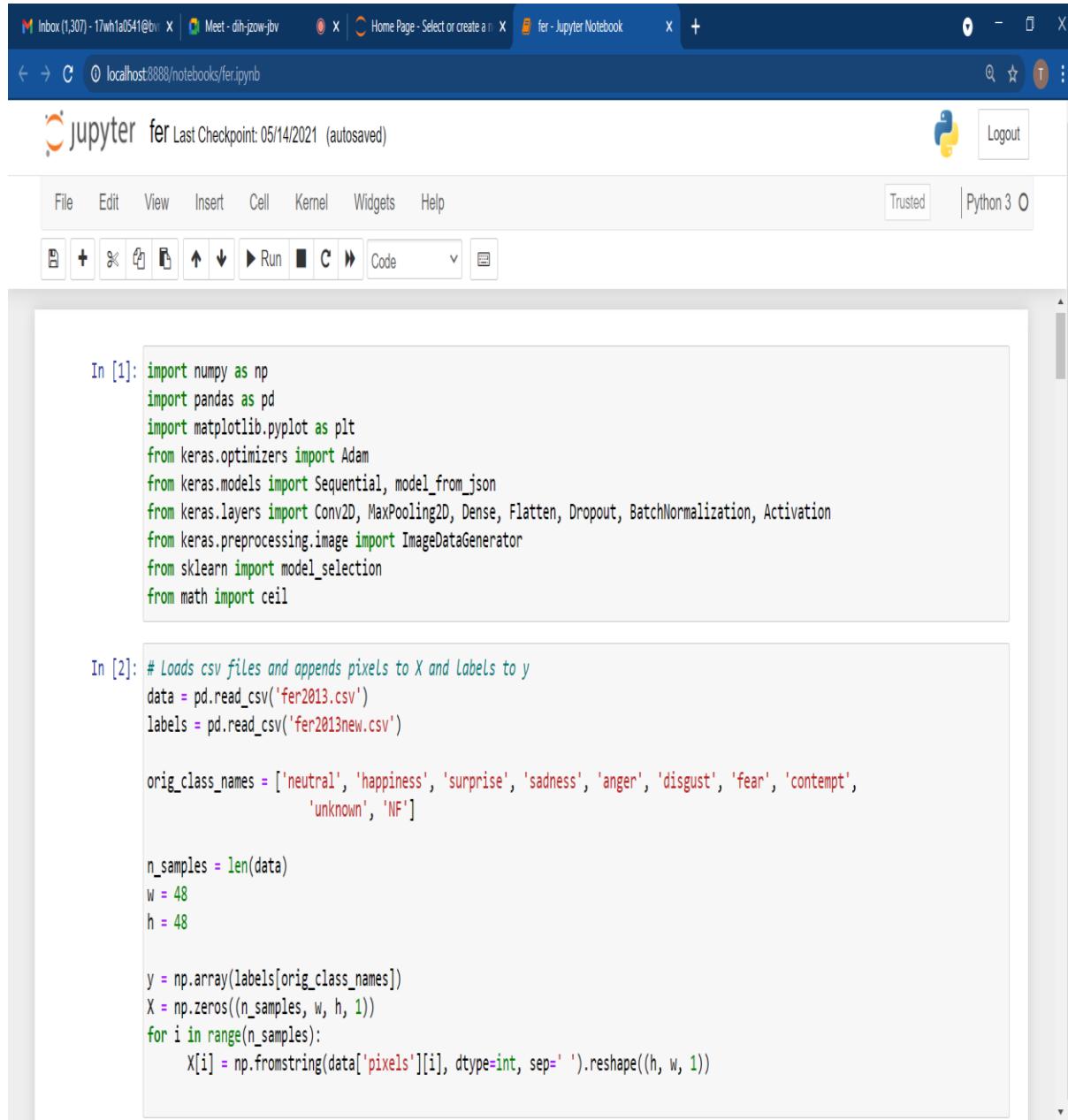


Fig 3.2.5: Class Diagram

4. IMPLEMENTATION

4.1 Data Preparation

In this phase initially the dataset is read which is in csv file format. The dataset contains labelled and unlabelled images. Removed all the unlabelled images. Next reshaped all the existing labelled images into same height and same width.



The screenshot shows a Jupyter Notebook interface with the following details:

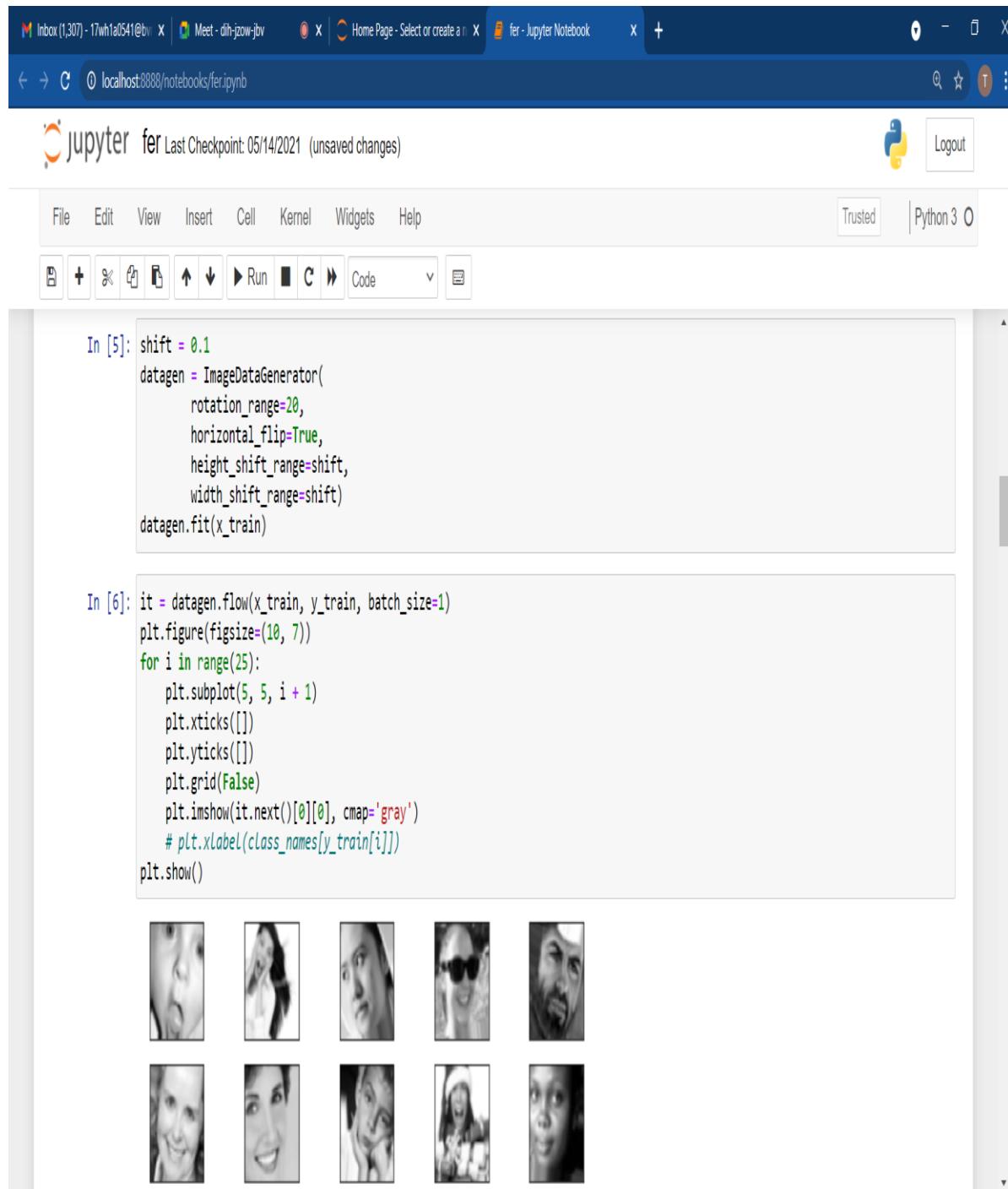
- Header:** Shows tabs for 'Inbox (1,307)', 'Meet - dih-zow-jbv', 'Home Page - Select or create a n...', and 'fer - Jupyter Notebook'. Below the tabs is a URL bar with 'localhost:8888/notebooks/fer.ipynb'.
- Title Bar:** Displays 'jupyter fer Last Checkpoint: 05/14/2021 (autosaved)' and a 'Logout' button.
- Toolbar:** Includes buttons for File, Edit, View, Insert, Cell, Kernel, Widgets, Help, Trusted, and Python 3 O.
- Code Cells:** Two code cells are visible:
 - In [1]:** Imports various Python libraries including numpy, pandas, matplotlib.pyplot, keras.optimizers, keras.models, keras.layers, keras.preprocessing.image, sklearn.model_selection, and math.
 - In [2]:** Loads CSV files, reads data and labels, defines original class names, sets dimensions (n_samples=48, w=48, h=48), and reshapes the data into a 3D array X.

Fig 4.1.1: Importing Libraries and Reading the Dataset

The screenshot shows a Jupyter Notebook interface with two code cells. The top cell (In [3]) contains code for processing a mask and normalizing image vectors:In [3]:
y_mask = y.argmax(axis=1)
mask = y_mask < orig_class_names.index('unknown')
X = X[mask]
y = y[mask]
y = y[:, :-2] * 0.1
y[:, 0] += y[:, 7]
y = y[:, :7]
Normalizing image vectors
X = X / 255.0The bottom cell (In [4]) contains code for splitting the data into training, testing, and validation sets, followed by a print statement showing the shapes of the resulting tensors:In [4]:
test_size = ceil(len(X) * 0.1)
Splitting the Data
x_train, x_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=test_size, random_state=42)
x_train, x_val, y_train, y_val = model_selection.train_test_split(x_train, y_train, test_size=test_size,
random_state=42)
print("X_train shape: " + str(x_train.shape))
print("Y_train shape: " + str(y_train.shape))
print("X_test shape: " + str(x_test.shape))
print("Y_test shape: " + str(y_test.shape))
print("X_val shape: " + str(x_val.shape))
print("Y_val shape: " + str(y_val.shape))

X_train shape: (28390, 48, 48, 1)
Y_train shape: (28390, 7)
X_test shape: (3549, 48, 48, 1)
Y_test shape: (3549, 7)
X_val shape: (3549, 48, 48, 1)
Y_val shape: (3549, 7)

Fig 4.1.2:Splitting the data



The screenshot shows a Jupyter Notebook interface with two code cells and their corresponding output images.

In [5]:

```
shift = 0.1
datagen = ImageDataGenerator(
    rotation_range=20,
    horizontal_flip=True,
    height_shift_range=shift,
    width_shift_range=shift)
datagen.fit(x_train)
```

In [6]:

```
it = datagen.flow(x_train, y_train, batch_size=1)
plt.figure(figsize=(10, 7))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(it.next()[0][0], cmap='gray')
    # plt.xlabel(class_names[y_train[i]])
plt.show()
```

The output of In [6] displays 25 generated images arranged in a 5x5 grid. The images show various faces with applied data augmentation effects such as rotations, horizontal flips, and shifts.

