TEXAS STATE UNIVERSITY

Advanced Software Engineering Project
CS 5394
Summer II

Facial Recognition of Emotions
Software Requirements Specification

By Group 5394_1

MEMBERS
Punitha Valli Devaraj - p_d120
Sonali Bante – s_b611
Rishita Sharma- r_s607

INSTRUCTOR
Rodion Podorozhny

Table of Contents

1. INTRODUCTION

1.1 Purpose

The purpose of this Software Requirement Specifications (SRS) document is to provide an overview of our software application called Faced Recognition of Emotions. The audience of this SRS will be the Data Science analyst team, Machine Learning experts who will be developing the system and clients who will be receiving the system, who describe their requirements which they wish to obtain in the final product.

1.2 Scope

The scope of this project is to build efficient face recognition of emotions with Keras using Deep Learning concepts. The user will make the machine identify the basic human emotions such as happy, sad, angry, fear, disguise, surprise, neutral. We train and design deep Convolutional Neural Networks (CNN) using OpenCV libraries. Once it is trained, it is deployed with a web interface using Flask to make predictions for inference and make it functional for further research developments using it.

1.3 Definitions, Acronyms, and Abbreviations

Name	Description
API	Application Programming Interface
CNN	Convolutional Neural Networks
OpenCV	Open Source Computer Vision Library
FER	Facial Expression Recognition
tf	TensorFlow

1.4 References

[1] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements and Specifications.
https://standards.ieee.org/standard/830-1998.html
[2] R.S Pressman, Software Engineering-A Practitioner's approach, R.S Pressman and Associates, Inc., 1996.
[3] Dataset for this project is taken from Kaggle - Challenges in Representation Learning Facial Expression Recognition challenge
Challenges in Representation Learning: Facial Expression Recognition Challenge
https://www.kaggle.com/jonathanoheix/face-expression-recognition-dataset?
[4] Data Science Toolkit/Platform used: Anaconda Navigator
https://www.anaconda.com/products/individual

1.5 Overview

The remaining sections of the SRS document are organized as follows:

- Section 2 introduces the overall description of the project.
- Section 3 describes specific requirements

