

APPLICATION  
FOR  
HUMANOID  
ROBOT

# CHAPTER 1:

## INTRODUCTION

A humanoid robot is a robot with its body shape built to resemble the human body. The design may be for functional purposes, such as interacting with human tools and environments, for experimental purposes, such as the study of bipedal locomotion, or for other purposes. In general, humanoid robots have a torso, a head, two arms, and two legs, though some forms of humanoid robots may model only part of the body, for example, from the waist up. Some humanoid robots also have heads designed to replicate human facial features such as eyes and mouths. Humanoid robots, while being one of the smallest groups of service robots in the current market, have the greatest potential to become the industrial tool of the future. Companies like Softbank Robotics have created human-looking robots to be used as medical assistants and teaching aids. Currently, humanoid robots are excelling in the medical industry, especially as companion robots. University of Southern California Professor Maja Matarić has been pairing robots with patients since 2014. Her robots helped children with autism copy the motions of socially assistive robots and, in 2015, the robots assisted stroke recovery victims with upper extremity exercises. The patients were more responsive to the exercises when promoted and motivated by the robot. However, companies are now using humanoid robots to fill engineering tasks. A four-year joint research project was conducted by Joint Robotics Laboratory and Airbus Group to use humanoid robotic technology in aircraft manufacturing facilities. By using humanoid robots on aircraft assembly lines, Airbus looks to relieve human operators of some of the more laborious and dangerous tasks. The human employers could then concentrate on higher value tasks.

- Our Application can recognize people's faces using **facial recognition** technology
- Our Application can have a conversation with people using **NLP** technology
- Our Application is built to be interfaced with the Humanoid Robot
- This type of Robot can be used in fields such as Therapy, Teaching and Personal Assistance

## CHAPTER 2:

### LITERATURE SURVEY

A method of Face detection has been proposed in [1]. The method is based on evaluating well-normalized local histograms of image gradient orientations in a dense grid. Similar features have seen increasing use over the past decade. The basic idea is that local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions. In practice this is implemented by dividing the image window into small spatial regions (“cells”), for each cell accumulating a local 1-D histogram of gradient directions or edge orientations over the pixels of the cell. The combined histogram entries form the representation. For better invariant to illumination, shadowing, etc., it is also useful to contrast-normalize the local responses before using them. This can be done by accumulating a measure of local histogram “energy” over somewhat larger spatial regions (“blocks”) and using the results to normalize all of the cells in the block. We will refer to the normalized descriptor blocks as Histogram of Oriented Gradient (HOG) descriptors. Tiling the detection window with a dense (in fact, overlapping) grid of HOG descriptors and using the combined feature vector in a conventional SVM based window classifier gives our human detection chain. In [2] the authors address the problem of Face Alignment for a single image. They show how an ensemble of regression trees can be used to estimate the face’s landmark positions directly from a sparse subset of pixel intensities, achieving super-realtime performance with high quality predictions. They present a general framework based on gradient boosting for learning an ensemble of regression trees that optimizes the sum of square error loss and naturally handles missing or partially labelled data. They show how using appropriate priors exploiting the structure of image data helps with efficient feature selection. Different regularization strategies and its importance to combat overfitting are also investigated. In addition, they analyse the effect of the quantity of training data on the accuracy of the predictions and explore the effect of data augmentation using synthesized data. For face encodings, the authors of [3] propose a deep neural network of ResNet - 34 architecture which produces a 128 dimensional feature vector for a given input face.

Coming to the conversation part, the authors of [4] have created a robotic system that any person could talk to in the English language, in particular, pairing an artificial intelligence algorithm that processes natural language with a physical robot that could synthesize speech. The result, called CleverNAO, is a successful combination of a chatbot application named Cleverbot with the NAO robot doing the speech synthesis. An attempt to include speech recognition was also made, with mixed success. The authors of [5] take it one step further with ERICA, an autonomous android system capable of conversational interaction, featuring advanced sensing and speech synthesis technologies, and arguably the most humanlike android built to date. This robot actually has a human like face with actuators for lip and eye movement. Looking into various research work, face recognition and NLP have not been combined together in a single piece of work. Our project attempts to do just that.

## CHAPTER 3:

### THEORY

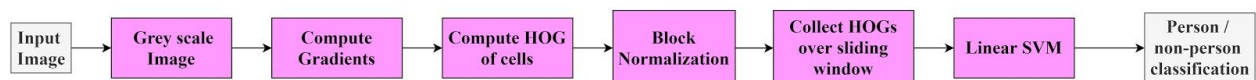
#### 3.1 FACE RECOGNITION

Face recognition is a method of identifying or verifying the identity of an individual using their face. Face recognition systems can be used to identify people in photos, video, or in real-time. Face Recognition is being used in more and more applications. Starting from unlocking your phone with your face to paying money with your face, face recognition has found its way in the market. Facebook's algorithms are able to recognize your friends' faces after they have been tagged only a few times. It's pretty amazing technology that Facebook can recognize faces with 98% accuracy which is pretty much as good as humans can do!

Face Recognition has 4 steps:

1. Face Detection
2. Face Alignment
3. Encoding Faces
4. Comparing Faces

##### 3.1.1 Face Detection



*Fig 3.1 Steps in Face Detection*

##### 3.1.1.1 Converting RGB image into Grey Scale image

The first step in our face detection pipeline is to convert the input RGB video stream into Grey scale video stream. We take each frame of our video stream and convert the RGB frame into a grey scale frame. We have used a webcam to feed the live input video. Webcam resolution = 640 x 480 pixels.

### 3.1.1.2 Compute Gradients

We calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is easily achieved by filtering the image with the following kernels.



The figure shows two 1D kernels. The first kernel is horizontal, with values -1, 0, and 1. The second kernel is vertical, with values -1, 0, and 1.

-1	0	1
----	---	---

-1
0
1

*Fig 3.2 X and Y direction kernels*

Next, we can find the magnitude and direction of gradient using the following formulas

$$g = \sqrt{g_x^2 + g_y^2}$$
$$\theta = \arctan \frac{g_y}{g_x}$$

We then get 2 matrices, namely Gradient Direction matrix and Gradient Magnitude matrix. But why do we replace pixels with Gradients?

If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the direction that brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve!

### 3.1.1.3 Compute the Histogram of Gradients in 8x8 cells

In this step, the image is divided into 8×8 cells and a histogram of gradients is calculated for each 8×8 cells.

Why did we divide the image into 8x8 cells?

One of the important reasons to use a feature descriptor to describe a patch of an image is that it provides a compact representation. An  $8 \times 8$  image patch contains  $8 \times 8 = 64$  pixel values. The gradient of this patch contains 2 values ( magnitude and direction ) per pixel which adds up to  $8 \times 8 \times 2 = 128$  numbers. Not only is the representation more compact, calculating a histogram over a patch makes this representation more robust to noise. Individual gradients may have noise, but a histogram over  $8 \times 8$  patch makes the representation much less sensitive to noise.