Fig 4.1.3: Data Augmentation

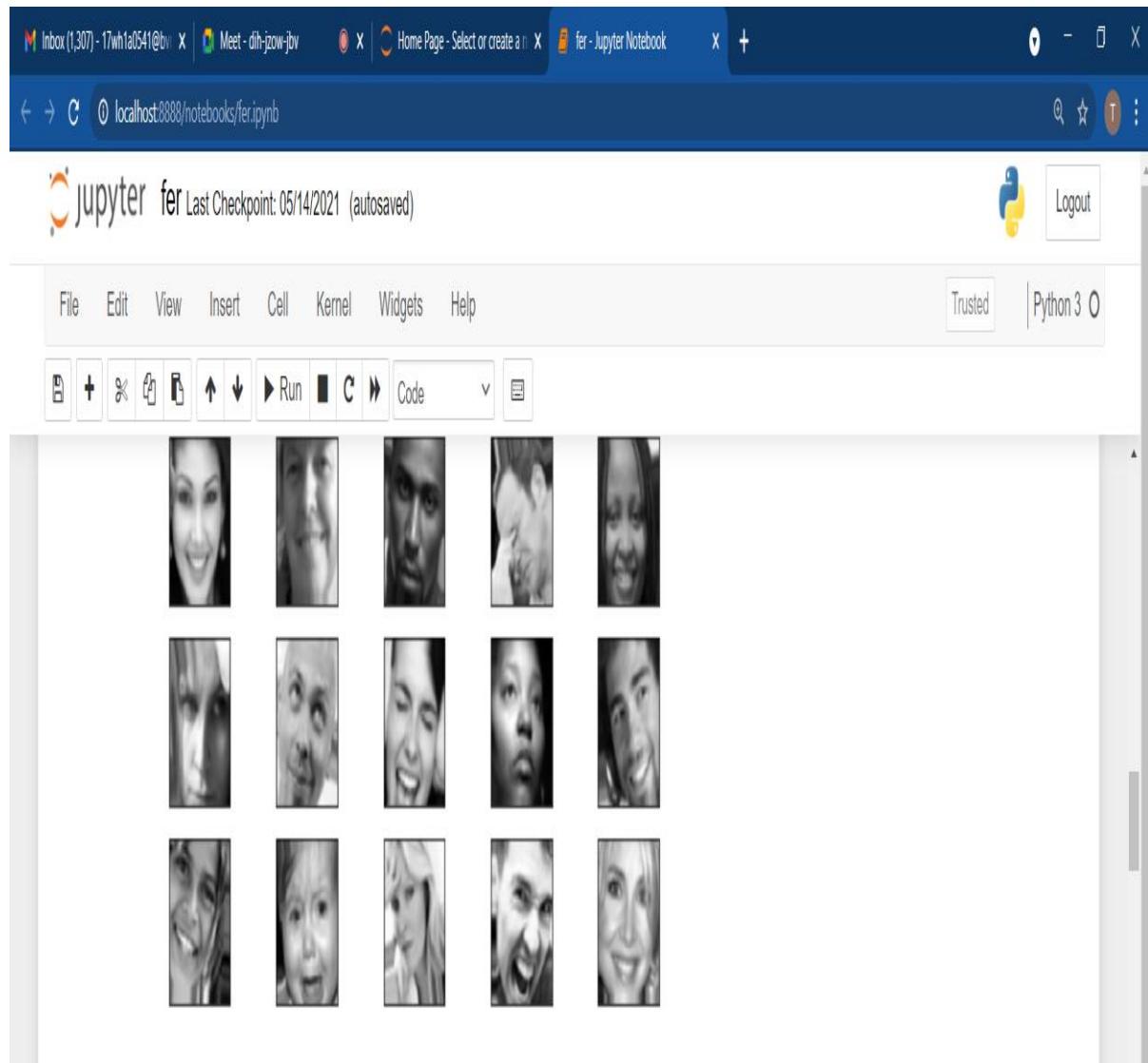
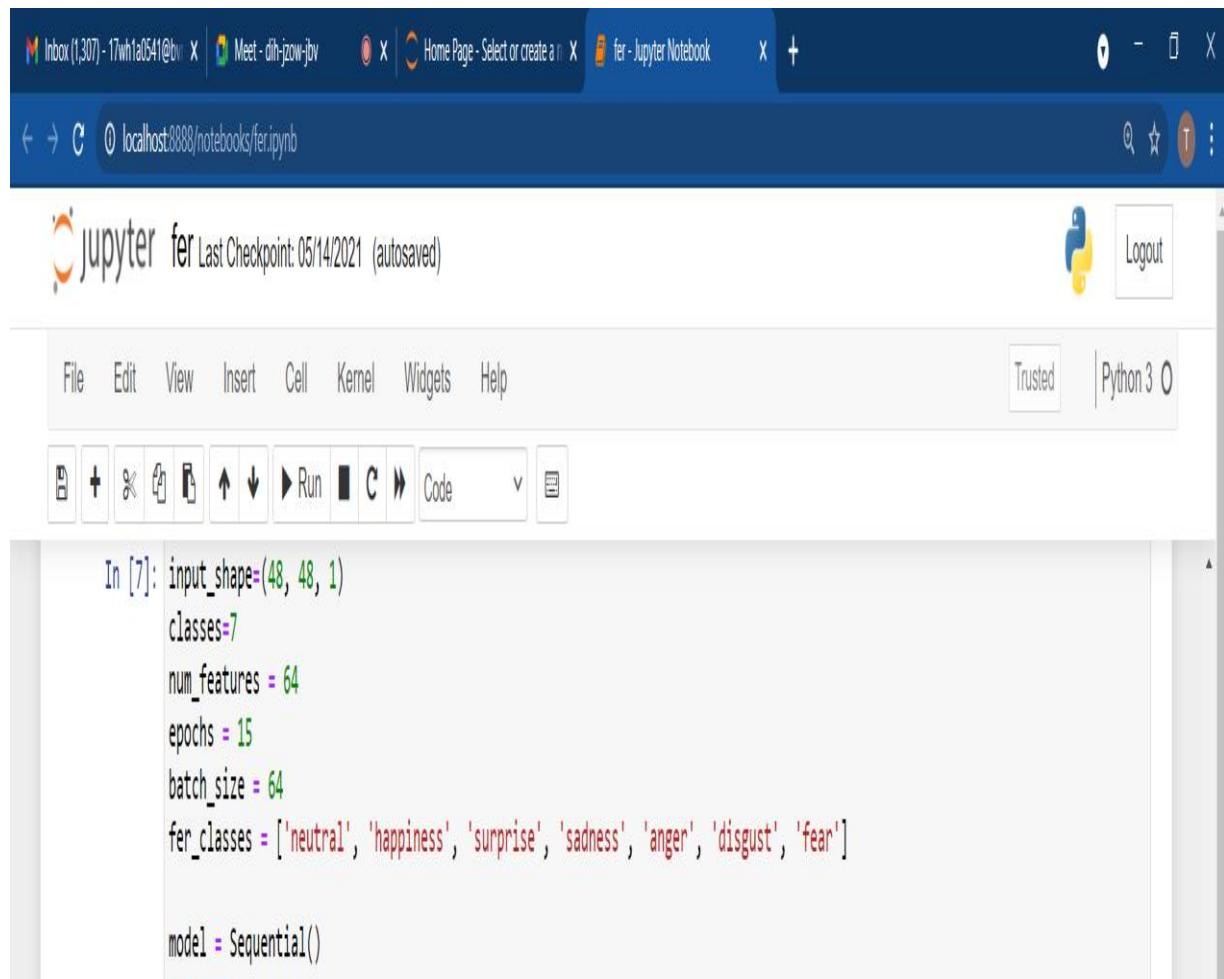


Fig 4.1.4: Displaying Reshaped Images

4.2 Model Creation

This is the second phase in this project. In this phase sequential convolutional model is created.



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Shows multiple browser tabs: 'Inbox (1,307) - 17wh1a0541@bx...', 'Meet - dln-izow-jbv', 'Home Page - Select or create a...', and 'fer - Jupyter Notebook'. The 'fer - Jupyter Notebook' tab is active.
- Breadcrumbs:** Shows the path: 'localhost:8888/notebooks/fer.ipynb'.
- User Information:** Includes a profile icon and 'Logout' button.
- Toolbar:** Includes standard Jupyter Notebook menu items: File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 Trusted status indicator.
- Code Editor:** Displays the following Python code in cell In [7]:

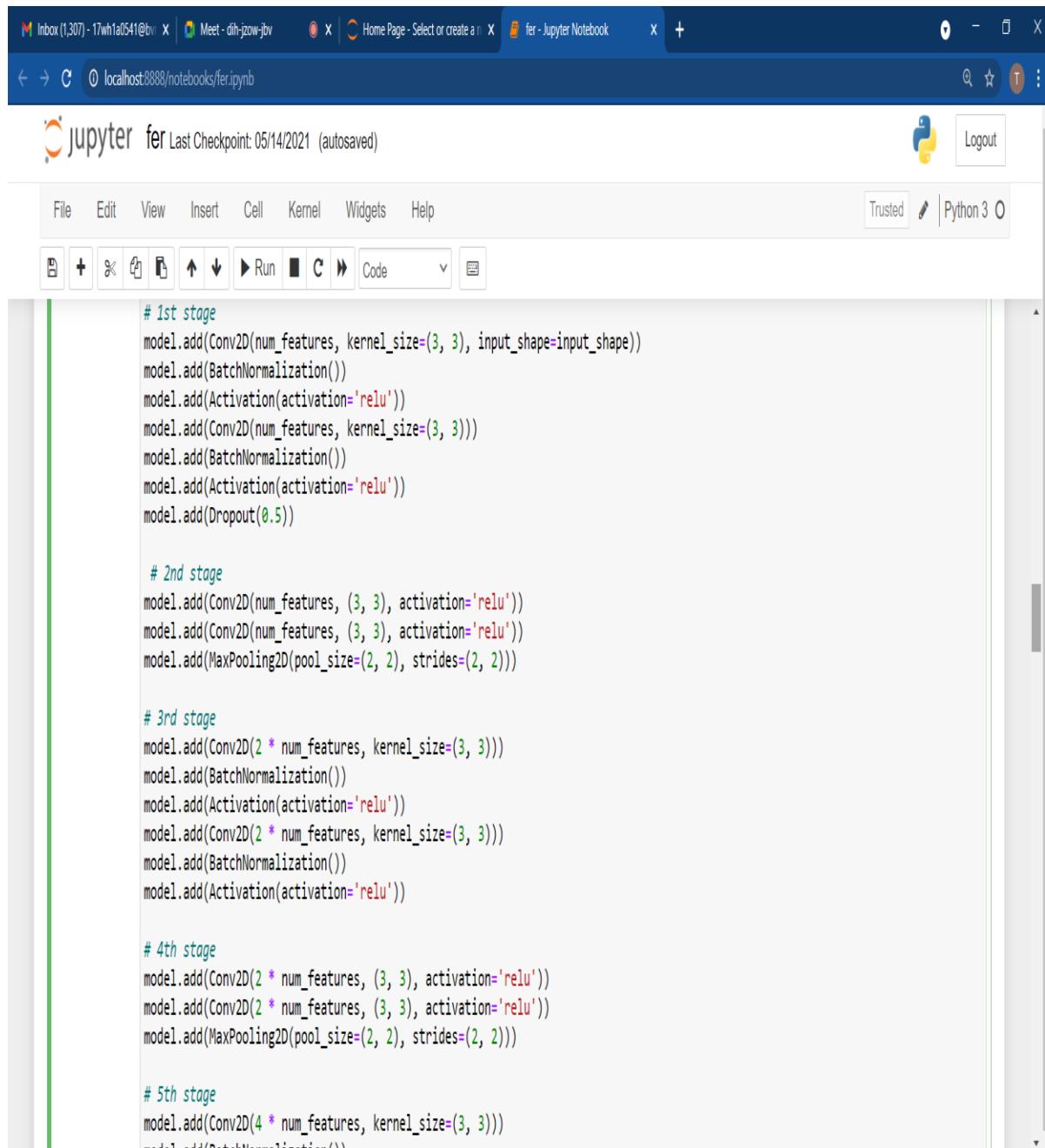
```
In [7]: input_shape=(48, 48, 1)
classes=7
num_features = 64
epochs = 15
batch_size = 64
fer_classes = ['neutral', 'happiness', 'surprise', 'sadness', 'anger', 'disgust', 'fear']

model = Sequential()
```

Fig 4.2.1: Creating the model

4.3 Model Building

It is the third phase in this project. Model is trained in this phase. In neural network algorithm the model is trained in a layered format, In each layer the understanding of the model increases which results in betterment of the accuracy.