2. OVERALL DESCRIPTION

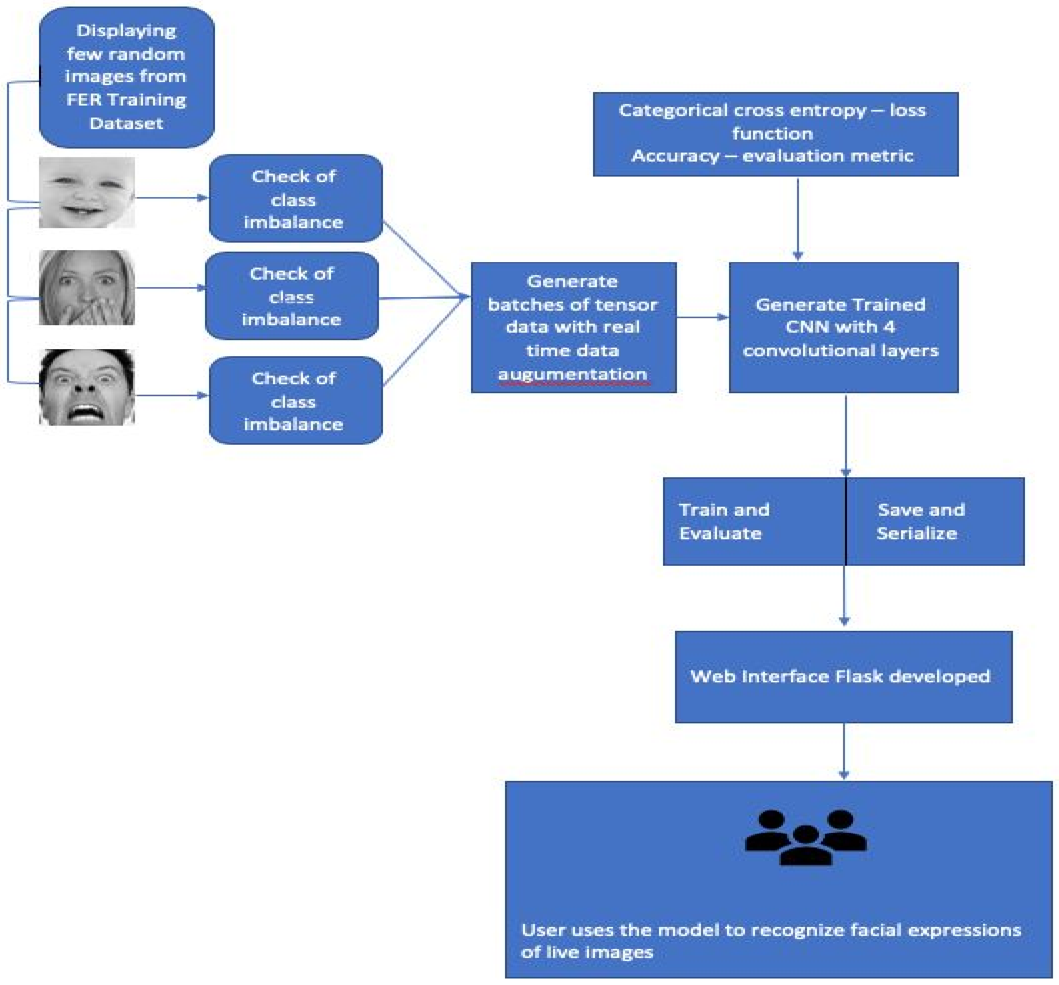
2.1 Product Perspective

We train and design a deep Convolutional Neural Networks (CNN) using tf.keras which is TensorFlow's high-level API for building and training deep learning models. Keras is running on top of TensorFlow, using Keras in deep learning allows for easy and fast prototyping as well as running seamlessly on CPU and GPU. This framework is written in Python code which is easy to debug and allows ease for extensibility.

CNN is primarily used here because it is one of the popular Deep Artificial Neural Networks which are majorly used in image recognition, image clustering and classification, and object detection. CNN uses relatively less preprocessing when compared with the other algorithms of image processing. CNN consists of different layers. There are input layers and output layers. Between these layers, there are some multiple hidden layers. There are no limitations for hidden layers present in the network. The input layer takes the input and trains specifically and gives an output from the output layer. With the help of CNN, we can use a large amount of data more effectively and accurately.

The face detection part is done using OpenCV where the face detection classifiers automatically detect faces and draw boundary boxes around them to analyze. Here OpenCV has opted because it is mainly used for Image Processing, like read or write images, face detection and its features, text recognition in images, detection of shapes, modifying the image quality and colors, for developing augmented reality apps.

Once the model is trained and saved, then it is deployed with a web interface using Flask to make predictions for inference and make it functional for further developments. All these are implemented in Jupyter Notebook through Anaconda Navigator.



Displaying few random images from FER Training Dataset
Check of class imbalance
Check of class imbalance
Check of class imbalance
Categorical cross entropy – loss function
Accuracy – evaluation metric
Generate batches of tensor data with real time data augumentation
Generate Trained CNN with 4 convolutional layers
Train and Evaluate
Save and Serialize
Web Interface Flask developed
User uses the model to recognize facial expressions of live images

Fig1: Sequence Diagram

2.2 Product Functions

The following list of functional descriptions explains the installations, functional importance, and features of the Face Recognition using Emotions.

2.2.1 Installation - Libraries Requirement

2.2.1.1 Description: This model mainly requires Jupyter Notebook. It can be installed through Anaconda where it serves as a giant solution for Data Science and ML platforms. This toolkit involves other platforms like spyder, Pytorch, SciPy, Numpy, Pandas, Matplotlib, and so on.
For this project, Keras, TensorFlow, Numpy, Pandas, Matplotlib, livelossplot, and Seaborn libraries are required. Pip install can be either done through graphical anaconda or through the terminal.

Rationale: Installation of conda and set up of all the libraries is successful.

2.2.2 Version Compatible

2.2.2.1 Description: All the versions should be compatible with one another. Mainly it requires Python 3.7 and above and TensorFlow to be version 2.0.0 and above.

Rationale: Version compatible is successful

2.2.3 Loading Dataset

2.2.3.1 Description: Dataset should be downloaded from Kaggle and the user should be able to sync it with Jupyter Notebook set book after importing all the libraries. Utility functions are used to load the data and plotted through matplotlib. Each image is grayscale and by default is 48*48 pixels.

Rationale: This function allows the dataset to be displayed with random images from the 7 categories.

2.2.4 Class Balance Check

2.2.4.1 Description: This function is used to check the distribution of images around the 7 groups to determine if there is any class imbalance problem. A training set of data is used. If a particular emotion has a low number of images, the user can use a data augmentation method to generate more samples from the minority classes.

