Towards partial fulfilment for Undergraduate Degree level Programmed

Bachelor of Technology in Computer Engineering

A Third Project Evaluation Report on:

Emotion Recognition Using Facial Features

Prepared by:

Admission No.	Student Name
U13CO014	**Riya Kothari**
U13CO018	**Vidushi Tyagi**
U13CO025	**Heli Mistry**
U13CO044	**Divya Sankhe**

Class	:	B.TECH. IV (Computer Engineering) 8th Semester
Year	:	2017-2018
Guided by	:	**Prof. M. A. Zaveri**

DEPARTMENT OF COMPUTER ENGINEERING
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY,
SURAT - 395 007 (GUJARAT, INDIA)

Student Declaration

This is to certify that the work described in this project report has been actually carried out and implemented by our project team consisting of

Sr.	**Admission No.**	**Student Name**
1	U13CO014	RIYA KOTHARI
2	U13CO018	VIDUSHI TYAGI
3	U13CO025	HELI MISTRY
4	U13CO044	DIVYA SANKHE

Neither the source code there in, nor the content of the project report have been copied or downloaded from any other source. We understand that our result grades would be revoked if later it is found to be so.

Signature of the Students:

Sr.	**Student Name**	**Signature of the Student**
1	RIYA KOTHARI	
2	VIDUSHI TYAGI	
3	HELI MISTRY	
4	DIVYA SANKHE	

Certificate

This is to certify that the project report entitled **Emotion Recognition Using Facial Features** is prepared and presented by

Sr.	**Admission No.**	**Student Name**
1	U13CO014	RIYA KOTHARI
2	U13CO018	VIDUSHI TYAGI
3	U13CO025	HELI MISTRY
4	U13CO044	DIVYA SANKHE

Final Year of Computer Engineering and their work is satisfactory.

SIGNATURE :

GUIDE	JURY	HEAD OF DEPT.

Abstract

Extracting and validating emotional cues through analysis of users' facial expressions is of high importance for improving the level of interaction in human-machine communication systems. Extraction of appropriate facial features and consequent recognition of the user's emotional state can be used in areas of security, entertainment and human machine interface (HMI).

This project presents an application that detects a human face and derives the emotion from the facial features using machine learning algorithms. This kind of emotion recognition system can have various applications in real time domains.

***Keywords**: face detection, feature extraction, face classification, local binary pattern, Viola-Jones, facial landmarks, fisherface, svm*

INDEX

1	Introduction		1
	1.1	Applications	1
	1.2	Motivation	2
	1.3	Objectives	3
	1.4	Challenges	3
2	Literature Survey		5
	2.1	Literature Survey	5
		2.1.1 Face Detection Methods	5
		2.1.2 Feature Extraction Methods	8
		2.1.3 Classification	10
	2.2	Characteristics of Emotion	13
3	Proposed Algorithms		15
	3.1	Local Binary Patterns	15
	3.2	Cascade of “weak-classifiers” using Haar features	17
		3.2.1 Features	18
		3.2.2 Integral Image	19
		3.2.3 Learning Classification Functions	19
		3.2.4 The Additional Cascade	20
4	Platform and Libraries		21
	4.1	Platform and libraries used	21
	4.2	OpenCV	21
	4.3	Android Studio	21
5	Implementation and Application: Phase I		22
	5.1	Application Overview	22
	5.2	Feature Extraction	23
	5.3	Applications	25
	5.4	Screenshots	26
6	Implementation: Phase II		30
	6.1	Theory	30
	6.2	Libraries used	31
	6.3	Procedure	31
	6.4	Code Snippet	32

6.5 Observation 37
6.6 Screenshots 38
7 Implementation: Phase III 40
7.1 Theory 40
7.2 Libraries used 41
7.3 Procedure 42
7.4 Code Snippet 43
7.5 Observation 45
7.6 Screenshots 46
7.7 Comparison 49
8 Conclusions and Future work 50
9 References 51

List of Figures

1	A generic Face Recognition System	5
2	Categorization of face detection techniques	8
3	Similarity-based classifiers	11
4	Probabilistic classifiers	12
5	Classifiers using decision boundaries	12
6	Different Emotions	13
7	The basic LBP operator	15
8	Examples of texture primitives	16
9	Example of rectangle features	18
10	Rectangular sum in integral image	19
11	Features selected by AdaBoost	20
12	Schematic depiction of the detection cascade	20
13	An overview of a system using emotion recognition	22
14	Flow diagram of feature extraction implementation	24
15	Emotion Recognition icon on the device screen	26
16	Application start page	27
17	Selecting input mode	28
18	Real-time feature detection of the camera feed	29
19	Neutral and angry face detection of two subjects	38
20	Surprised, angry and neutral face detection of four subjects	38
21	Disgust, happy and sad face detection of four subjects	39
22	Facial landmarks of two subjects	46