After dividing the image into many  $8 \times 8$  pixel cells, In each cell, the magnitude values of these 64 cells are binned and cumulatively added into 9 buckets of unsigned direction (no sign, so 0-180 degree rather than 0-360 degree; this is a practical choice based on empirical experiments). For better robustness, if the direction of the gradient vector of a pixel lays between two buckets, its magnitude does not all go into the closer one but proportionally split between two. For example, if a pixel's gradient vector has magnitude 8 and degree 15, it is between two buckets for degree 0 and 20 and we would assign 2 to bucket 0 and 6 to bucket 20. This interesting configuration makes the histogram much more stable when small distortion is applied to the image.

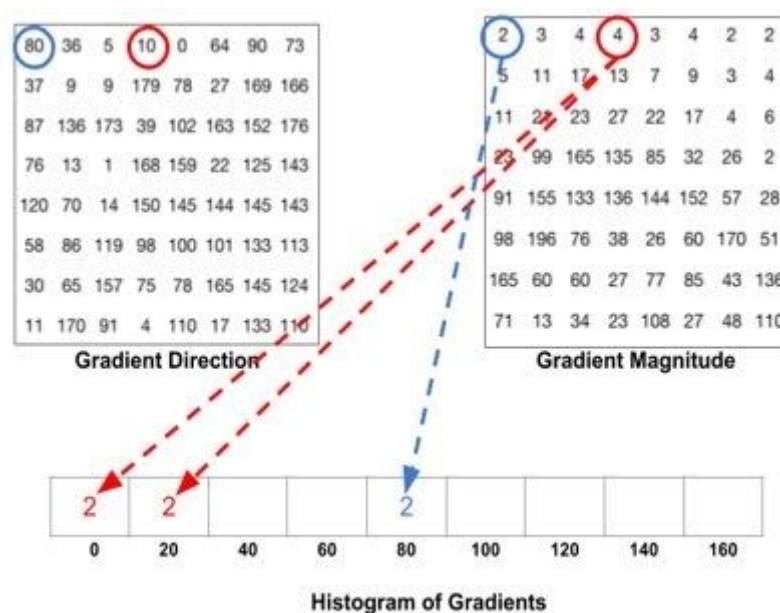


Fig 3.3 HOG from Gradient Direction and Magnitude matrices

There is one more detail to be aware of. If the angle is greater than 160 degrees, it is between 160 and 180, and we know the angle wraps around making 0 and 180 equivalent. So in the example below, the pixel with angle 165 degrees contributes proportionally to the 0 degree bin and the 160 degree bin.

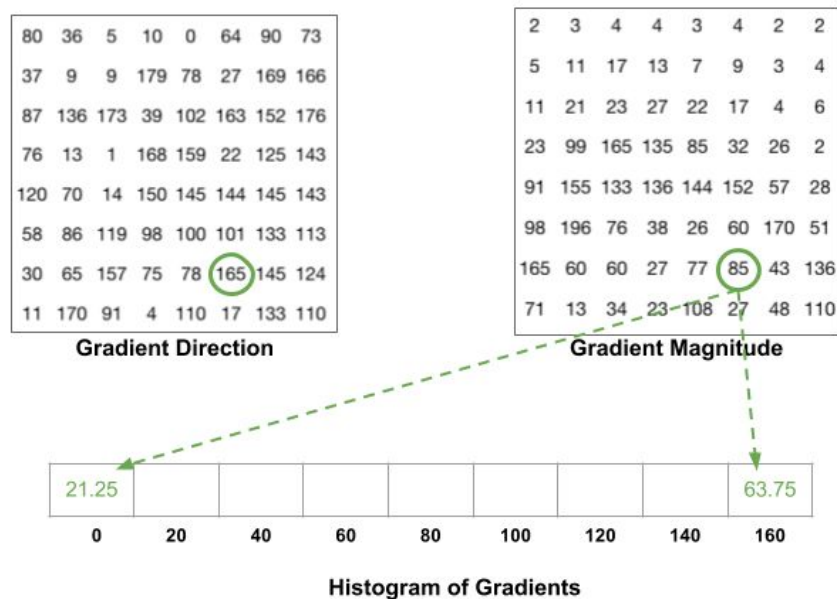


Fig 3.4 HOG for gradient directions greater than 160 degrees

The contributions of all the pixels in the 8×8 cells are added up to create the 9-bin histogram. For the patch above, it looks like this

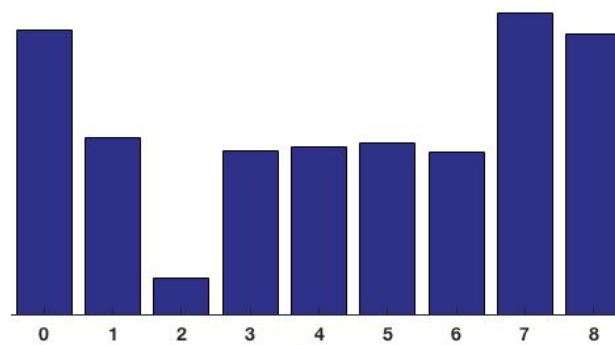


Fig 3.5 9 bin Histogram of Oriented Gradients of a cell

In our representation, the y-axis is 0 degrees. You can see the histogram has a lot of weight near 0 and 180 degrees, which is just another way of saying that in the patch gradients are pointing either up or down.



### 3.1.1.4 16x16 Block Normalization

In the previous step, we created a histogram based on the gradient of the image. Gradients of an image are sensitive to overall lighting. If you make the image darker by dividing all pixel values by 2, the gradient magnitude will change by half, and therefore the histogram values will change by half. Ideally, we want our descriptor to be independent of lighting variations. In other words, we would like to “normalize” the histogram so they are not affected by lighting variations.

Let's look at this example on normalization. Let's say we have an RGB color vector  $[128, 64, 32]$ . The length of this vector is  $\sqrt{128^2 + 64^2 + 32^2} = 146.64$ . Dividing each element of this vector by 146.64 gives us a normalized vector  $[0.87, 0.43, 0.22]$ . Now consider another vector in which the elements are twice the value of the first vector  $2 \times [128, 64, 32] = [256, 128, 64]$ . Normalizing  $[256, 128, 64]$  will result in  $[0.87, 0.43, 0.22]$ , which is the same as the normalized version of the original RGB vector. You can see that normalizing a vector removes the scale.

We normalize over a bigger sized block of  $16 \times 16$ . A  $16 \times 16$  block has 4 histograms which can be concatenated to form a  $36 \times 1$  element vector. The block is normalized.