Rationale: Class Check balance is checked before doing the modeling and consistency is maintained.

2.2.5 Training and Validating Batches

2.2.5.1 Description: This function imports the ImageDataGenerator from Keras preprocessing images library and creates data augmentation images. The user uses the horizontal axis as a parameter, which flips images in the horizontal axis. First, this process is done using training data then the same process is repeated with different objects for Validation.

Rationale: Paths of training and validation image directories are created and generated batches of augmented data.

2.2.6 Generate trained CNN

2.2.6.1 Description: This function designs and trains Convolutional Neural Network with 4 convolutional layers and 2 fully connected layers to predict the 7 facial emotions. This uses categorical cross-entropy as the loss function and accuracy as the evaluation metric. Always the losses should be minimum which is achieved through MiniBatch Gradient Descent Algorithm and the accuracy is achieved high with correct weights and evaluation function

Rationale: The user is able to design and train CNN by minimizing the neural loss and increasing the accuracy.

2.2.7 Web Interface with Flask

2.2.7.1 Description: This function is used to deploy the model using a Web interface with Flask which applies the module to real-time image data. This can be tested using the webcam which inputs live images, hence performs real-time facial recognition through emotions.

Rationale: The user implements the model and makes it functional to deploy and test live images.

2.3 User Characteristics

Users of this model are the Data Engineer team, Data Science team, Machine learning experts which help business platforms to analyze and implement it in different platforms according to the requirement. Users must know how Deep learning concepts and how to launch the working model and ought to have a negligible instructive level.

2.4 Constraints

The constraints of this model functions are as follows:

- Primarily works on CPU but compatible for GPU as well
- Model response time can take no longer than 10 seconds to detect the emotion
- TB of data cannot be imported in this model. Can hold MB and GB data.

2.5 Assumptions and Dependencies

This model requires access to Anaconda Navigator, Jupyter Notebook, Command Prompt or Terminal, Web camera, Flask, and accessible all-time through the internet. It is expected that the user has access to all of these which can deduct the emotions of facial expressions.

3. SPECIFIC REQUIREMENTS

3.1 Functional Requirements

3.1.1 Installation - Libraries Requirement

Description:	Refer to section 2.2.1.1
Inputs:	Install using pip through command prompt or with binary executables in windows.
Source:	Official release on respective websites
Outputs:	Up-to-date system to built code
Destination:	Command Prompt or Terminal
Requires:	User to execute installation commands.
Pre-condition:	Environment executable should be downloaded and ready to install with proper environment variables set up.
Post-condition:	The environment is ready with proper extensions.
Side-effects:	If dependencies are not installed properly it may cause an error later

3.1.2 Version Compatible

Description:	Refer to section 2.2.2.1
Inputs:	None
Source:	Official release on respective websites
Outputs:	Proper installation of the required setup
Destination:	Command Prompt or Terminal

Requires:	Python 3.7 and above and TensorFlow to be version 2.0.0 and above.
Pre-condition:	Functional OS and executables
Post-condition:	Functional environment.
Side-effects:	None

3.1.3 Loading Dataset

Description:	Refer to section 2.2.3.1
Inputs:	Kaggle account credentials
Source:	Kaggle and Jupyter notebook official websites
Outputs:	Synced and plotted data
Destination:	Jupyter Notebook
Requires:	Import required libraries
Pre-condition:	Kaggle dataset synced with Jupyter notebook
Post-condition:	The dataset is populated with random images from the 7 categories.
Side-effects:	None

3.1.4 Class Balance Check

Description:	Refer to section 2.2.4.1
Inputs:	The training set of data
Source:	Kaggle dataset
Outputs:	Balanced classes in the dataset

Destination:	Mentioned 7 groups
Requires:	Loaded and synced dataset
Pre-condition:	Imported libraries and dataset, Class Check
Post-condition:	Consistency is maintained
Side-effects:	None

3.1.5 Training and Validating Batches

Description:	Refer to section 2.2.5.1
Inputs:	Horizontal axis as a parameter
Source:	Imported Libraries
Outputs:	Data augmentation images
Destination:	Keras preprocessing images library
Requires:	Libraries
Pre-condition:	System requirement set up
Post-condition:	Paths of training and validation image directories are created
Side-effects:	None

3.1.6 Generate trained CNN

Description:	Refer to section 2.2.6.1
Inputs:	Adam as an optimizer, categorical cross-entropy as loss function
Source:	Batches of augmented data from the training and validation batch