Chapter 1 - Introduction

Development of efficient visual feature extracting algorithms and high processing power for retrieval from a huge image database is difficult as image is a complex high dimension (3D) matrix and processing matrix operation is not so fast and perfect. Hence, we focus on the new algorithms which are more real-time and more efficient with maximum percentage of accuracy. Efficient and effective recognition of human face from image databases is now a requirement

Detecting emotion of human can be achieved by using facial image, voice, body shape, etc... Among them, the facial image is the most frequent source to detect emotion. Face recognition is also one of the most successful applications of image analysis and understanding.

A very important requirement for facial expression recognition is that all processes therein have to be performed without or with the least possible user intervention. This typically involves initial detection of face, extraction and tracking of relevant facial information, and facial expression classification. In this framework, actual implementation and integration details are enforced by the particular application.

In this report our main focus is on studying face detection and expression recognition techniques. We believe the results of this kind of study would help in creating attractive facial appearance and thus have wide applicability in art, gaming, and messaging applications in which a pleasing degree of realism is desirable without exaggerated comedy or caricature.

1.1 Applications

The possible applications of an interface capable of assessing human emotional states are numerous. One of the uses of such an interface is to enhance human judgement of emotion in situations where objectivity and accuracy are required.

1. **Medicine**: It is used in counselling to determine patients emotion state as well as patients feeling about the treatment. It can also be used in developing a therapy for patients having Autism.

   Another example is clinical studies of schizophrenia and particularly the diagnosis of flattened affect that so far relies on the psychiatrists' subjective judgement of subjects' emotionality based on various physiological clues. . An automatic emotion-sensitive system could augment these judgements, so minimising the dependence of the diagnostic procedure on individual psychiatrists' perception of emotionality.

2. **E-Learning**: It helps the online tutor to adjust the presentation style according to the student by detecting the state of the learner. An emotion sensitive automatic tutor can interactively adjust the content of the tutorial and the speed at which it is delivered based on whether the user finds it boring and dreary or exciting and thrilling or even unapproachable and daunting.
3. **Monitoring**:

   a) It can be used to detect the state of the car driver and alert other cars. Emotion responsive car that can alert the driver when it detects signs of stress or anger that could impair their driving abilities

   b) While using an ATM machine, it can detect the emotion of the user and not dispense the money if the user is scared.

4. **Entertainment**: The most obvious commercial application of emotion sensitive systems is the game and entertainment industry with either interactive games that offer the sensation of naturalistic human-like interaction, or pets, dolls and so on that are sensitive to the owner's mood and can respond accordingly
5. **Marketing**: it is used to evaluate the impact of the ads by measuring the attention and engagement. Emotions are vital in the purchasing decision.
6. **Lie detection**
7. **Emotion-sensitivity** can be added to automatic customer services, call centres or personal assistants, for example, to help detect frustration and avoid further irritation, with the options to pass the interaction over to a human, or even terminate it altogether.

1.2 Motivation

Humans detect and identify faces in a scene with little or no effort. However, building an automated system that accomplishes this task is very difficult. The human face is a dynamic object and has a high degree of variability in its appearance, which makes face detection a difficult problem in computer vision.Facial features and expressions are critical to everyday communication. Social psychology research has shown that conveying messages in meaningful conversations can be dominated by facial expressions, and not spoken words.

Increased use of computing technology in communication requires improved human-computer interaction. Thus for better interaction amongst people we need systems for detecting those emotions. Also, these systems are helpful to a user in determining the opposite user’s emotion and thus helping them determine their path of action. So we want to create an application that would recognize the emotion and subsequently suggest the user some path they might choose to assuage or ameliorate the situation

1.3 Objectives

The objectives of this research project are:
1)To study various parameters for detecting face from a given image
2)To pick the facial features and recognizing the emotion
3)To style the captured image to create appealing cartoonized image

The used classification methods are evaluated in terms of the level of efficiency.