### 3.1.1.5 Collect HOGs over sliding window

A window of size  $128 \times 64$  is used with  $16 \times 16$  Blocks.

1. The normalized  $36 \times 1$  vector of first Block is calculated.
2. The Block is then moved by 8 pixels and a normalized  $36 \times 1$  vector is calculated over this Block and the process is repeated.
3. In one window we obtain  $(9 \times 128 \times 64) / (8 \times 8) = 1152 \times 1$  vector

### 3.1.1.6 Linear SVM

Support vector machine is a simple algorithm that every machine learning expert should have in his/her arsenal. Support vector machines are highly preferred by many as it produces significant accuracy with less computation power. Support Vector Machine, abbreviated as SVM can be used for both regression and classification tasks. But, it is widely used in classification objectives. The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

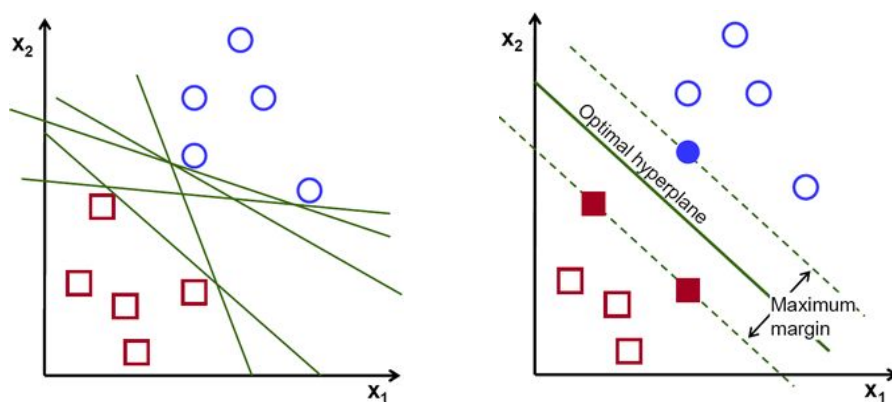


Fig 3.6 Possible Hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.

Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Cost and Gamma are the hyper-parameters that decide the performance of an SVM model. There should be a fine balance between Variance and Bias for any ML model. For SVM, a High value of Gamma leads to more accuracy but biased results and vice-versa. Similarly, a large value of Cost parameter (C) indicates poor accuracy but low bias and vice-versa. Optimum C and gamma need to be found out. Here we have used  $C = 0.01$ .

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions. These functions can be different types. For example linear, nonlinear, polynomial, radial basis function (RBF), and sigmoid. Here, we have used a Linear kernel.

At each window we extract  $1152 \times 1$  HOG vector and apply Linear SVM classifier. If the classifier detects a face with sufficiently large probability, we record the bounding box of the window and hence we detect the face.

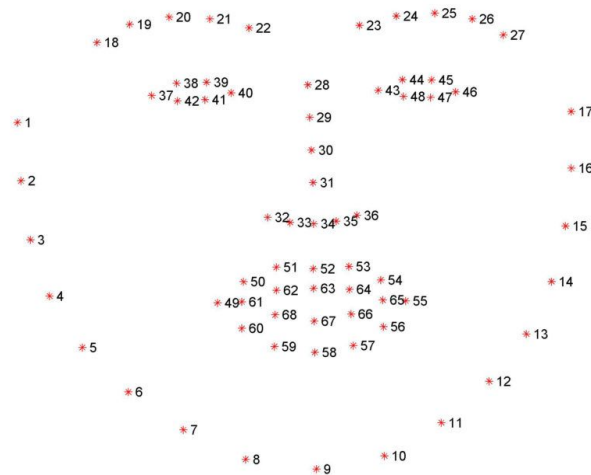
### **3.1.2 Face Alignment**

Face Alignment consists of two steps:

1. Face Landmark Estimation
2. Affine Transformation

### 3.1.2.1 Face Landmark Estimation

First we need to find certain feature points. To do this, we have used an algorithm called face landmark estimation. The basic idea is we will come up with 68 specific points (called landmarks) that exist on every face — Eyes, Eyebrows, Nose, Mouth and Jawline. Then we will use an ensemble of regression trees to be able to find these 68 specific points on any face. We then get the x-y coordinates of the face landmarks in the image.



*Fig 3.7 68 Landmarks found on every human face*

### 3.1.2.2 Affine Transformation

Now that we know where the eyes and mouth are, we'll simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible. We won't do any fancy 3d warps because that would introduce distortions into the image. We are only going to use basic image transformations like translation, rotation and scale that preserve parallel lines (called affine transformation)

1. Translation- Point  $P(x,y)$  can be translated by  $dx$  along the x-axis and by  $dy$  along the y-axis

$$p = \begin{bmatrix} x \\ y \end{bmatrix}, p' = \begin{bmatrix} x' \\ y' \end{bmatrix}, T = \begin{bmatrix} dx \\ dy \end{bmatrix}$$

2. Rotation- Points P(x,y) can be rotated by an angle theta about the origin

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

3. Scaling- Points P(x,y) can be scaled by Sx along the x-axis and by sy along the y-axis

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Homography Matrix - Homography matrix is a 3x3 matrix when multiplied with the input image, performs affine transformation on the input image. The Homography matrix consists of Translation, Rotation and Scale components needed for affine transformation.

$$H = \begin{pmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

The reason we perform face alignment is because the neural network that we use in our next step to find face embeddings is trained on images with straight faces. So this step is used to normalize the input image with respect to the training images used in the neural network making sure that we get accurate results.

### 3.1.3 Encoding Faces

We pass the image of the face into a Convolutional Neural Network. This model is a ResNet network with 29 convolutional layers. It's essentially a version of the ResNet-34 network from the paper Deep Residual Learning for Image Recognition by He, Zhang, Ren, and Sun with a few layers removed. This network was trained on

‘Labelled Faces in the Wild’ dataset by Davis King with an accuracy of **99.38%**. The network gives an output of a Feature vector with 128 dimensions. Resnet addresses the vanishing gradient problem by introducing ‘skip connections’.