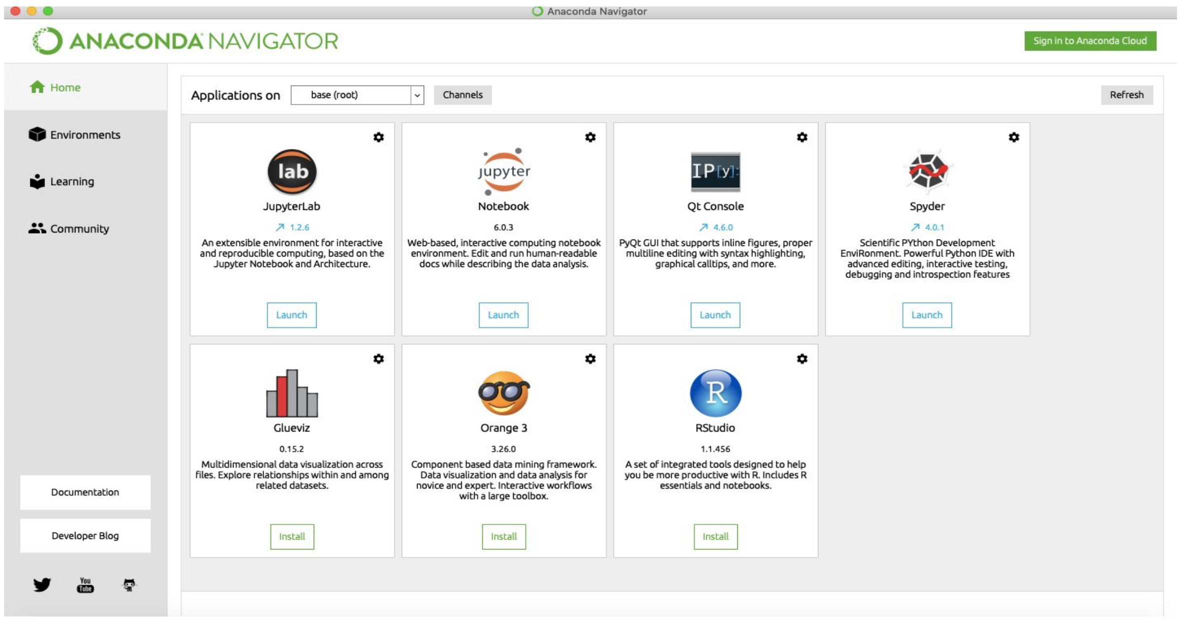
Outputs:	Predicts the 7 facial emotions
Destination:	Observe live training loss and accuracy plots in Jupyter notebook for Keras
Requires:	MiniBatch Gradient Descent Algorithm, model.fit() method, ModelCheckPoint() to save the weights
Pre-condition:	Losses are minimum and high validation accuracy is obtained.
Post-condition:	Designed and trained CNN
Side-effects:	None

3.1.7 Web Interface with Flask

Description:	Refer to section 2.2.7.1
Inputs:	Real-time image data
Source:	Live data feed
Outputs:	Deployed model
Destination:	The web interface in Flask
Requires:	The modules - main.py script
Pre-condition:	Flask set up
Post-condition:	Implemented web interface
Side-effects:	None