1.4 Challenges

1. Light - “the variations between the images of the same face due to illumination and viewing direction are almost always larger than image variations due to change in face identity”.

   Face recognition must contend with uncontrolled lighting conditions, large pose variations, facial expressions, makeup, changes in facial hair, ageing and partial occlusions without forgetting that the human face is not a unique rigid object.

   For decades, geometric feature based methods have used properties and relations between facial features such as eyes, mouth, nose, and chin to perform recognition. Despite their economical representation and their insensitivity to small variations in illumination and viewpoint, feature-based methods are quite sensitive to the feature extraction and measurement process. It has been argued that existing techniques for the extraction and measurement of facial features are not reliable enough.

   Natural outdoor lighting has proven difficult, not simply because of the strong shadows cast by a light source such as the sun, but also because subjects tend to distort their faces when illuminated by a strong source.

2. Pose/viewpoint - The images of a face vary because of the relative camera face pose and some facial features such as the eyes or nose may become partially or wholly occluded.

3. Occlusion - Faces may be partially occluded by other objects. In an image with a group of people, some faces or other objects may partially occlude other faces, which in turn results in only a small part of the face is available in many situations.

4. Real Emotions – Also the training set of images of different emotions are generally not real i.e. they are recorded by some actor who is not feeling the same emotion that he/she is portraying. So it is plausible that the images for training might be not accurate.

Chapter 2 - Literature Survey

A study on the theory of various techniques and methods involved in the emotion recognition process is reviewed in this chapter.

2.1 Literature Survey

Facial expression recognition involves three steps face detection, feature extraction and expression classification.

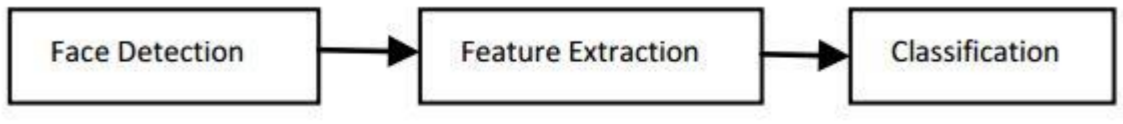


Figure 1: A generic Face Recognition System [4]

2.1.1 Face Detection Methods

The first step in face detection is Pre-processing. Steps involved in converting a image to a normalized pure facial image for feature extraction is detecting feature points, rotating to line up, locating and cropping the face region using a rectangle, according to the face model.

Knowledge based: These rule-based methods [1] encode human knowledge of what constitutes a typical face. Usually, the rules capture the relationships between facial features. These methods are designed mainly for face localization.

Feature based: These algorithms [1] aim to find structural features that exist even when the pose, viewpoint, or lighting conditions vary, and then use these to locate faces. These methods are designed mainly for face localization.These algorithmsdetects the feature points from the image using spatial filtering techniques and then groups the points into face candidates using perceptual grouping principles and selects the true candidate using probabilistic frame work.

a) **Texture based**: Human faces have a distinct texture that can be used to separate them from different objects. Emotion recognition is based on the texture feature extracted from the gray-level co-occurrence matrix (GLCM)

,the extracted GLCM features are trained with Support Vector Machine with different kernels [4].

b) **Skin Colour based**: Three colour spaces RGB, YCbCr and HSI are used to compare algorithms. These colors are combined to get a new skin color based on face detection algorithm. After getting skin region, facial features eyes and mouth are extracted by darkening the background colors and then the image is transformed to gray scale and binary image by suitable threshold [4].

c) **Multiple features**: Numerous methods that combine several facial features have been proposed to locate or detect faces. Most of them utilize global features such as skin colour, size, and shape to find face candidates, and then verify these candidates using local, detailed features such as eye brows, nose, and hair. A typical approach [1] begins with the detection of skin-like regions. Next, skin-like pixels are grouped together using connected component analysis or clustering algorithms. If the shape of a connected region has an elliptic or oval shape, it becomes a face candidate. Finally, local features are used for verification.