### 3.1.4 Comparing Faces

We find the Euclidean distances between the two faces and we associate this face with the closest match. The Euclidean distance between points  $p$  and  $q$  is the length of the line segment connecting them. In Cartesian coordinates, if  $p = (p_1, p_2, \dots, p_n)$  and  $q = (q_1, q_2, \dots, q_n)$  are two points in Euclidean  $n$ -space, then the Euclidean distance ( $d$ ) from  $p$  to  $q$ , or from  $q$  to  $p$  is given by the Pythagorean formula:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.$$

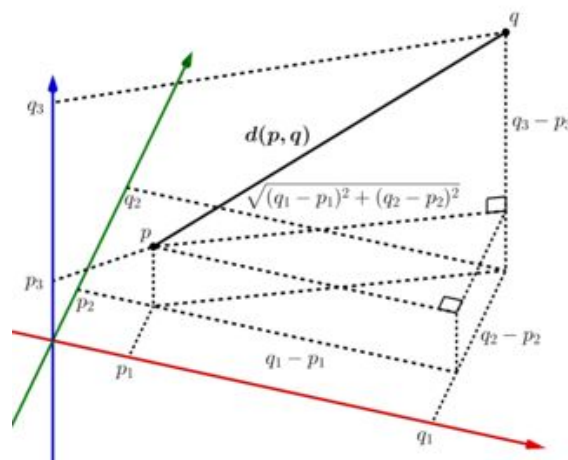


Fig 3.8 Illustration of Euclidean distance for  $n=3$

If the euclidean distance between the input image and the database images is less than 0.6, then the distance of database photo with the least euclidean distance with the input image will be considered as a match. If we don't find a match, we feed this new face into the database and generate its own identity.

Hence, we achieve FACE RECOGNITION.

## **3.2 NATURAL LANGUAGE PROCESSING (NLP)**

Natural language processing (NLP) is a branch of artificial intelligence that helps computers understand, interpret and manipulate human language. NLP draws from many disciplines, including computer science and computational linguistics, in its pursuit to fill the gap between human communication and computer understanding.

Natural language processing helps computers communicate with humans in their own language and scales other language-related tasks. For example, NLP makes it possible for computers to read text, hear speech, interpret it, measure sentiment and determine which parts are important. Today's machines can analyze more language-based data than humans, without fatigue and in a consistent, unbiased way. Considering the staggering amount of unstructured data that's generated every day, from medical records to social media, automation will be critical to fully analyze text and speech data efficiently.

Natural language processing includes many different techniques for interpreting human language, ranging from statistical and machine learning methods to rules-based and algorithmic approaches. We need a broad array of approaches because the text- and voice-based data varies widely, as do the practical applications. Basic NLP tasks include tokenization and parsing, lemmatization/stemming, part-of-speech tagging, language detection and identification of semantic relationships. If you ever diagrammed sentences in grade school, you've done these tasks manually before. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation.

### **3.2.1 Chatbot**

A chatbot is an artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile apps or through the telephone. Why are chatbots important? A chatbot is often described as one of the most advanced and promising expressions of interaction

between humans and machines. However, from a technological point of view, a chatbot only represents the natural evolution of a Question Answering system leveraging Natural Language Processing (NLP). Formulating responses to questions in natural language is one of the most typical Examples of Natural Language Processing applied in various enterprises' end-use applications.

### **3.2.1.1 Types of Chatbots**

Depending on the way bots are programmed, we can categorize them into two variants of chatbots: Rule-Based & Self Learning.

1. Rule-Based Chatbots: This variety of bots answer questions based on some simple rules that they are trained on.
2. Self-Learning Chatbots: This variety of bots rely on Artificial Intelligence(AI) & Machine Learning(ML) technologies to converse with users. Self-learning Chatbots are further divided into Retrieval based and Generative.
  - I. Retrieval based chatbots: The bot is trained to rank the best response from a finite set of predefined responses. The responses here are entered manually, or based on a knowledge base of pre-existing information.
  - II. Generative based chatbots: These chatbots are not built with predefined responses. Instead, they are trained using a large number of previous conversations, based upon which responses to the user are generated. They require a very large amount of conversational data to train.



### 3.2.2 Our Chatbot Architecture

Here, the Bot will have conversations with the person standing in front of it. This is a 3 step process:

1. **Speech to Text conversion:** The user speaks a sentence. This sentence is converted into text using the Google Speech to Text API
2. **Text Processing:** This text sentence is then processed using Google Dialog Flow and Natural Language ToolKit (NLTK)

The bot can understand 5 types of intentions:

01. User is giving his information -- My name is Piyush
02. User is asking his information -- What is my name?
03. User is asking Bot information -- What are your hobbies?
04. Small Talk -- I hate you, tell me a joke
05. User want general information -- Who is Mahatma gandhi? what is coronavirus?

The chat bot identifies the Intention and generates a response for the input text. The output of this Step is in terms of text.

3. **Text to Speech conversion:** The output text is converted into speech by using Google Text to Speech API.

### 3.2.3 Text Processing

We have used a combination of Natural Language Tool Kit (NLTK) and Google Dialogflow to do the job of Text processing.

### 3.2.3.1 Natural Language Tool Kit (NLTK)

The NLTK module is a massive tool kit, aimed at helping you with the entire Natural Language Processing (NLP) methodology. NLTK will aid you with everything from splitting sentences from paragraphs, splitting up words, recognizing the part of speech of those words, highlighting the main subjects, and then even with helping your machine to understand what the text is all about. Few text processing functions include Tokenizing, Stop words filtration, stemming, part of speech tagging, etc.

#### 1. **Tokenizing** - Splitting sentences and words from the body of text.

```
EXAMPLE_TEXT = "Hello Mr. Smith, how are you doing today? The weather is great, and Python is awesome. The sky is pinkish-blue. You shouldn't eat cardboard."
```

Sentence Tokenizing - ['Hello Mr. Smith, how are you doing today?', 'The weather is great, and Python is awesome.', 'The sky is pinkish-blue.', 'You shouldn't eat cardboard.']

Word Tokenizing: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', ',', 'and', 'Python', 'is', 'awesome', '.', 'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', 'n't', 'eat', 'cardboard', '.']

2. **Filtering out Stop Words** – Stop words are words which do not carry useful information in the sentence. Stop words are used in a sentence just for punctuation sake. Eg- 'with', 'they', 'so', 'a', 'the', 'too', 'only', 'those', 'i'

```
EXAMPLE_TEXT = ['This', 'is', 'a', 'sample', 'sentence', ',', 'showing', 'off', 'the', 'stop', 'words', 'filtration', '.']
```

```
Filtered Text = ['This', 'sample', 'sentence', ',', 'showing', 'stop', 'words', 'filtration', '.']
```

3. **Stemming** – Stemming is used to normalize sentences.

Consider: I was taking a **ride** in the car.

I was **riding** in the car.

This sentence means the same thing. The 'ing' in the second sentence is not really required. So Stemming helps us find the basic form of words.

```
example_words = ["python", "pythoner", "pythoning", "pythoned"]
```

```
output = ["python", "python", "python", "python"]
```

4. **Part of Speech Tagging** – This means labeling words in a sentence as nouns, adjectives, verbs...etc. Even more impressive, it also labels by tense, and more.