3.2 Wire Frame

3.2.1 Anaconda Navigator Setup



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

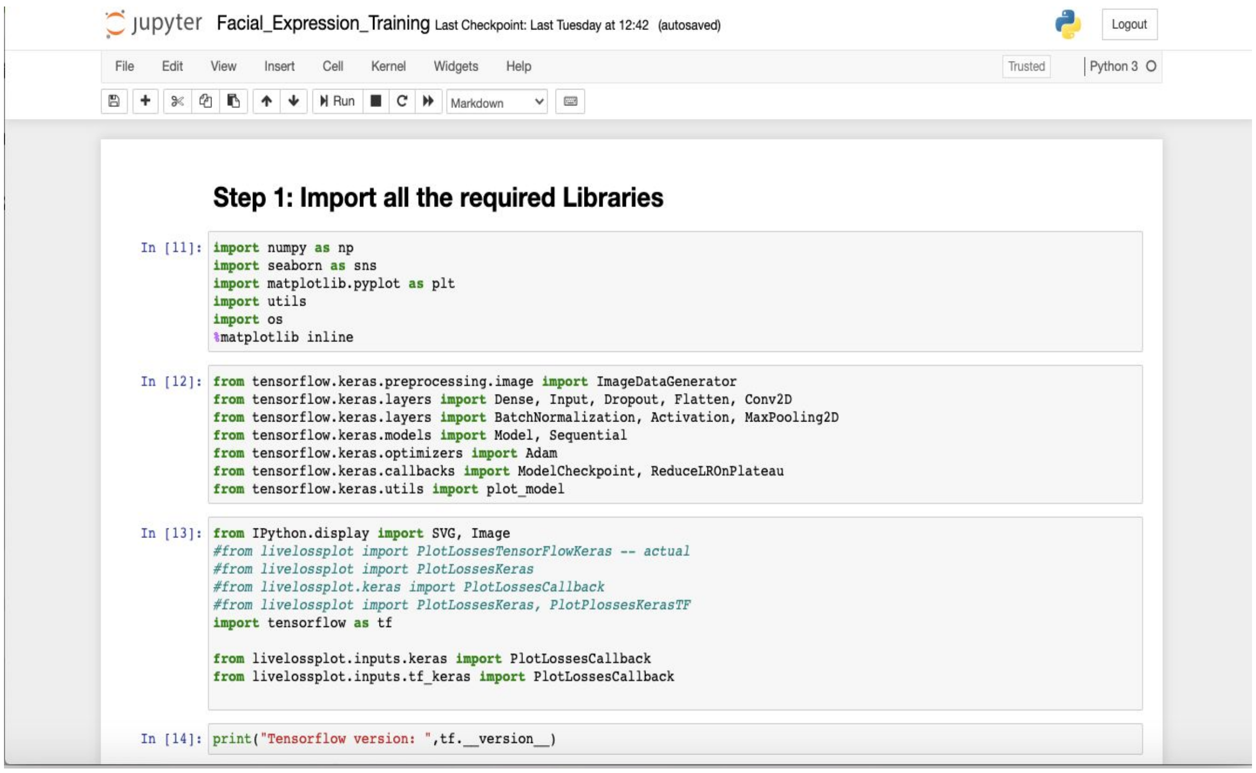
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

3.2.2 Jupyter Notebook


jupyter Facial_Expression_Training Last Checkpoint: Last Tuesday at 12:42 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown

Step 1: Import all the required Libraries

```
In [11]: import numpy as np
         import seaborn as sns
         import matplotlib.pyplot as plt
         import utils
         import os
         %matplotlib inline
```