Template Matching: Several standard patterns of a face are stored to describe the face as a whole or the facial features separately. The correlations between an input image and the stored patterns are computed for detection. These methods have been used for both face localization and detection.

a) **Predefined Templates:** Several sub-templates for the eyes, nose, mouth, and face contour to model a face are used by an algorithm [12]. Each sub-template is defined in terms of line segments. Lines in the input image are extracted based on greatest gradient change and then matched against the sub-templates. The correlations between sub-images and contour templates are computed first to detect candidate locations of faces. Then, matching with the other sub-templates is performed at the candidate positions. In other words, the first phase determines focus of attention or region of interest and the second phase examines the details to determine the existence of a face.
b) **Temporal Templates:** Yuille et al. used deformable templates to model facial features that fit an a-priori elastic model to facial features (e.g., eyes) [1]. In

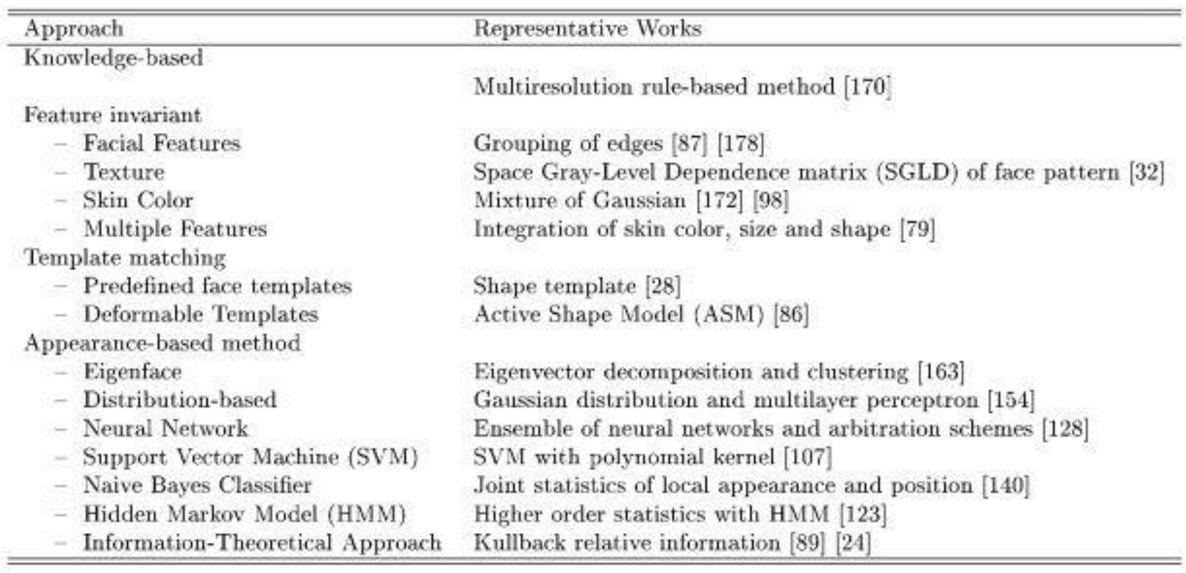
this approach, facial features are described by parameterized templates. An energy function is defined to link edges, peaks, and valleys in the input image to corresponding parameters in the template. The best fit of the elastic model is found by minimizing an energy function of the parameters.

Appearance based: In contrast to template matching, the models (or templates) are learned from a set of training images which should capture the representative variability of facial appearance. In general, appearance-based methods rely on techniques from statistical analysis and machine learning to find the relevant characteristics of face and non-face images. The learned characteristics are in the form of distribution models or discriminant functions that are consequently used for face detection. Meanwhile, dimensionality reduction is usually carried out for the sake of computation efficiency and detection efficacy.

Distribution Features: The images were acquired and five significant portions are cropped from the image which is performed for extraction and thus store Eigen vectors specific to the expressions. Eigen vectors are computed and input facial image is recognized when similarity was obtained, the Euclidean distance between the test image and different expression is calculated. The classification is done with Support Vector Machine.

Neural Networks: Neural networks have been applied successfully in many pattern recognition problems, such as optical character recognition, object recognition, and autonomous robot driving. Since face detection can be treated as a two class pattern recognition problem, various neural network architectures have been proposed. The advantage of using neural networks for face detection is the feasibility of training a system to capture the complex class conditional density of face patterns. However, one drawback is that the network architecture has to be extensively tuned (number of layers, number of nodes, learning rates, etc.) to get exceptional performance.

For more relevant methods and tools, refer [2].