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Shows multiple tabs including "Inbox (1,307)", "Meet - dih-jzow-jbv", "Home Page - Select or create a n...", and "fer - Jupyter Notebook".
- Title Bar:** Displays the URL "localhost:8888/notebooks/fer.ipynb" and the title "jupyter fer Last Checkpoint: 05/14/2021 (autosaved)".
- Toolbar:** Includes standard Jupyter Notebook icons for file operations, cell execution, and help.
- Menu Bar:** Offers options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help.
- Code Cell:** Contains the following Python code for building a neural network model:

```
# 1st stage
model.add(Conv2D(num_features, kernel_size=(3, 3), input_shape=input_shape))
model.add(BatchNormalization())
model.add(Activation(activation='relu'))
model.add(Conv2D(num_features, kernel_size=(3, 3)))
model.add(BatchNormalization())
model.add(Activation(activation='relu'))
model.add(Dropout(0.5))

# 2nd stage
model.add(Conv2D(num_features, (3, 3), activation='relu'))
model.add(Conv2D(num_features, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

# 3rd stage
model.add(Conv2D(2 * num_features, kernel_size=(3, 3)))
model.add(BatchNormalization())
model.add(Activation(activation='relu'))
model.add(Conv2D(2 * num_features, kernel_size=(3, 3)))
model.add(BatchNormalization())
model.add(Activation(activation='relu'))

# 4th stage
model.add(Conv2D(2 * num_features, (3, 3), activation='relu'))
model.add(Conv2D(2 * num_features, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

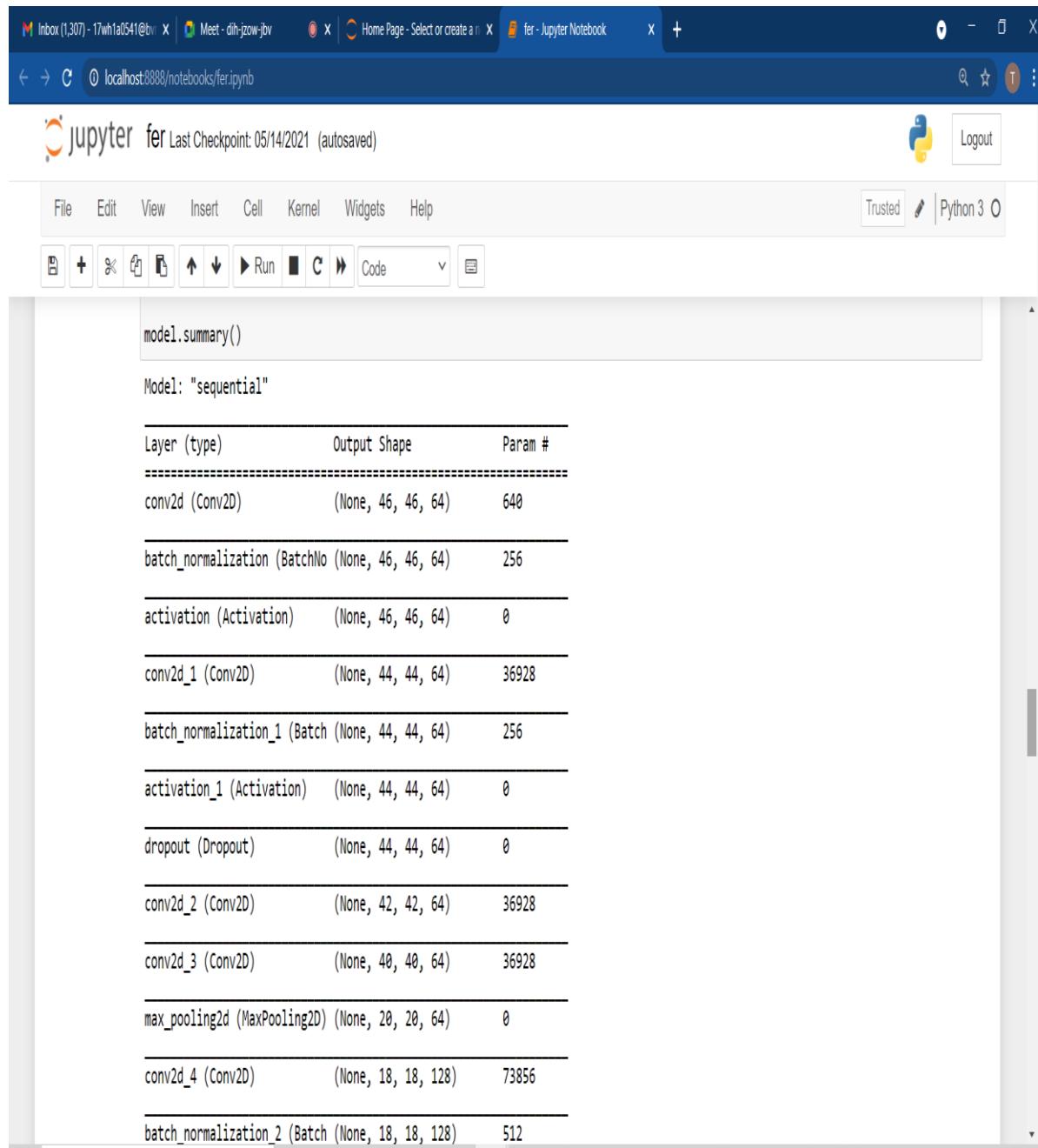
# 5th stage
model.add(Conv2D(4 * num_features, kernel_size=(3, 3)))
model.add(BatchNormalization())
```

Fig 4.3.1: Building the model

The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Shows the URL "localhost:8888/notebooks/fer.ipynb", the Jupyter logo, and a "Logout" button.
- Toolbar:** Includes standard Jupyter notebook icons for file operations, cell execution, and help.
- Code Cell:** Contains Python code for defining a neural network model. The code includes layers like Conv2D, BatchNormalization, Activation (relu), and Dense layers, followed by a Flatten layer and two fully connected layers of 1024 units each.
- Output Cell:** Labeled "In [8]". It shows the command "# Training model from scratch" and the output of `model.summary()`. The output displays the model's architecture as "sequential" with two layers: "conv2d" (Conv2D) and "batch_normalization" (BatchNormal).

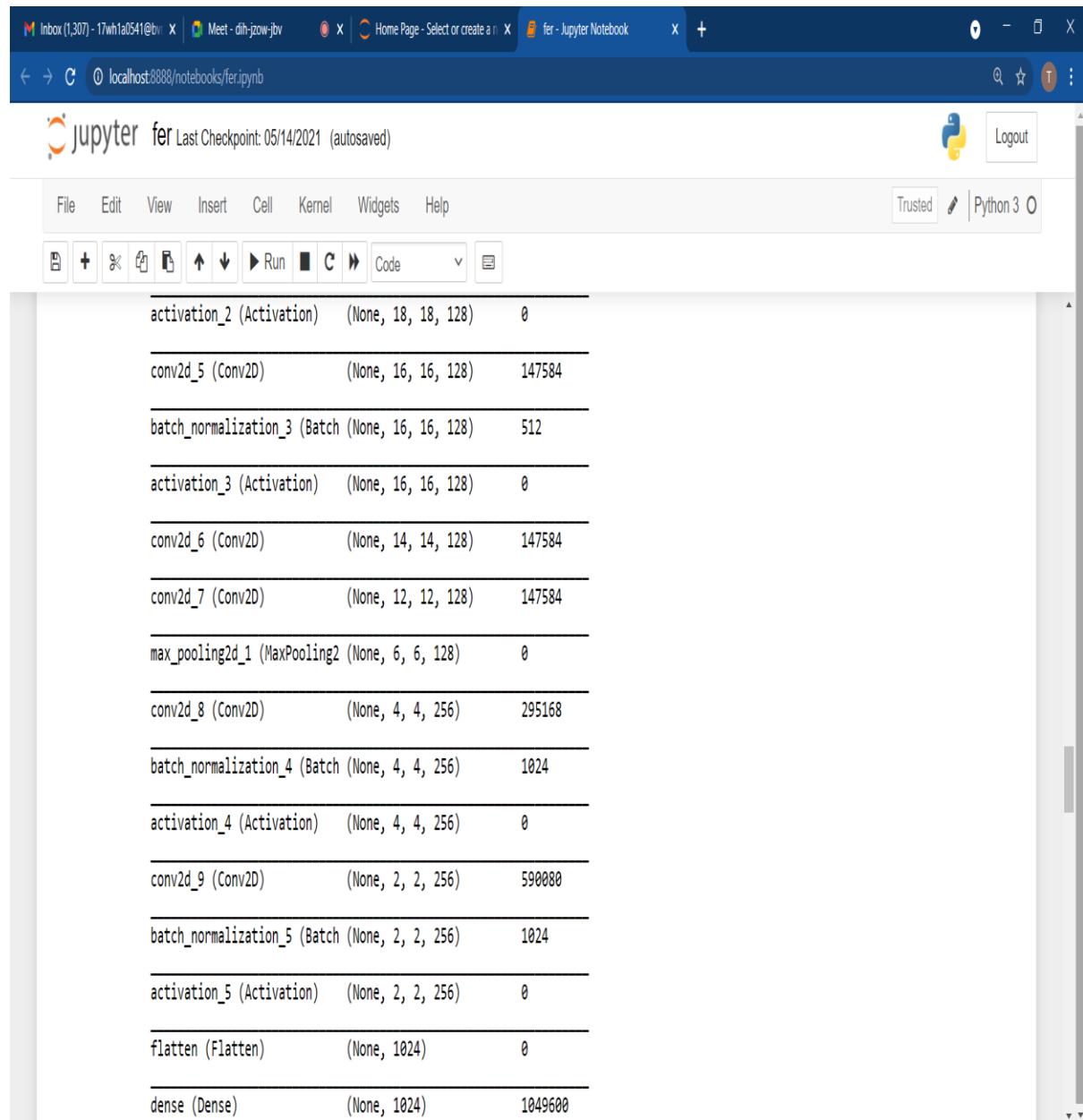
Fig 4.3.2: Flattening the model



The screenshot shows a Jupyter Notebook interface with a single code cell containing the command `model.summary()`. The output of this command is a detailed summary of the model's architecture, displayed as a table:

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 46, 46, 64)	640
batch_normalization (BatchNormal)	(None, 46, 46, 64)	256
activation (Activation)	(None, 46, 46, 64)	0
conv2d_1 (Conv2D)	(None, 44, 44, 64)	36928
batch_normalization_1 (BatchNormal)	(None, 44, 44, 64)	256
activation_1 (Activation)	(None, 44, 44, 64)	0
dropout (Dropout)	(None, 44, 44, 64)	0
conv2d_2 (Conv2D)	(None, 42, 42, 64)	36928
conv2d_3 (Conv2D)	(None, 40, 40, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 20, 20, 64)	0
conv2d_4 (Conv2D)	(None, 18, 18, 128)	73856
batch_normalization_2 (BatchNormal)	(None, 18, 18, 128)	512

Fig 4.3.3 : Model Summary



The screenshot shows a Jupyter Notebook interface with a table displaying the architecture of a neural network. The table lists 15 layers, each with its name, type, shape, and number of parameters.