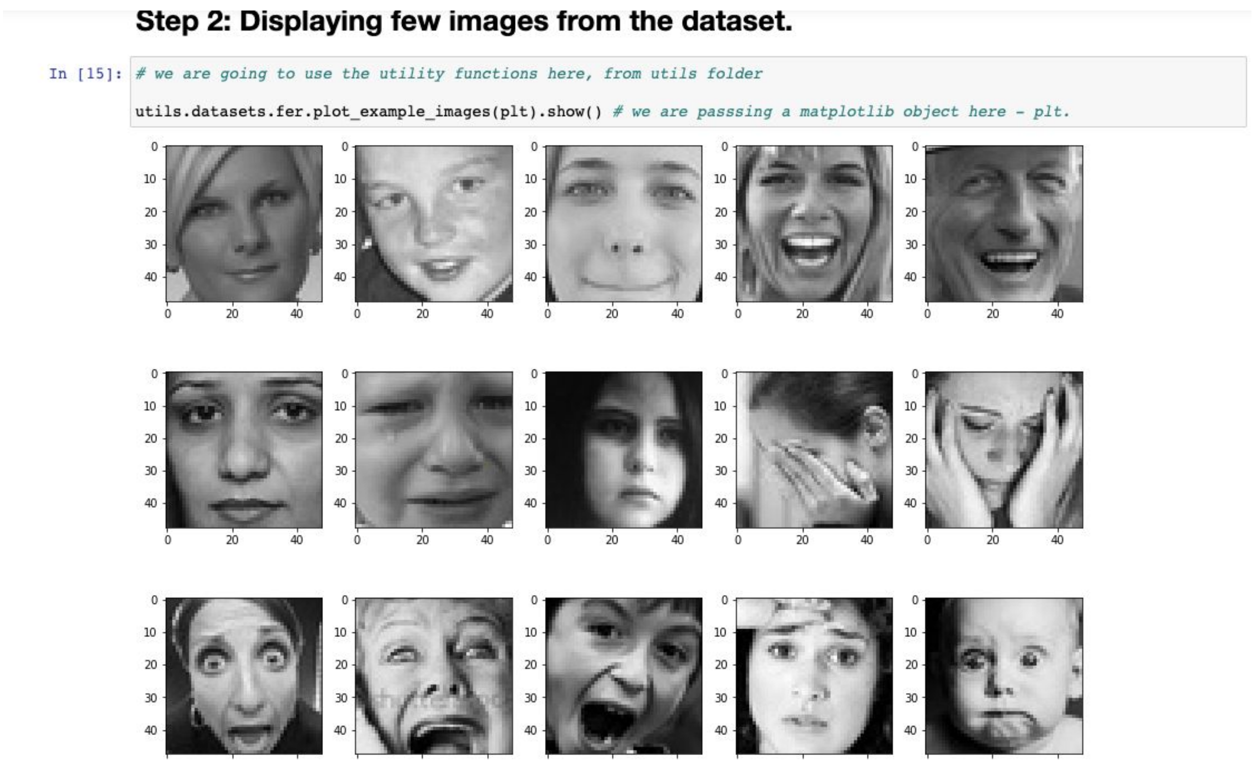
```
In [12]: from tensorflow.keras.preprocessing.image import ImageDataGenerator
         from tensorflow.keras.layers import Dense, Input, Dropout, Flatten, Conv2D
         from tensorflow.keras.layers import BatchNormalization, Activation, MaxPooling2D
         from tensorflow.keras.models import Model, Sequential
         from tensorflow.keras.optimizers import Adam
         from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
         from tensorflow.keras.utils import plot_model
```

```
In [13]: from IPython.display import SVG, Image
         #from livelossplot import PlotLossesTensorFlowKeras -- actual
         #from livelossplot import PlotLossesKeras
         #from livelossplot.keras import PlotLossesCallback
         #from livelossplot import PlotLossesKeras, PlotplossesKerasTF
         import tensorflow as tf

         from livelossplot.inputs.keras import PlotLossesCallback
         from livelossplot.inputs.tf_keras import PlotLossesCallback
```

```
In [14]: print("Tensorflow version: ",tf.__version__)
```

3.2.3 Dataset training images



Step 2: Displaying few images from the dataset.

```
In [15]: # we are going to use the utility functions here, from utils folder

         utils.datasets.fer.plot_example_images(plt).show() # we are passsing a matplotlib object here - plt.
```

3.3 Performance Requirements

The performance requirements are as follows:

- The system should be able to recognize and process a single image in less than 5 seconds.
- At least 32000 images can be processed at a time.

3.4 Logical Requirements

The model must be able to process Gigabyte data images. Each user can have their own Jupyter and Flask setup. It should be available to access at all times. All the setup and files are accessible through Git. None of the users can modify the image data files.

A logical database can stretch over multiple physical hard disks and information files.
The data's initial storage system is retrieved from Kaggle and can be stored in a local disk or can be accessed from the cloud. All given information must be accessible from a single source. An example would be a personal computer able to access its information files stored on multiple hard drives from a single user interface. None of the users has access to modify data files.

3.5 Design Constraints

Design Constraints are as follows:

- Able to support Windows, MAC, and Linux platforms.
- The system supports the following web browsers: Chrome, Safari, and Firefox.

3.6 Software System Attributes

3.6.1 Availability

The Face recognition system is available on git and can be used anytime. The user has to have a specific Git account to access and is added to the master branch. The application can be periodically tested to ensure the availability of API.

3.6.2 Security

The system would use the computer's default operating system security. The system will not use any of its own security features. Github accounts are secured, private, and maintained by each user.

3.6.3 Usability

The goal is accomplished quickly with few or no user errors. It is user friendly and easy to use. Users should be able to import data into web pages and easily handle it. Messages shown to the user should be meaningful and the user should be able to resolve the conflicts easily through them.

3.6.4 Reliability

This model is available 24/7. The software might fail from time to time due to problems with capturing the image. In such cases, the software will not crash and ask the user to re-enter the image which would rectify this issue, hence having a reliable connection throughout.

3.6.5 Robustness

OpenCV will constraint input options wherever it is possible to prevent failures. Upon an image failure, OpenCV will apply the current error handling strategy to respond.

3.6.6 Maintainability

The framework is made in python which can be easily debugged and allows easy extensions of the project. Any Maintenance activity or version update activity should be priorly informed to the user, to update their model if required.