Approach	Representative Works
Knowledge-based	
	Multiresolution rule-based method [170]
Feature invariant	
– Facial Features	Grouping of edges [87] [178]
– Texture	Space Gray-Level Dependence matrix (SGLD) of face pattern [32]
– Skin Color	Mixture of Gaussian [172] [98]
– Multiple Features	Integration of skin color, size and shape [79]
Template matching	
– Predefined face templates	Shape template [28]
– Deformable Templates	Active Shape Model (ASM) [86]
Appearance-based method	
– Eigenface	Eigenvector decomposition and clustering [163]
– Distribution-based	Gaussian distribution and multilayer perceptron [154]
– Neural Network	Ensemble of neural networks and arbitration schemes [128]
– Support Vector Machine (SVM)	SVM with polynomial kernel [107]
– Naive Bayes Classifier	Joint statistics of local appearance and position [140]
– Hidden Markov Model (HMM)	Higher order statistics with HMM [123]
– Information-Theoretical Approach	Kullback relative information [89] [24]

Figure 2: Categorization of face detection techniques [1]

2.1.2 Feature Extraction Methods

The feature extraction process can be defined as the procedure of extracting relevant information from a face image. This information must be valuable to the later step of identifying the subject with an acceptable error rate. The feature extraction process must be efficient in terms of computing time and memory usage. The output should also be optimized for the classification step. Following are some of the known methods for feature extraction.

Feature Extraction using Discrete Cosine Transform (DCT): Discrete Cosine Transform (DCT) is applied on input image (img).The DCT coefficient are kept in Zigzag positions and these are implemented to convert 2D DCT image matrix to feature vector, here domain frequency component is kept at the start vector. The features like eyes and mouth are extracted from input image and, DCT is applied on this sub image to get DCT coefficient of eyes and mouth region, finally these features are added with feature vector. AdaBoost is used as a classifier for emotion recognition. The recognition rate is 75.94%.

Facial Feature Extraction using Gabor Filter: The facial images are pre-processed based on affine transform to normalize the faces, the evaluation is done with separability of different Gabor filters, and these filters are used to separate different expression. Dimension reduction achieved by Principle Component Analysis (PCA) and Feasibility of Linear Discriminant Analysis(FLDA) multi-classifier are used for recognition. The classifier assumes the discriminate function to be a linear function of the feature data. In this case the data is the feature vector obtained. According to the author the Gabor filter selection reduce the dimension of feature space and computation complexity. The expression database used here to examine recognition system is the JAFFE database and the recognition rate is 93% and above.

Facial Feature Extraction using Principle Component Analysis: Weighted Principal Component Analysis (WPCA) method is used on multi feature fusion. These high dimensional local self-related features are extracted from facial expression images, thus the images are divided into regions. Then WPCA is used for dimension reduction, the weights is quickly determined based on facial action coding system, also Radial Basis Function(RBF) is used. Support Vector Machine algorithm is applied for classification of facial expression characteristics. Finally Euclidean distance is calculated to obtain the similarity between templates and then facial expression recognition is done with nearest algorithm. The recognition rate is 88.25% in this method.

Facial Feature Extraction using Independent Component Analysis: Independent Component Analysis (ICA) derived from the principle of optimal information transfer through sigmodal neurons. ICA is used to perform on face images in the FERET database under two different architectures. The first architecture takes images as random variables and pixels as random trials, hence ICA consider images statistically independent images these images are sparse and localized in space resembling facial features. The second architecture takes pixels as random variables and images as random trials, here image coefficient are approximately independent and thus give result in factorial face code .Finally the two ICA representation are combined in single classifier. In this paper sources was controlled by dimension reduction through Principal Component Analysis and information maximization algorithm is used for performance of ICA. The overall classification performance is 99.8% with 400 out of 500 test images.

Facial Feature Extraction using Linear Discriminant Analysis: In this paper [13] more efficient, accurate and stable method is proposed to calculate the discriminant vectors based on Fisher's criterion and here a two stage procedure is implemented. In the first stage the homogeneous regions of a face image are grouped as same partition based on geometric characteristics, then the mean gray value is used for pixels within the partition region and thus the face image is reduced as feature vector .In the second stage the feature vector is used to determine discriminant projection axes based on the proposed LDA. The vectors are clustered using K-means clustering method with each changed samples. The performance in the proposed method is 91% [4].