Layer	Type	Shape	Parameters
activation_2	(Activation)	(None, 18, 18, 128)	0
conv2d_5	(Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_3	(Batch Normalization)	(None, 16, 16, 128)	512
activation_3	(Activation)	(None, 16, 16, 128)	0
conv2d_6	(Conv2D)	(None, 14, 14, 128)	147584
conv2d_7	(Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_1	(MaxPooling2D)	(None, 6, 6, 128)	0
conv2d_8	(Conv2D)	(None, 4, 4, 256)	295168
batch_normalization_4	(Batch Normalization)	(None, 4, 4, 256)	1024
activation_4	(Activation)	(None, 4, 4, 256)	0
conv2d_9	(Conv2D)	(None, 2, 2, 256)	590080
batch_normalization_5	(Batch Normalization)	(None, 2, 2, 256)	1024
activation_5	(Activation)	(None, 2, 2, 256)	0
flatten	(Flatten)	(None, 1024)	0
dense	(Dense)	(None, 1024)	1049600

Fig 4.3.4: Describing the model layers

The screenshot shows a Jupyter Notebook interface with the following content:

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 Logout
```

Cell toolbar icons: File, New, Open, Save, Run, Cell, Kernel, Help, Code, etc.

Model summary output:

dropout_1 (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 1024)	1049600
dropout_2 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 7)	7175

=====

```
Total params: 3,623,239
Trainable params: 3,621,447
Non-trainable params: 1,792
```

In [9]:

```
model.compile(optimizer=Adam(lr=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

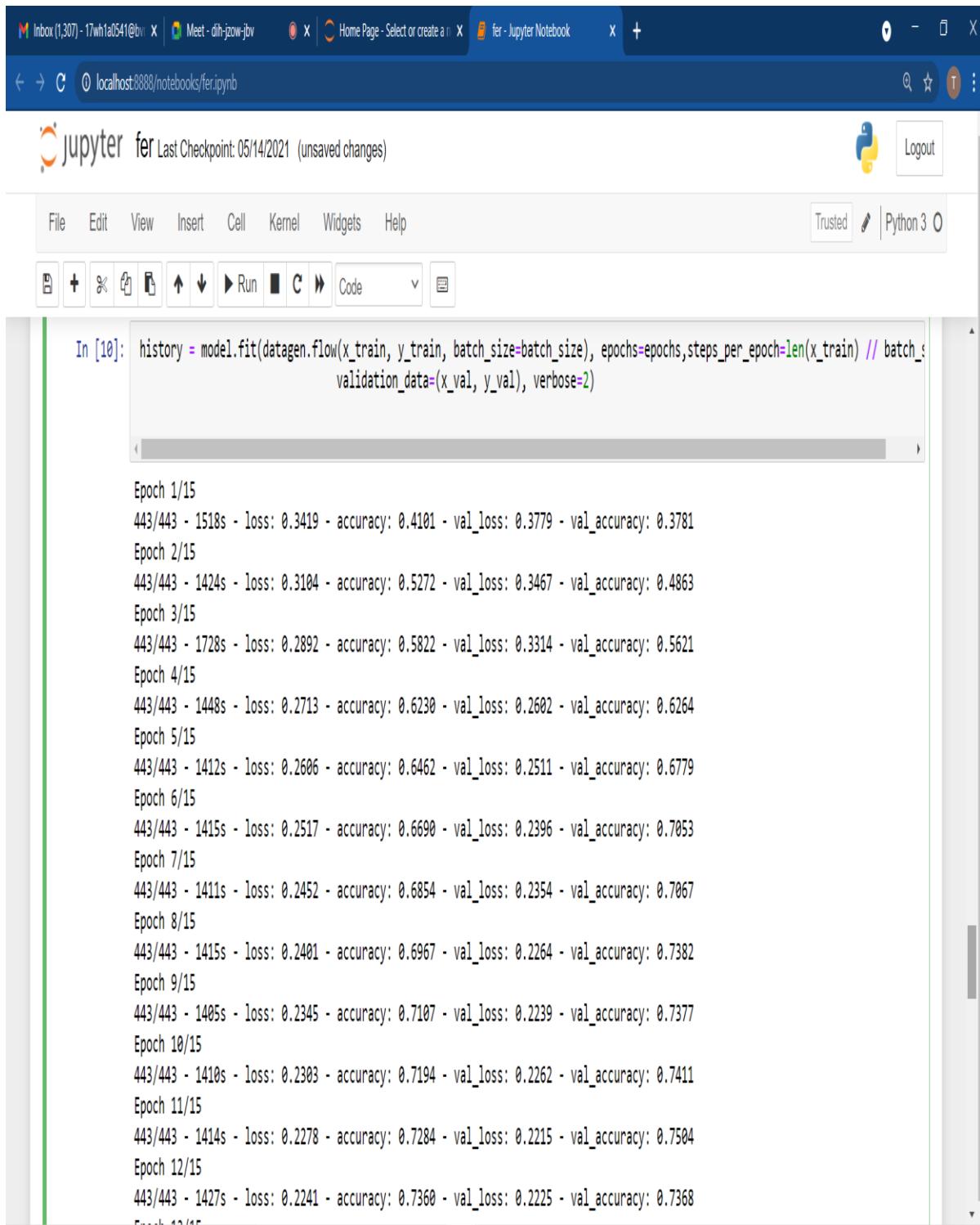
In [10]:

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size), epochs=epochs, steps_per_epoch=len(x_train) // batch_size, validation_data=(x_val, y_val), verbose=2)
```

Output logs (Epochs 1-4):

```
Epoch 1/15
443/443 - 1518s - loss: 0.3419 - accuracy: 0.4101 - val_loss: 0.3779 - val_accuracy: 0.3781
Epoch 2/15
443/443 - 1424s - loss: 0.3104 - accuracy: 0.5272 - val_loss: 0.3467 - val_accuracy: 0.4863
Epoch 3/15
443/443 - 1728s - loss: 0.2892 - accuracy: 0.5822 - val_loss: 0.3314 - val_accuracy: 0.5621
Epoch 4/15
```

Fig 4.3.5: Model Compilation



The screenshot shows a Jupyter Notebook interface running in a browser window. The title bar indicates the URL is `localhost:8888/notebooks/feripynb`. The notebook header shows "jupyter fer Last Checkpoint: 05/14/2021 (unsaved changes)". The toolbar includes standard options like File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a Python 3 kernel selector. Below the toolbar is a toolbar with icons for file operations like Open, Save, and Run, along with a Code dropdown menu.

In [10]:

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=batch_size), epochs=epochs, steps_per_epoch=len(x_train) // batch_size, validation_data=(x_val, y_val), verbose=2)
```

Epoch 1/15
443/443 - 1518s - loss: 0.3419 - accuracy: 0.4101 - val_loss: 0.3779 - val_accuracy: 0.3781
Epoch 2/15
443/443 - 1424s - loss: 0.3104 - accuracy: 0.5272 - val_loss: 0.3467 - val_accuracy: 0.4863
Epoch 3/15
443/443 - 1728s - loss: 0.2892 - accuracy: 0.5822 - val_loss: 0.3314 - val_accuracy: 0.5621
Epoch 4/15
443/443 - 1448s - loss: 0.2713 - accuracy: 0.6230 - val_loss: 0.2602 - val_accuracy: 0.6264
Epoch 5/15
443/443 - 1412s - loss: 0.2606 - accuracy: 0.6462 - val_loss: 0.2511 - val_accuracy: 0.6779
Epoch 6/15
443/443 - 1415s - loss: 0.2517 - accuracy: 0.6690 - val_loss: 0.2396 - val_accuracy: 0.7053
Epoch 7/15
443/443 - 1411s - loss: 0.2452 - accuracy: 0.6854 - val_loss: 0.2354 - val_accuracy: 0.7067
Epoch 8/15
443/443 - 1415s - loss: 0.2401 - accuracy: 0.6967 - val_loss: 0.2264 - val_accuracy: 0.7382
Epoch 9/15
443/443 - 1405s - loss: 0.2345 - accuracy: 0.7107 - val_loss: 0.2239 - val_accuracy: 0.7377
Epoch 10/15
443/443 - 1410s - loss: 0.2303 - accuracy: 0.7194 - val_loss: 0.2262 - val_accuracy: 0.7411
Epoch 11/15
443/443 - 1414s - loss: 0.2278 - accuracy: 0.7284 - val_loss: 0.2215 - val_accuracy: 0.7504
Epoch 12/15
443/443 - 1427s - loss: 0.2241 - accuracy: 0.7360 - val_loss: 0.2225 - val_accuracy: 0.7368

Fig 4.3.6: Model training

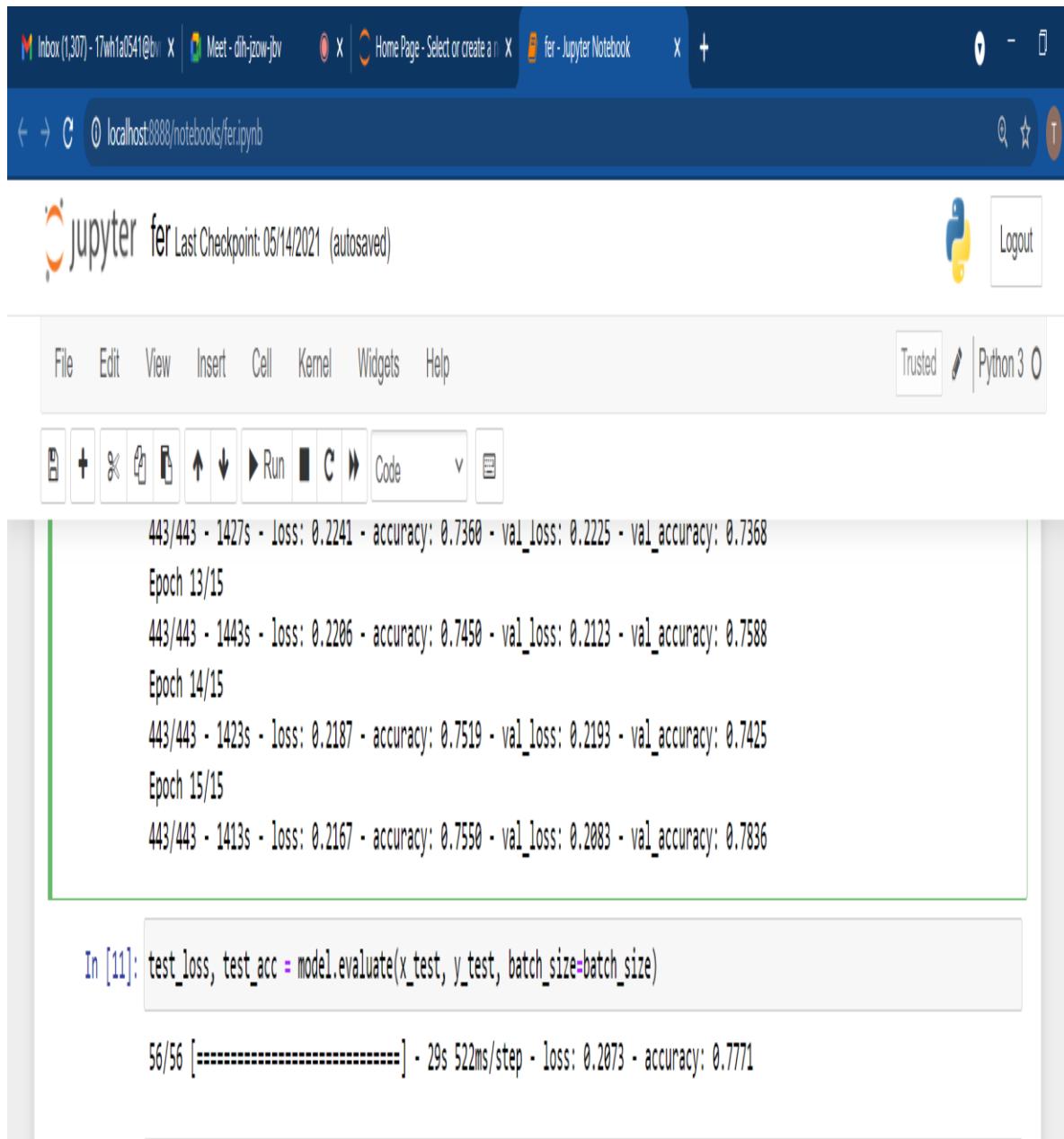
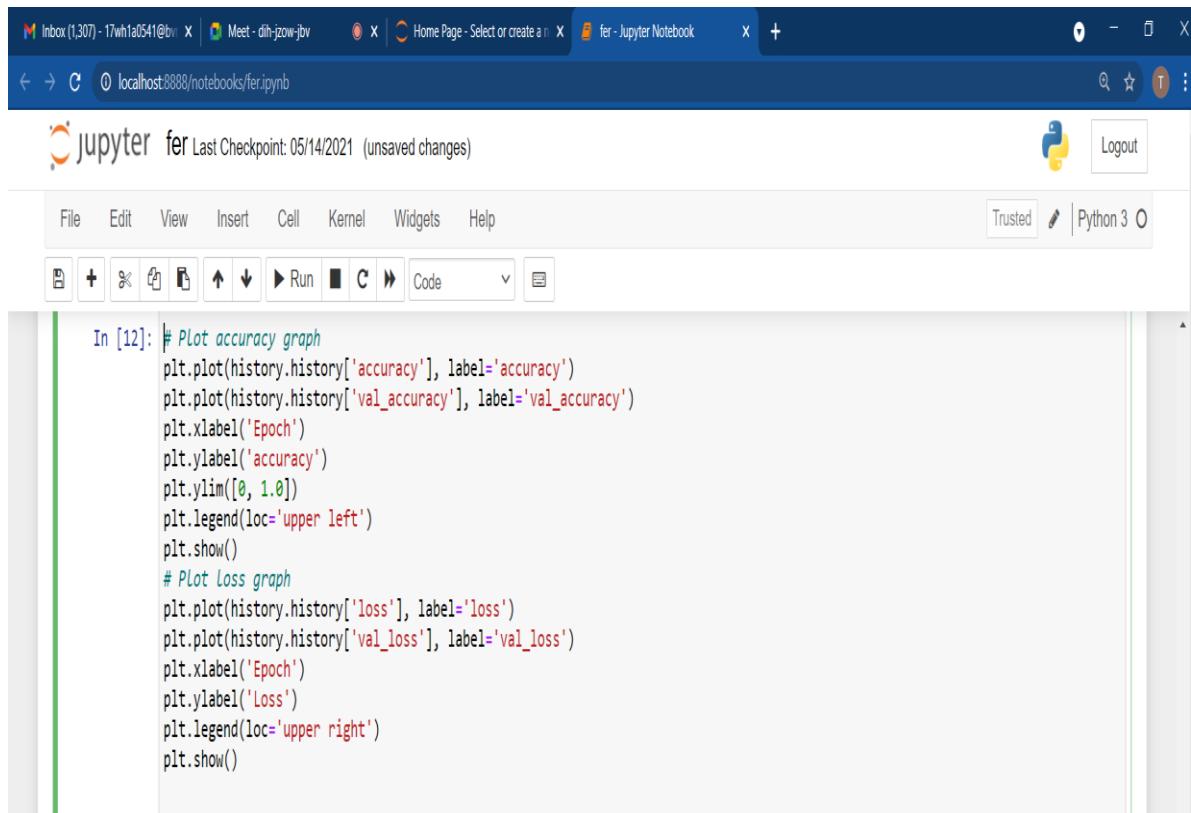