List of Speech Tags:

CC	coordinating conjunction
JJ	adjective 'big'
MD	modal could, will
NN	noun, singular 'desk'
NNS	noun plural 'desks'
NNP	proper noun, singular 'Harrison'
NNPS	proper noun, plural 'Americans'
PDT	predeterminer 'all the kids'
RBS	adverb, superlative best
RP	particle give up
TO	to go 'to' the store.
UH	interjection errrrrrrm
VB	verb, base form take
VBD	verb, past tense took
WRB	wh-abverb where, when

```
Input Text = ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The',  
'weather', 'is', 'great', ',', 'and', 'Python', 'is', 'awesome', '.']
```

Output = [('Hello', 'NN')] [('Mr', 'NNP'), (',', ',')] [('Smith', 'NNP')] [('(', '(', ')')] [('how', 'WRB')] [('are', 'VBP')] [('you', 'PRP')] [('doing', 'VBG')] [('today', 'NN')] [('?', ',')] [('The', 'DT')] [('weather', 'NN')] [('is', 'VBZ')] [('great', 'JJ')] [('(', '(', ')')] [('and', 'CC')] [('Python', 'NN')] [('is', 'VBZ')] [('awesome', 'NN')] [('(', '(', ')')]

### 3.2.3.2 Google Dialogflow

Dialogflow is a natural language understanding platform developed by Google, which is used to design and integrate a conversational user interface into applications.

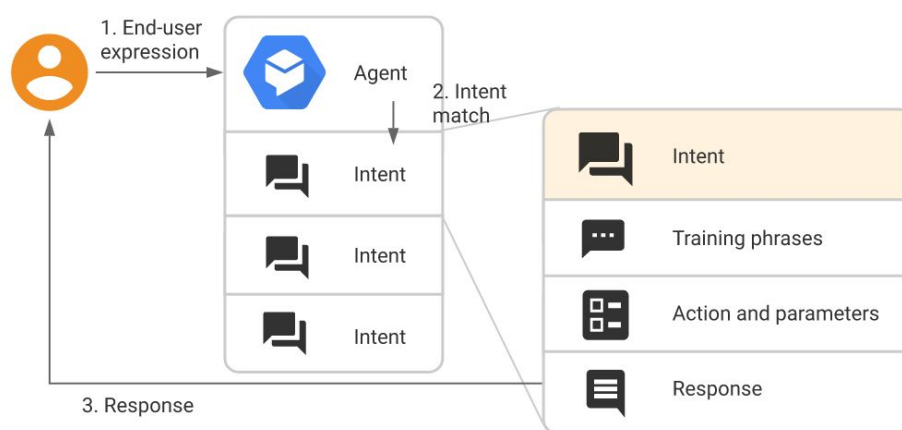


Fig 3.9 Dialogflow flowchart

An intent categorizes an end-user's intention for one conversation turn. Dialogflow matches the end-user expression to the best intent in the agent. Matching an intent is also known as intent classification.

**Training phrases:** These are example phrases for what end-users might say. When an end-user expression resembles one of these phrases, Dialogflow matches the intent.

**Responses:** You define text, speech, or visual responses to return to the end-user.

These may provide the end-user with answers, ask the end-user for more information, or terminate the conversation.

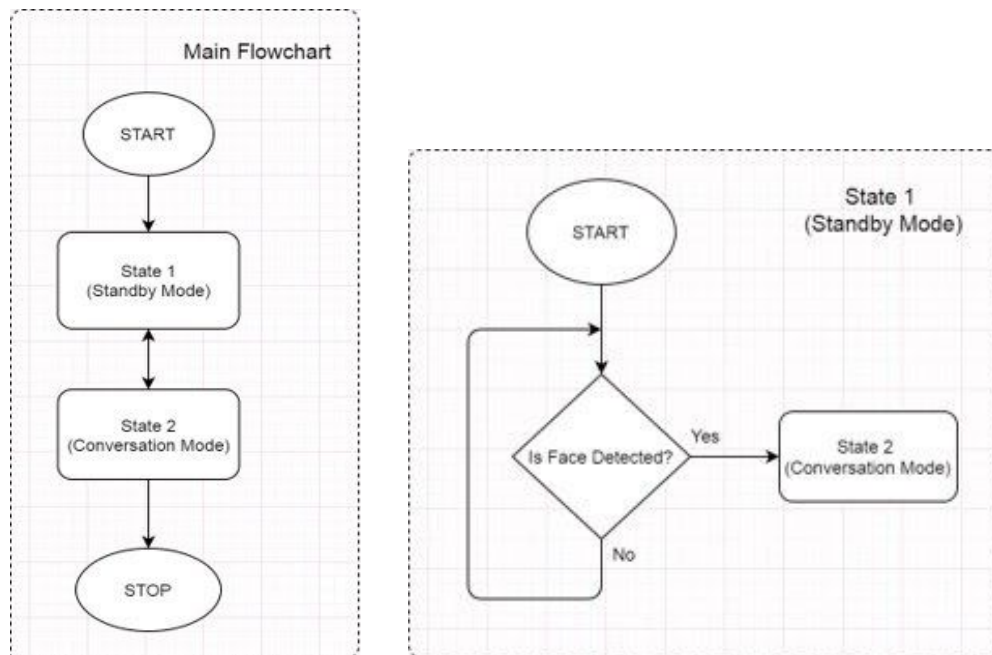
## CHAPTER 4:

### IMPLEMENTATION DETAILS

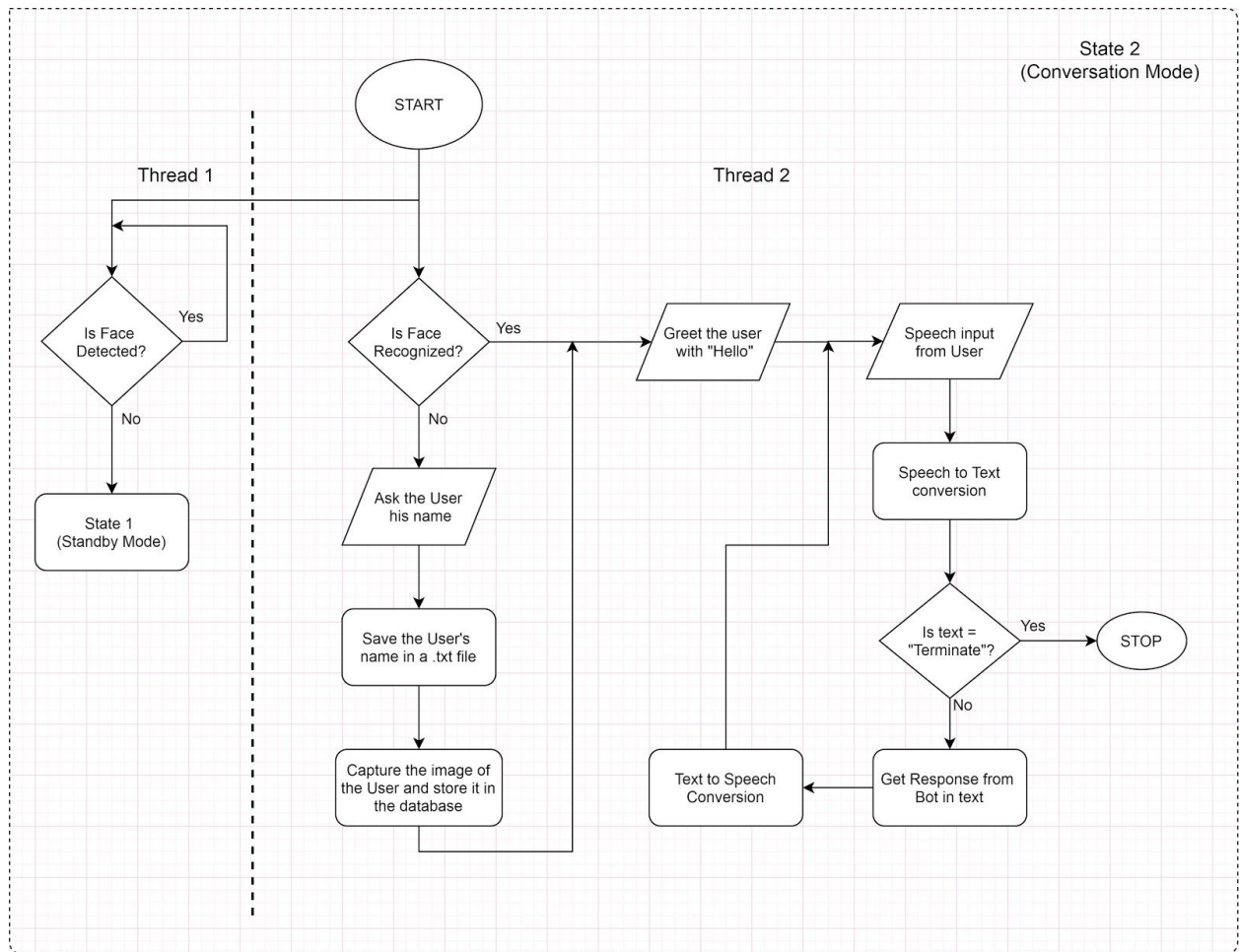
The code for this project was written entirely in **Python** programming language.

The Bot has 2 states. The first state is called **Standby Mode**. In this state, there is no person in front of the bot and the bot is waiting for a person to come in front of it.

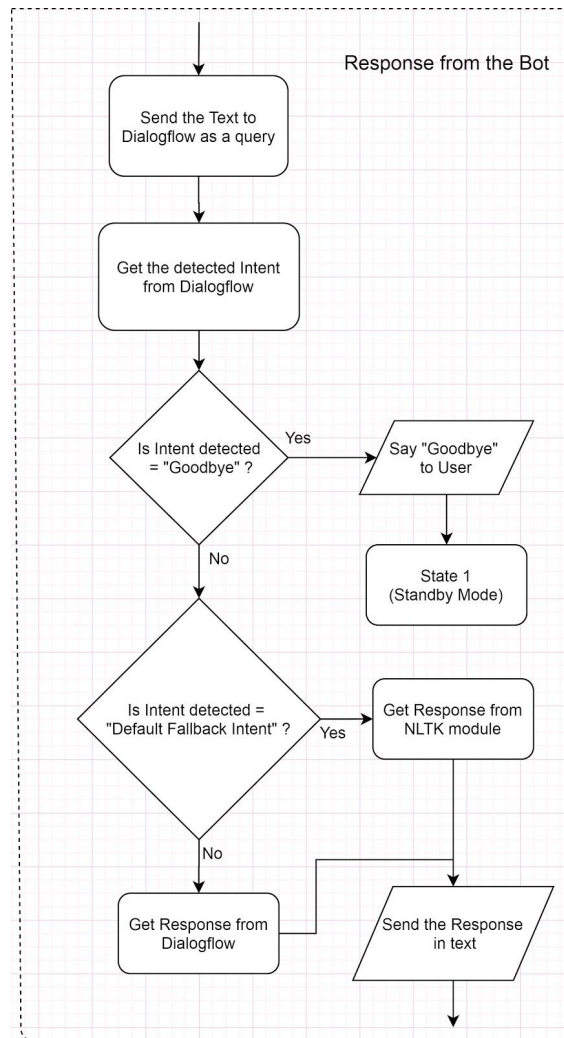
The second state is called **Conversation Mode**. In this state, there is a person in front of the bot and the bot converses with that person until the person either says Goodbye or he moves away from the screen. The reason for using the concept of states is because it makes the flow of the program error free and makes programming easier.



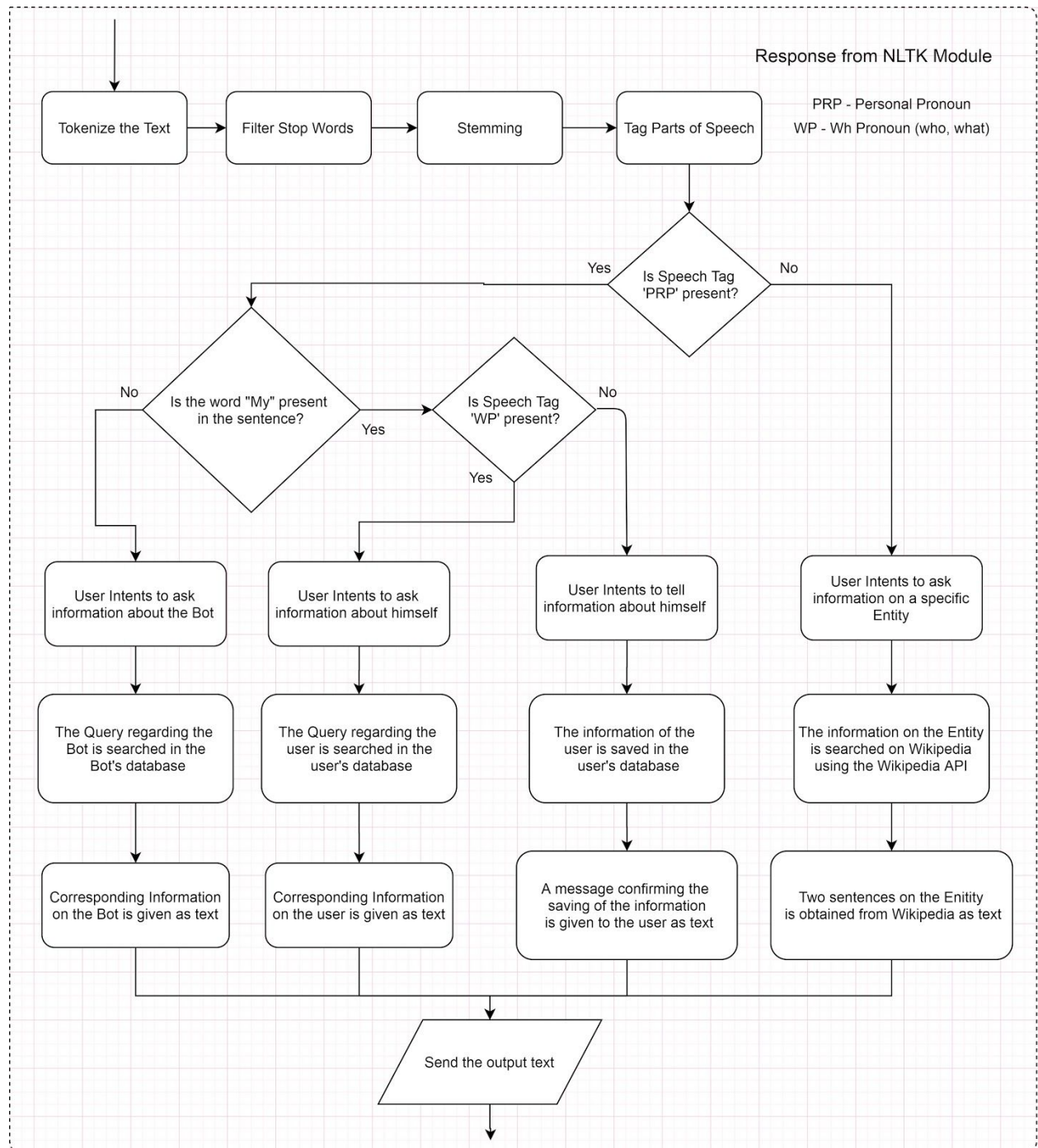
The Program starts with the Bot being in state 1 - standby mode. In state 1, the bot continuously monitors whether there is any face in front of the camera. Once the bot detects a face, it then goes to State 2 - conversation mode.