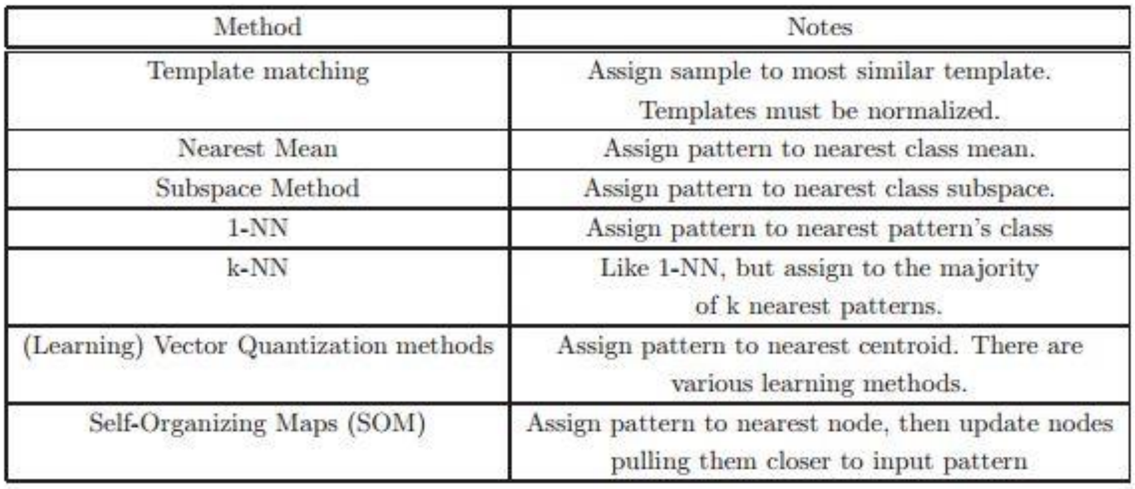
2.1.3 Classification

Once the features are extracted and selected, the next step is to classify the image. Appearance-based emotion recognition algorithms use a wide variety of classification methods. Sometimes two or more classifiers are combined to achieve better results. On the other hand, most model-based algorithms match the samples with the model or template. Then, a learning method is can be used to improve the algorithm. One way or another, classifiers have a big impact in emotion recognition.

Classification algorithms usually involve some learning - supervised, unsupervised or semi-supervised. Unsupervised learning is the most difficult approach, as there are no tagged examples. However, many emotion recognition applications include a tagged set of subjects. Consequently, most face recognition systems implement supervised learning methods. There are also cases where the labelled data set is small. Sometimes, the acquisition of new tagged samples can be infeasible. Therefore, semi-supervised learning is required.

According to Jain, Duin and Mao [5], there are three concepts that are key in building a classifier - similarity, probability and decision boundaries. We will present the classifiers from that point of view.

a) **Similarity** - This approach is intuitive and simple. Patterns that are similar should belong to the same class. This approach has been used in the face recognition algorithms implemented later. The idea is to establish a metric that de- fines similarity and a representation of the same-class samples. For example, the metric can be the Euclidean distance. The representation of a class can be the mean vector of all the patterns belonging to this class. The 1-NN decision rule can be used with this parameters. It’s classification performance is usually good. This approach is similar to a k-means clustering algorithm in unsupervised learning. There are other techniques that can be used. For example, Vector Quantization, Learning Vector Quantization or Self-Organizing Maps, see table below.

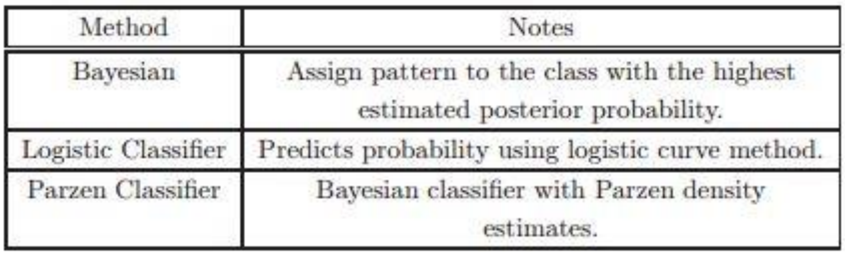


Method	Notes
Template matching	Assign sample to most similar template. Templates must be normalized.
Nearest Mean	Assign pattern to nearest class mean.
Subspace Method	Assign pattern to nearest class subspace.
1-NN	Assign pattern to nearest pattern's class
k-NN	Like 1-NN, but assign to the majority of k nearest patterns.
(Learning) Vector Quantization methods	Assign pattern to nearest centroid. There are various learning methods.
Self-Organizing Maps (SOM)	Assign pattern to nearest node, then update nodes pulling them closer to input pattern

Figure 3: Similarity-based classifiers [4]

b) **Probability** - Some classifiers are built based on a probabilistic approach. Bayes decision rule is often used. The rule can be modified to take into account different factors that could lead to miss-classification. Bayesian decision rules can give an optimal classifier, and the Bayes error can be the best criterion to evaluate features. Therefore, a posteriori probability functions can be optimal.