Fig 4.3.7: Model Accuracy



The screenshot shows a Jupyter Notebook interface with a Python 3 kernel. In cell In [12], the following code is displayed:

```
# Plot accuracy graph
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('accuracy')
plt.ylim([0, 1.0])
plt.legend(loc='upper left')
plt.show()

# Plot Loss graph
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.show()
```

Fig 4.3.8: Plotting Accuracy and Loss graph



Fig 4.3.9: Plot between accuracy and val_accuracy

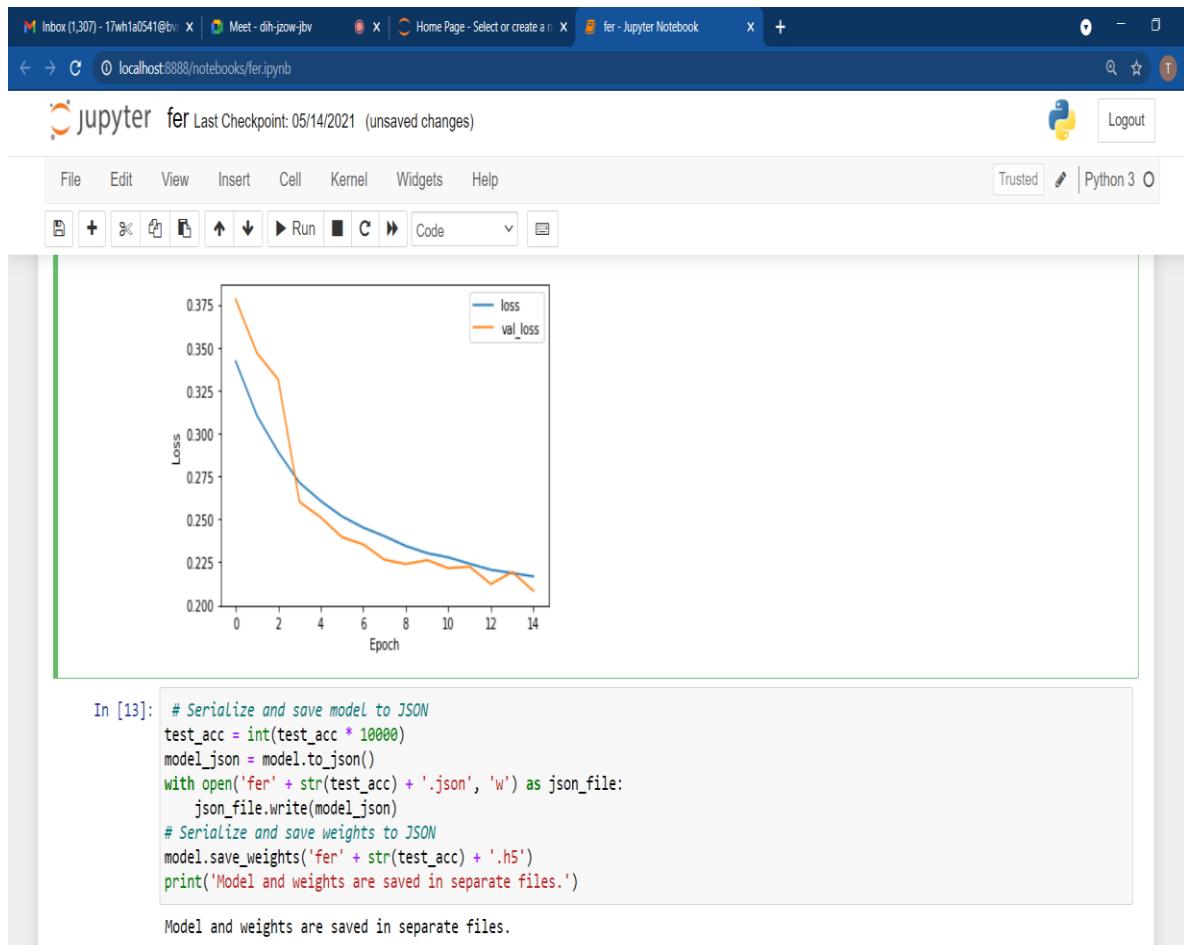
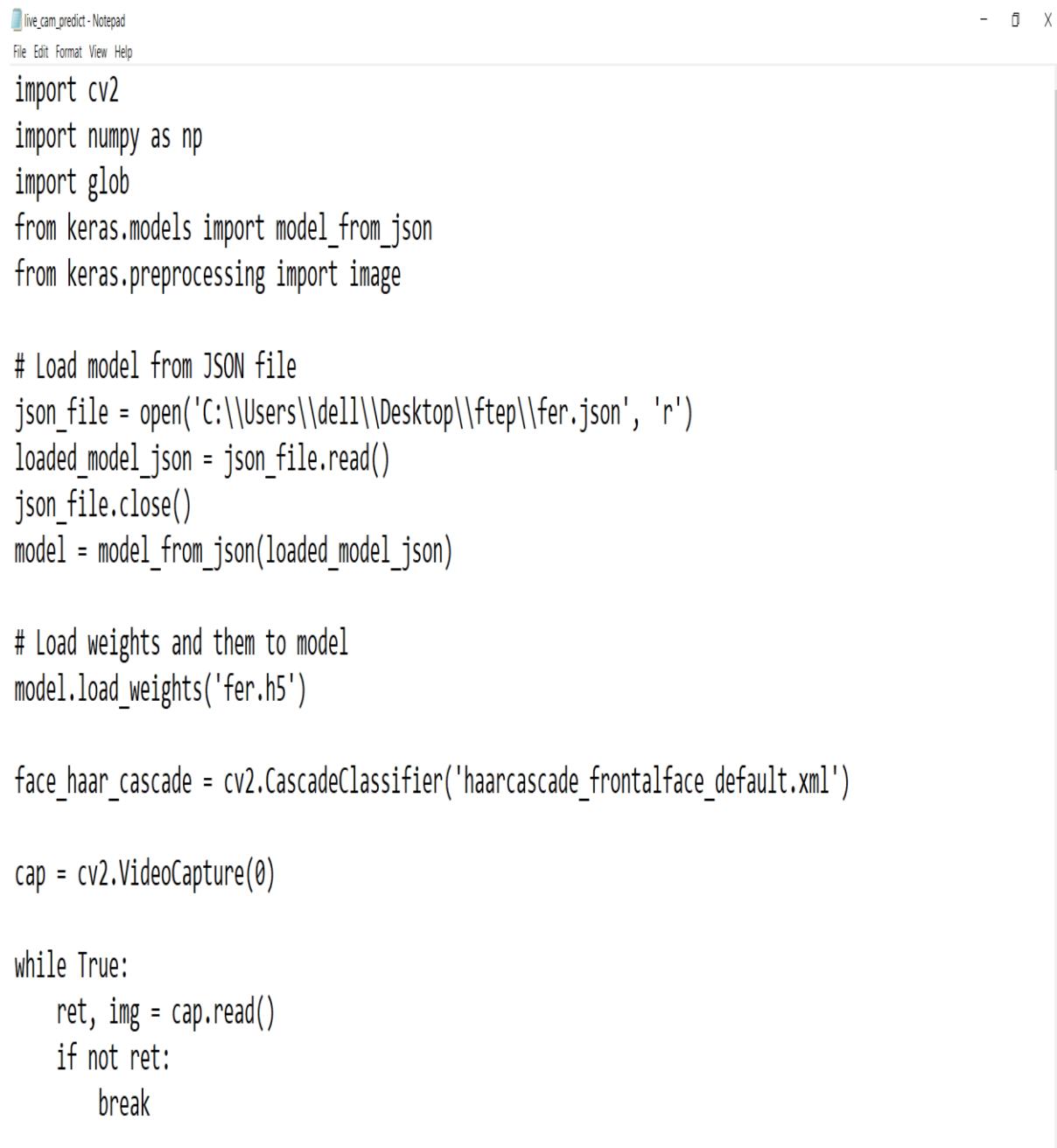


Fig 4.3.10: Plot between loss and val_loss

4.4 Image Capturing

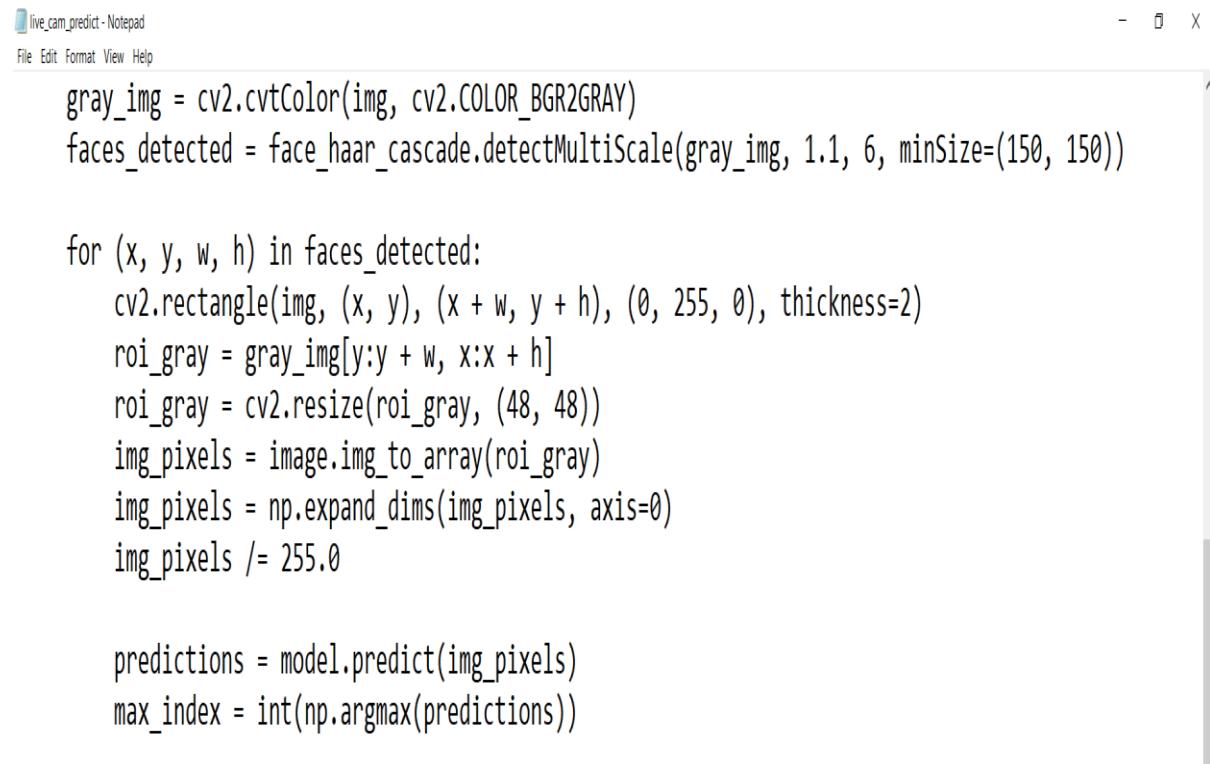
In this phase the image is captured from the live video and features of the face like eyes, mouth, eye brows are detected. This is achieved using Haar Cascade Classifier.