The state 2 - conversation mode is run on 2 threads. The first thread performs face detection. This thread continuously checks whether there is a face in front of the bot. The second thread is used for conversation. Once the bot enters state 2, it performs face recognition operation on the detected face. If the face is not a match with the faces in the databases, then the bot asks the name of the person and saves the new face in the database with the name. Then the bot greets the user and the conversation pipeline takes place in a loop. If the user utters “Terminate”, then the program ends.



The response from Bot is a combination of Dialogflow and NLTK. Our Dialogflow agent has been trained on Small Talk. Our dialogflow agent can give a response for small talk intention. But if any sentence from the other 4 intentions is fed as input query to the agent, it triggers “Default Feedback Intent”. If Default Feedback Intent is detected then we go on with NLTK processing. If not, then we output the dialogflow response text to the Text to Speech converter. We have created an intent in our dialogflow agent as “Goodbye” and have trained it for when the user wants to leave. If Goodbye intent is detected, we go to State 1.



First we tokenize the input text, then we filter the stop words, then we apply stemming to the words and then we tag parts of speech. Based on the tags present in the speech, we find out the remaining 4 intentions of the user.



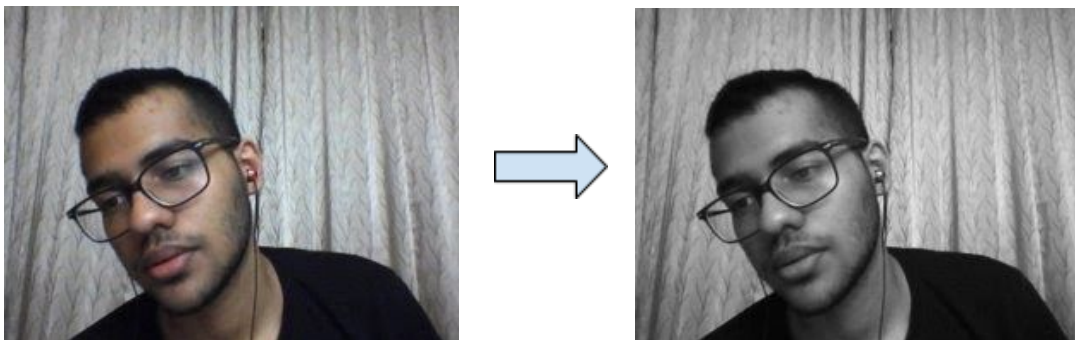
## CHAPTER 5:

## RESULTS

### 5.1 FACE RECOGNITION

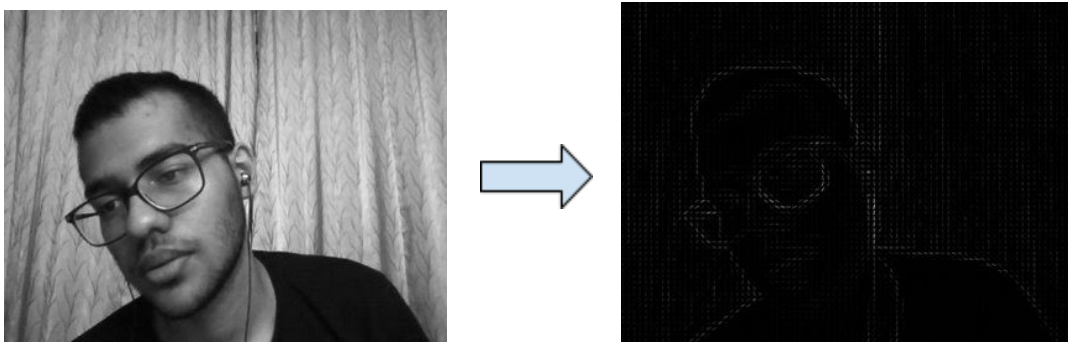
#### Face Detection:

Converting RGB image into Grey scale image



*Fig 5.1 RGB image to Grey scale image conversion*

HOG image of the person



*Fig 5.2 HOG visualization*

## Face Alignment:

Face Landmark Estimation feature points on the face



*Fig 5.3 68 Landmarks found on input image*

Affine Transformation on the face using the 68 face landmarks



*Fig 5.4 Affine transformation on our input image*

## Face Encodings obtained from CNN

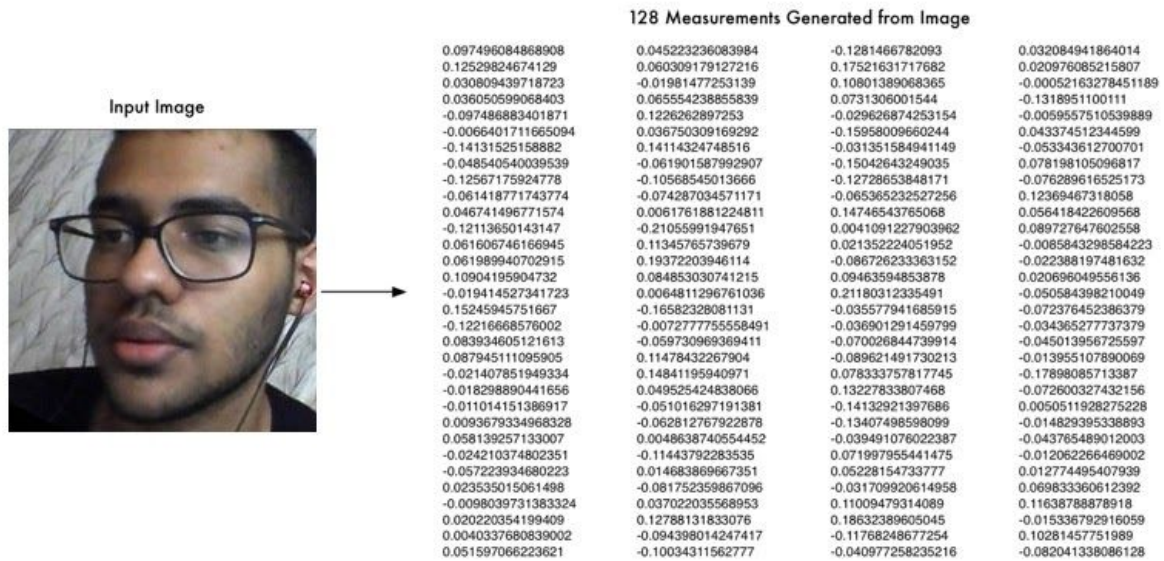


Fig 5.5 CNN output of 128 dimension vector

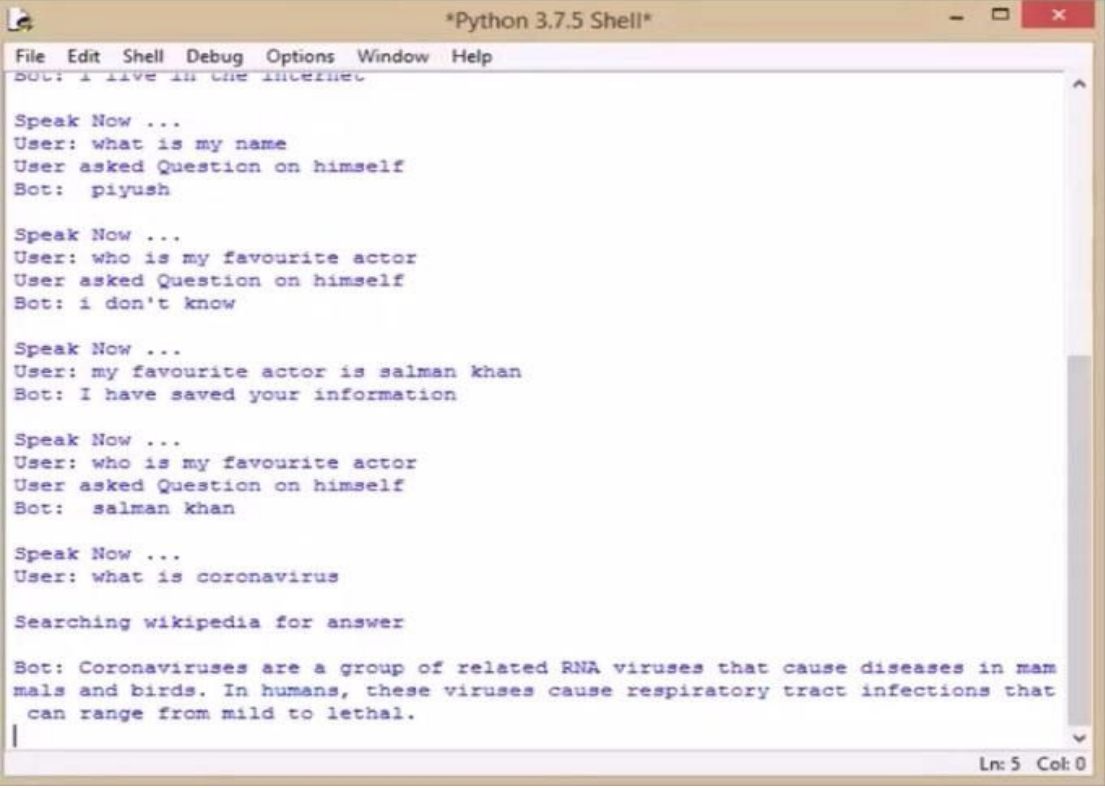
Face Recognition output with box around the detected face and name of person.



Fig 5.6 The Face Recognition output video stream

## 5.2 Natural Language Processing (NLP)

Conversation in Speech was achieved. Every conversation piece is written in text for display.



```
*Python 3.7.5 Shell*
File Edit Shell Debug Options Window Help
Bot: I live in the internet

Speak Now ...
User: what is my name
User asked Question on himself
Bot: piyush

Speak Now ...
User: who is my favourite actor
User asked Question on himself
Bot: i don't know

Speak Now ...
User: my favourite actor is salman khan
Bot: I have saved your information

Speak Now ...
User: who is my favourite actor
User asked Question on himself
Bot: salman khan

Speak Now ...
User: what is coronavirus

Searching wikipedia for answer

Bot: Coronaviruses are a group of related RNA viruses that cause diseases in mam
mals and birds. In humans, these viruses cause respiratory tract infections that
can range from mild to lethal.
|
Ln: 5 Col: 0
```

*Fig 5.7 The conversation log between the Bot and the User in text*

## **CHAPTER 6:**

### **CONCLUSION**

This Application can successfully recognize people through face recognition and can converse with them in speech in English language.

By deploying the Face Detection module in parallel to the NLP module makes the overall functioning of the Bot more stable and fast.

This Application can be interfaced with Humanoid Robots and Androids for a more realistic conversational experience.

The Application can be improved by adding more functionalities such as setting an Alarm, scheduling events in Calendar, etc.

## **CHAPTER 7:**

### **REFERENCES**

- [1] Navneet Dalal and Bill Triggs, “Histograms of Oriented Gradients for Human Detection”
- [2] Vahid Kazemi and Josephine Sullivan “One Millisecond Face Alignment with an Ensemble of Regression Trees”
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun “Deep Residual Learning for Image Recognition”
- [4] Jessel Serrano, Fernando Gonzalez, Janusz Zalewski “CleverNAO: The Intelligent Conversational Humanoid Robot ”
- [5] Dylan F. Glas, Takashi Minato, Carlos T. Ishi, Tatsuya Kawahara, “ERICA: The ERATO Intelligent Conversational Android ”
- [6] Aditya Pradana<sup>1</sup> , Goh Ong Sing and Yogan Jaya Kumar, “SamBot - Intelligent Conversational Bot for Interactive Marketing with Consumer-centric Approach”