The screenshot shows a Notepad window titled "live_cam_predict - Notepad". The code is written in Python and performs the following steps:

- Imports cv2, numpy, glob, model_from_json, and image from keras.
- Loads a JSON file containing the model architecture from "fer.json".
- Closes the JSON file.
- Creates a Keras model from the loaded JSON.
- Loads weights into the model from "fer.h5".
- Creates a CascadeClassifier object for frontal face detection using "haarcascade_frontalface_default.xml".
- Opens a VideoCapture object for the camera (index 0).
- Enters a loop where it reads frames from the camera. If no frame is returned (ret is False), it breaks out of the loop.

Fig 4.4.1: Starting the Camera



The screenshot shows a Notepad window titled "live_cam_predict - Notepad". The menu bar includes "File", "Edit", "Format", "View", and "Help". The code in the editor is as follows:

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces_detected = face_haar_cascade.detectMultiScale(gray_img, 1.1, 6, minSize=(150, 150))

for (x, y, w, h) in faces_detected:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), thickness=2)
    roi_gray = gray_img[y:y + w, x:x + h]
    roi_gray = cv2.resize(roi_gray, (48, 48))
    img_pixels = image.img_to_array(roi_gray)
    img_pixels = np.expand_dims(img_pixels, axis=0)
    img_pixels /= 255.0

    predictions = model.predict(img_pixels)
    max_index = int(np.argmax(predictions))
```

Fig 4.4.2: Face detection

4.5 Facial Expression to Emoji

It is the last phase in this project. In this phase, the detected facial expression is converted to the corresponding emoji.



The screenshot shows a Notepad window titled "live_cam_predict - Notepad". The code is a Python script for facial emotion recognition. It starts by defining a list of emotions and a dictionary of emotion-to-image paths. It then initializes variables for prediction, reads the predicted emotion from the dictionary, and displays it on the image. The image is then resized and concatenated with another image. Finally, it shows the resulting image and waits for a key press to exit.

```
live_cam_predict - Notepad
File Edit Format View Help

emotions = ['neutral', 'happy', 'surprise', 'sadness', 'anger', 'disgust', 'fear']

EMOTIONS = {'anger':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\anger.png', 'disgust':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\disgust.png', 'fear':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\fear.png', 'happy':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\happy.png', 'neutral':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\neutral.png', 'sadness':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\sadness.png', 'surprise':'C:\\Users\\dell\\Desktop\\ftep\\emojis\\surprise.png'}

pr = emotions[max_index]
predicted_emotion = pr
pimg = EMOTIONS[predicted_emotion]
img1 = cv2.imread(pimg)
cv2.putText(img, predicted_emotion,(int(x), int(y)), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2)
resized_img = cv2.resize(img,(300, 300))
img2 = cv2.resize(img1,(300, 300))
Hori = np.concatenate((resized_img,img2),axis = 1)
cv2.imshow('Facial Emotion Recognition', Hori)

if cv2.waitKey(1) & 0xFF == ord('q'):
    break
cap.release()
cv2.destroyAllWindows()
```

Fig 4.5.1: Displaying the emojis

4.6 Test Cases

Test Case ID	Test Scenario	Expected Result	Actual Result	Pass/Fail
TC01	Check If Camera is On.	Camera must be On.	As Expected	Pass
TC02	Check If User Window is Open.	User Window must be Opened.	As Expected	Pass
TC03	Check If Camera is Capturing the Face.	Camera must Capture the Face.	As Expected	Pass
TC04	Check If Corresponding Emoji of Given Facial Expression is Displaying.	Corresponding Emoji of given Facial Expression must be Displayed.	As Expected	Pass

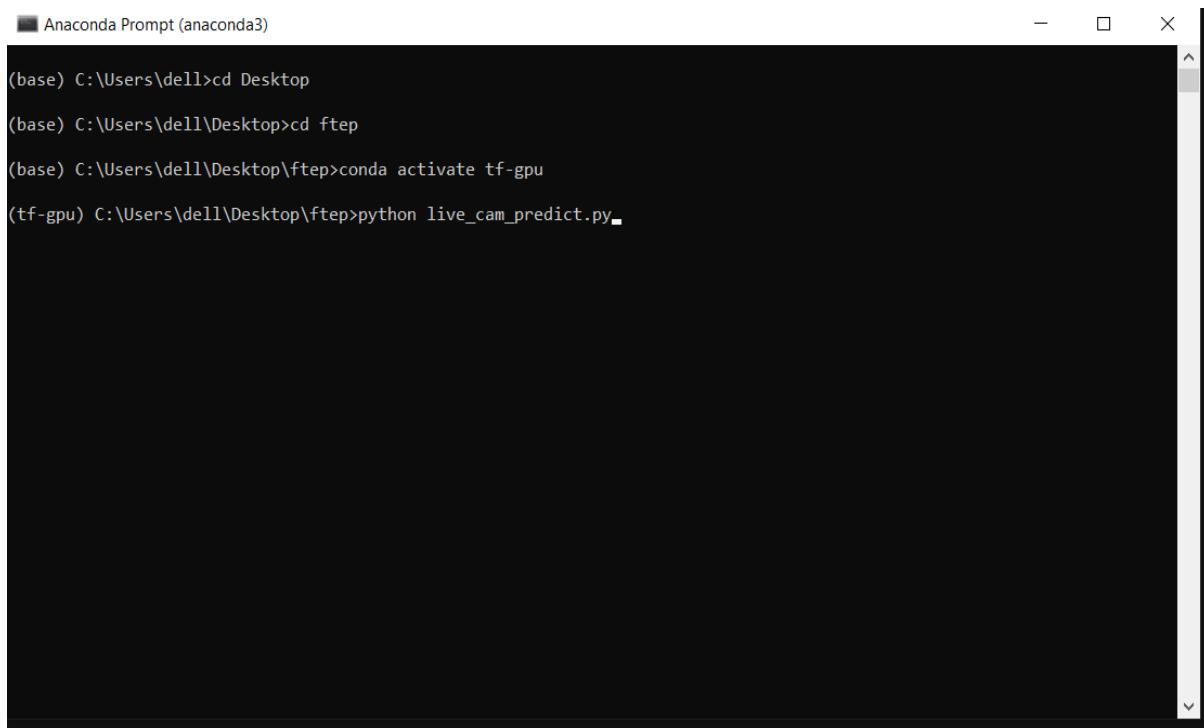
Fig 4.6.1: Test Cases

4.7 DATASET TRAINING SCREENSCHOTS

```
Epoch 1/15
443/443 - 1518s - loss: 0.3419 - accuracy: 0.4101 - val_loss: 0.3779 - val_accuracy: 0.3781
Epoch 2/15
443/443 - 1424s - loss: 0.3104 - accuracy: 0.5272 - val_loss: 0.3467 - val_accuracy: 0.4863
Epoch 3/15
443/443 - 1728s - loss: 0.2892 - accuracy: 0.5822 - val_loss: 0.3314 - val_accuracy: 0.5621
Epoch 4/15
443/443 - 1448s - loss: 0.2713 - accuracy: 0.6230 - val_loss: 0.2602 - val_accuracy: 0.6264
Epoch 5/15
443/443 - 1412s - loss: 0.2606 - accuracy: 0.6462 - val_loss: 0.2511 - val_accuracy: 0.6779
Epoch 6/15
443/443 - 1415s - loss: 0.2517 - accuracy: 0.6690 - val_loss: 0.2396 - val_accuracy: 0.7053
Epoch 7/15
443/443 - 1411s - loss: 0.2452 - accuracy: 0.6854 - val_loss: 0.2354 - val_accuracy: 0.7067
Epoch 8/15
443/443 - 1415s - loss: 0.2401 - accuracy: 0.6967 - val_loss: 0.2264 - val_accuracy: 0.7382
Epoch 9/15
443/443 - 1405s - loss: 0.2345 - accuracy: 0.7187 - val_loss: 0.2239 - val_accuracy: 0.7377
Epoch 10/15
443/443 - 1410s - loss: 0.2303 - accuracy: 0.7194 - val_loss: 0.2262 - val_accuracy: 0.7411
Epoch 11/15
443/443 - 1414s - loss: 0.2278 - accuracy: 0.7284 - val_loss: 0.2215 - val_accuracy: 0.7504
Epoch 12/15
443/443 - 1427s - loss: 0.2241 - accuracy: 0.7360 - val_loss: 0.2225 - val_accuracy: 0.7368
Epoch 13/15
443/443 - 1443s - loss: 0.2206 - accuracy: 0.7450 - val_loss: 0.2123 - val_accuracy: 0.7588
Epoch 14/15
443/443 - 1423s - loss: 0.2187 - accuracy: 0.7519 - val_loss: 0.2193 - val_accuracy: 0.7425
Epoch 15/15
443/443 - 1413s - loss: 0.2167 - accuracy: 0.7550 - val_loss: 0.2083 - val_accuracy: 0.7836
```

Fig 4.7.1: Training dataset

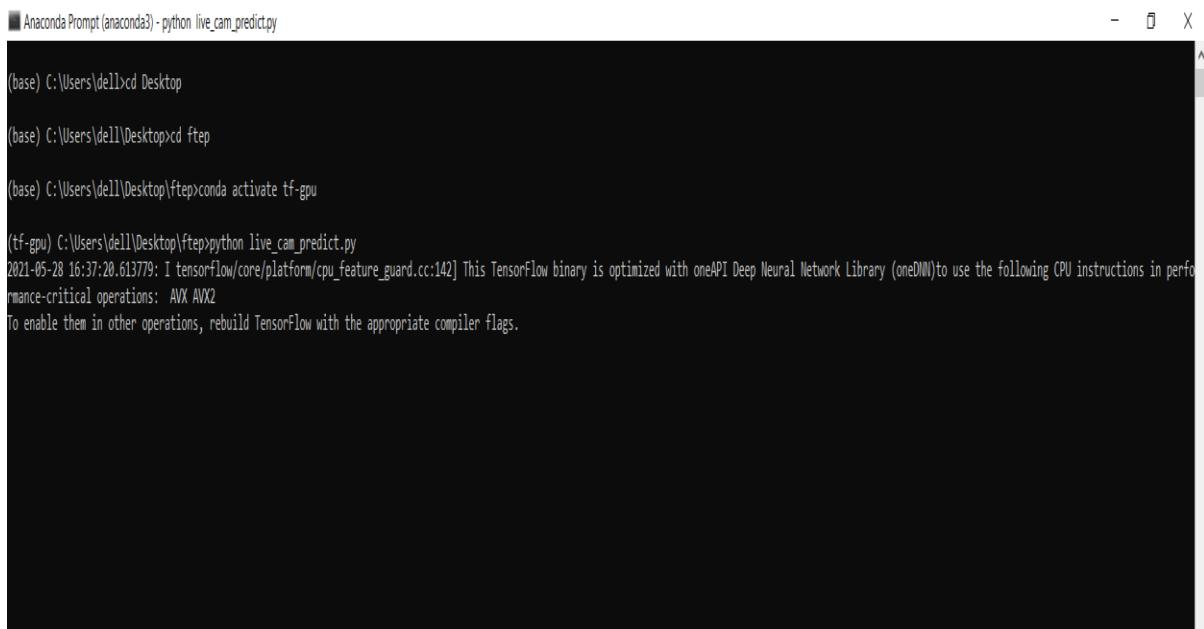
4.8 INPUT SCREENSHOTS



Anaconda Prompt (anaconda3)

```
(base) C:\Users\dell>cd Desktop
(base) C:\Users\dell\Desktop>cd ftep
(base) C:\Users\dell\Desktop\ftep>conda activate tf-gpu
(tf-gpu) C:\Users\dell\Desktop\ftep>python live_cam_predict.py
```

Fig 4.8.1: Command to run project



Anaconda Prompt (anaconda3) - python live_cam_predict.py

```
(base) C:\Users\dell>cd Desktop
(base) C:\Users\dell\Desktop>cd ftep
(base) C:\Users\dell\Desktop\ftep>conda activate tf-gpu
(tf-gpu) C:\Users\dell\Desktop\ftep>python live_cam_predict.py
2021-05-28 16:37:20.613779: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operations: AVX AVX2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

Fig 4.8.2: Camera Starting

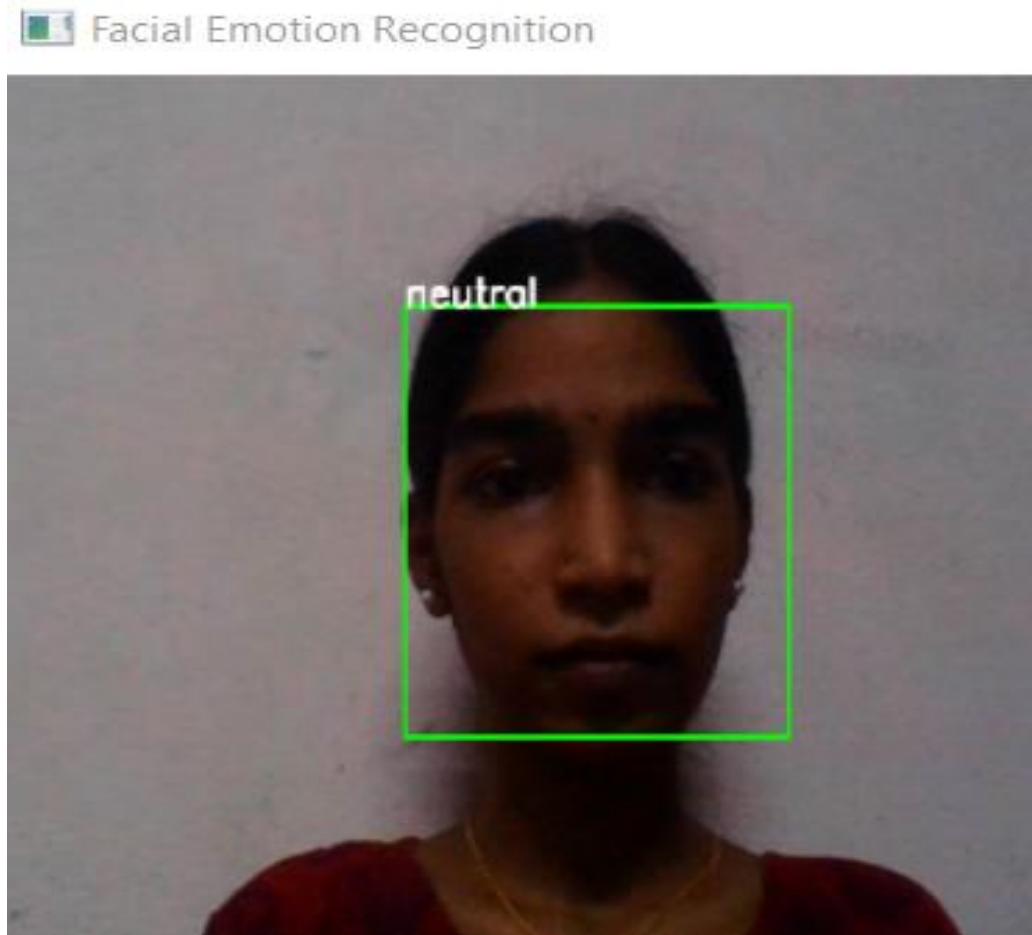


Fig 4.8.3: Image detection

4.9 OUTPUT SCREENSHOTS

Displaying the neutral emoji for the neutral expression of an user.

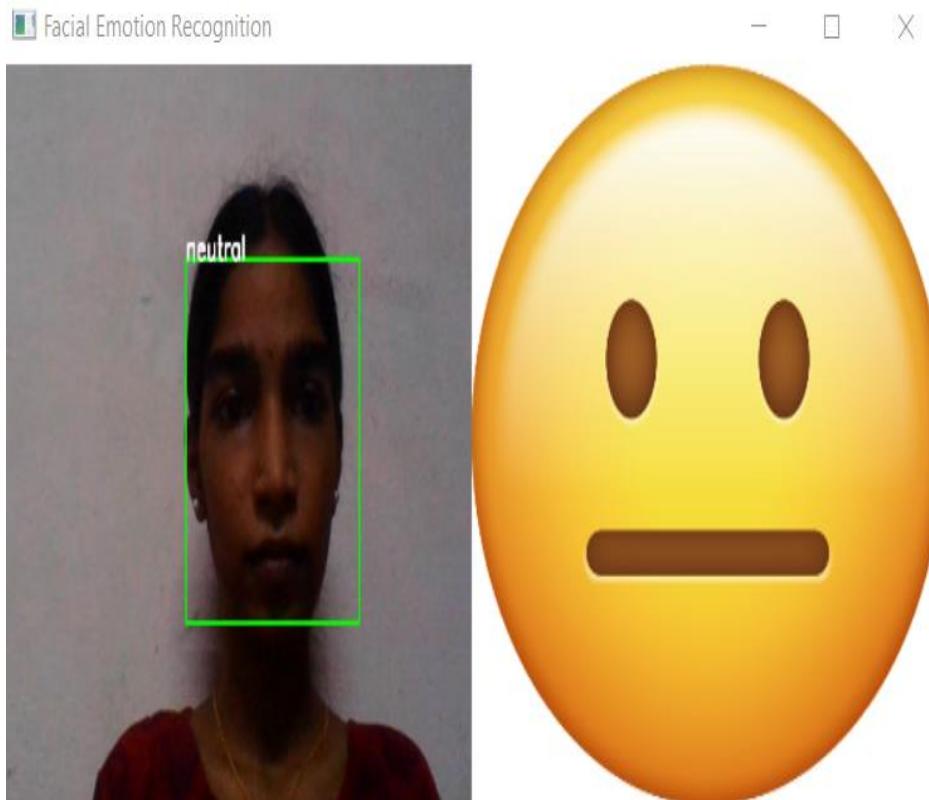


Fig 4.9.1: Neutral

Displaying Happy Emoji for the happy facial expression of an user.

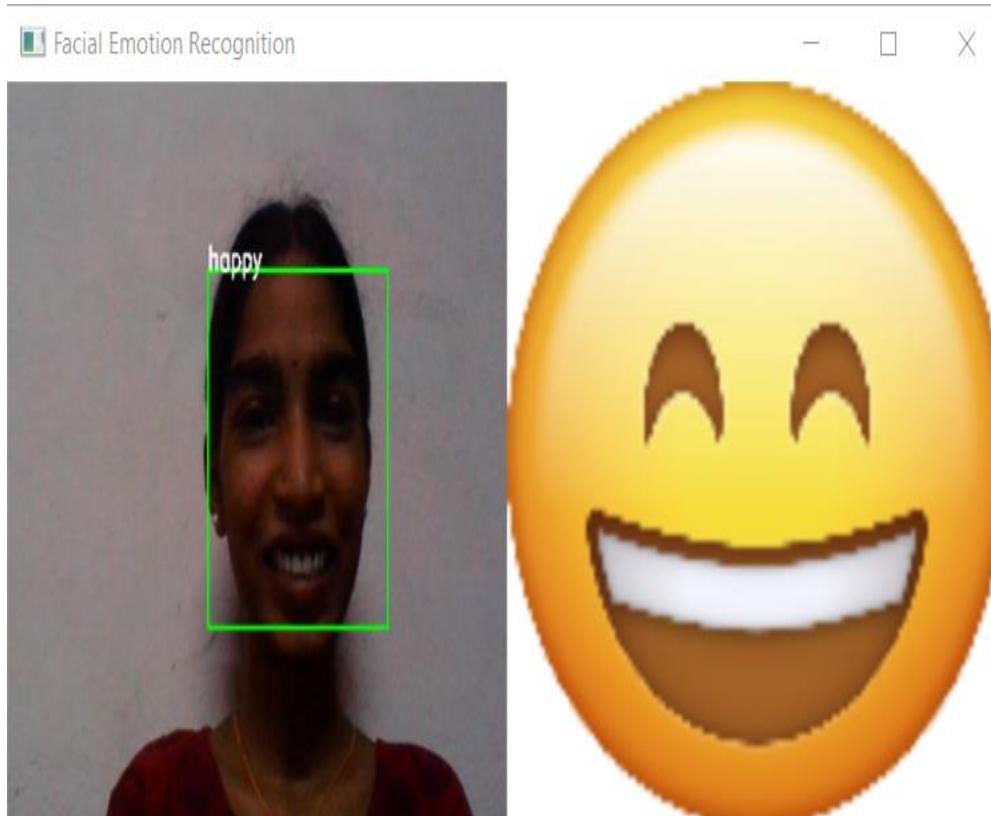


Fig 4.9.2: Happiness

Displaying surprise emoji for the surprise facial expression of an user

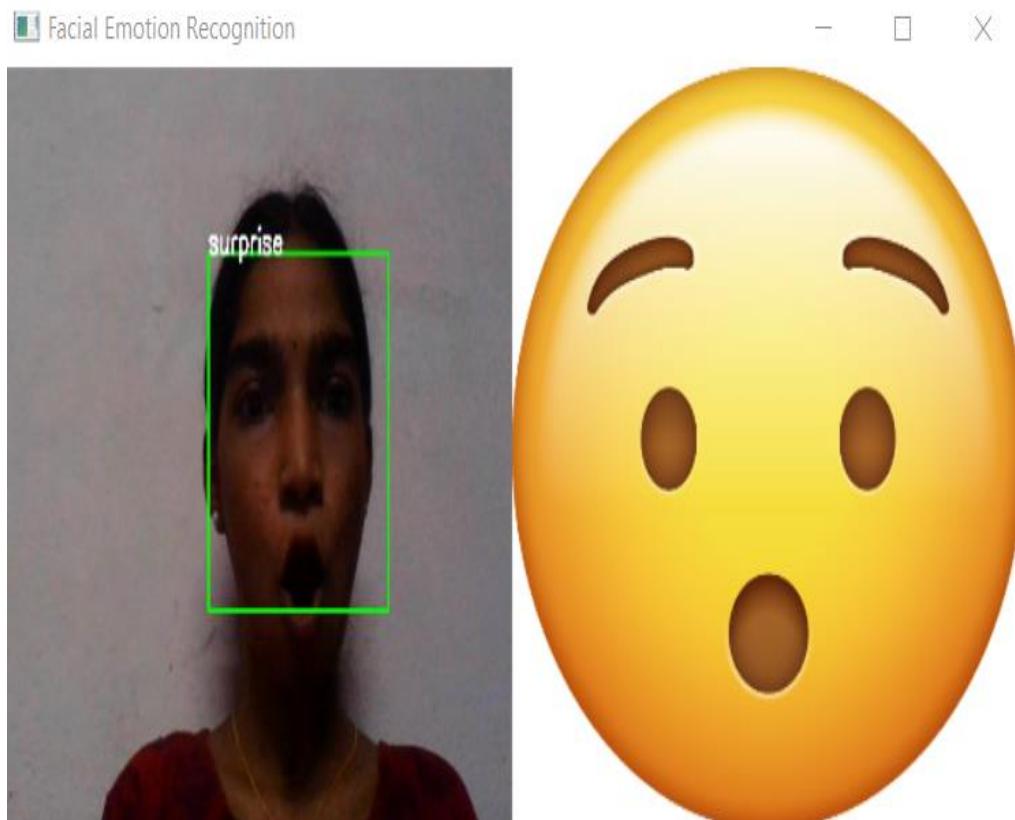


Fig 4.9.3: Surprise

Displaying sadness emoji for the sad facial expression of an user

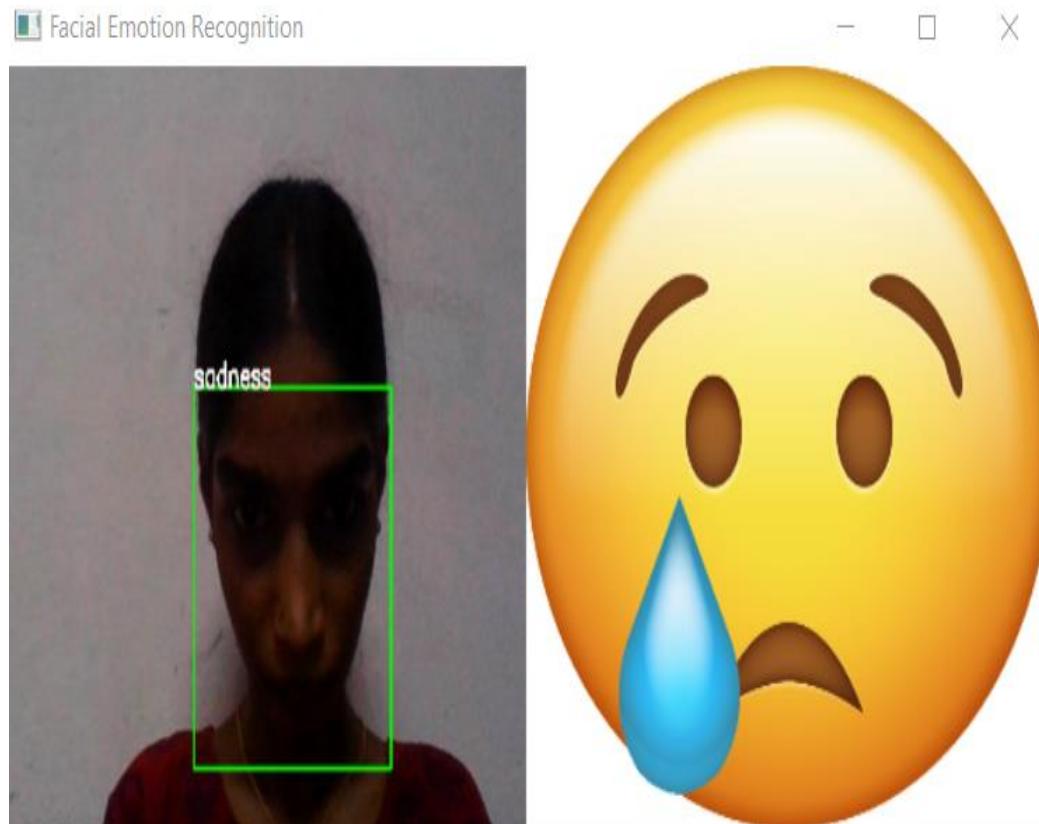


Fig 4.9.4: Sadness

Displaying anger emoji for the anger facial expression of an user

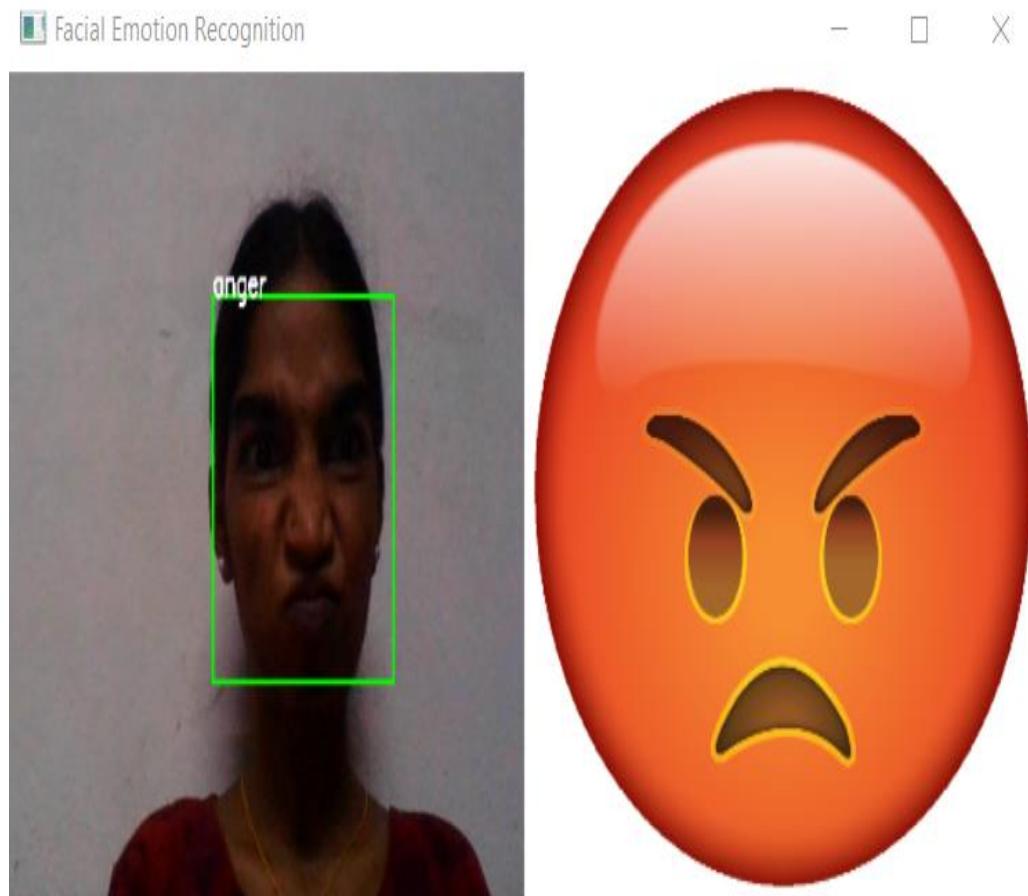


Fig 4.9.5: Anger

Displaying disgust emoji for the disgust facial expression of an user

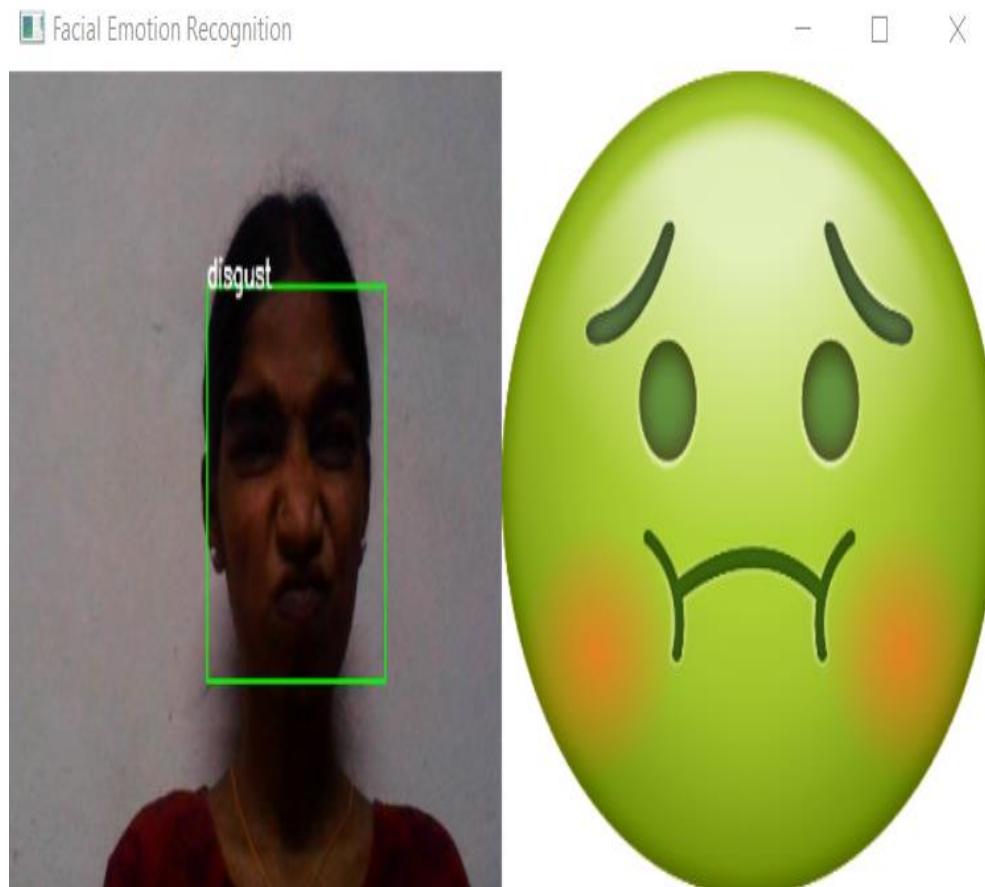


Fig 4.9.6: Disgust

Displaying disgust emoji for the disgust facial expression of an user

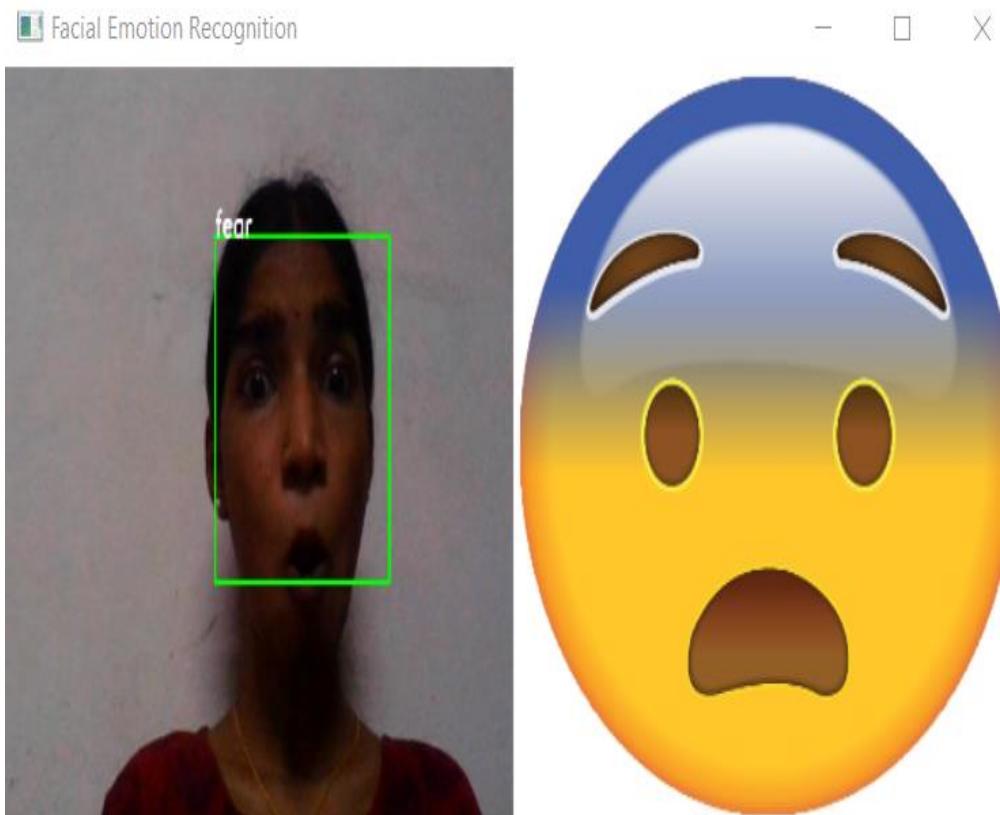


Fig 4.9.7: Fear

5.CONCLUSION

The project is a Human emotion detection using emoji. It uses machine learning, python to predict the emotions of the people and represents those emotions using emoji. It includes Image preprocessing, detection of the face, extraction of features, classification of emotion and displays the facial expression into corresponding emoji.

This project develops an automatic facial emotion recognition system in which an emoji is used for giving the output for the recognised facial expression. It mainly focuses on live videos taken from webcam. The emotions used in this project include happy, sad, surprise, anger, disgust, fear and neutral.

6. REFERENCES

https://www.researchgate.net/publication/333686711_Facial_Emoji_Recognition

Dataset is taken from <https://www.kaggle.com/c/emotion-detection-from-facial-expressions/data>

Open CV from <https://opencv.org/>