This algorithms estimate the densities instead of using the true densities. These density estimates can be either parametric or nonparametric. Commonly used parametric models in face recognition are multivariate Gaussian distributions, as in [6]. Two well-known non-parametric estimates are the k-NN rule and the Parzen classifier. They both have one parameter to be set, the number of neighbour k, or the smoothing parameter (bandwidth) of the Parzen kernel, both of which can be optimized. Moreover, both these classi- fiers require the computation of the distances between a test pattern and all the samples in the training set. These large numbers of computations can be avoided by vector quantization techniques, branch-and-bound and other techniques. A summary of classifiers with a probabilistic approach can be seen in table below.

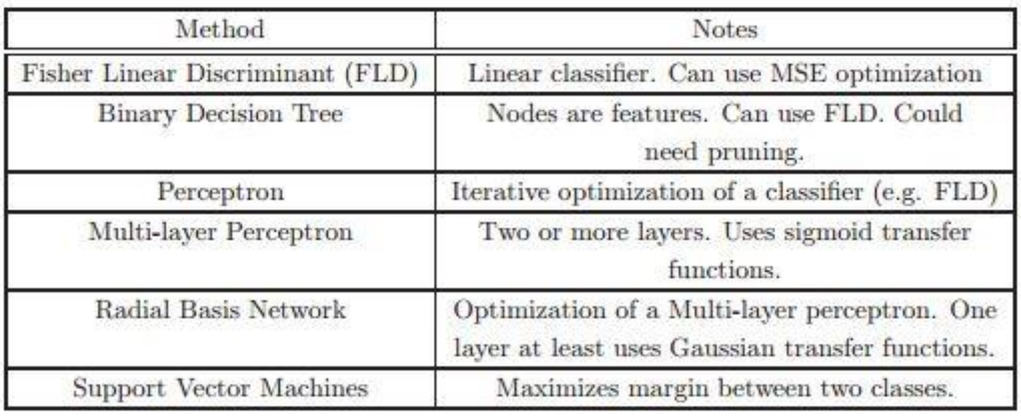


Method	Notes
Bayesian	Assign pattern to the class with the highest estimated posterior probability.
Logistic Classifier	Predicts probability using logistic curve method.
Parzen Classifier	Bayesian classifier with Parzen density estimates.

Figure 4: Probabilistic classifiers [4]

c) **Decision boundaries -** This approach can become equivalent to a Bayesian classifier. It depends on the chosen metric. The main idea behind this approach is to minimize a criterion (a measurement of error) between the candidate pattern and the testing patterns. One example is the Fisher's Linear Discriminant (often FLD and LDA are used interchangeably). It's closely related to PCA. FLD attempts to model the difference between the classes of data, and can be used to minimize the mean square error or the mean absolute error. Other algorithms use neural networks. Multilayer perceptron is one of them. They allow nonlinear decision boundaries. However, neural networks can be trained in many different ways, so they can lead to diverse classifiers.

A special type of classifier is the decision tree. It is trained by an iterative selection of individual features that are most salient at each node of the tree. During classification, just the needed features for classification are evaluated, so feature selection is implicitly built-in. The decision boundary is built iteratively.



Method	Notes
Fisher Linear Discriminant (FLD)	Linear classifier. Can use MSE optimization
Binary Decision Tree	Nodes are features. Can use FLD. Could need pruning.
Perceptron	Iterative optimization of a classifier (e.g. FLD)
Multi-layer Perceptron	Two or more layers. Uses sigmoid transfer functions.
Radial Basis Network	Optimization of a Multi-layer perceptron. One layer at least uses Gaussian transfer functions.
Support Vector Machines	Maximizes margin between two classes.

Figure 5: Classifiers using decision boundaries [4]

2.2 Characteristics of Emotion

Each and every emotion has some characteristics which are identified by the relative orientation of eyes, lips, eyebrows, eyelids etc.… These orientations for the seven basic emotions are specified below.

1. **Anger**
   - **a.** Eyebrows are pulled down
   - **b.** Upper lids are pulled up
   - **c.** Lower lids are pulled up
   - **d.** Lips may be tightened.

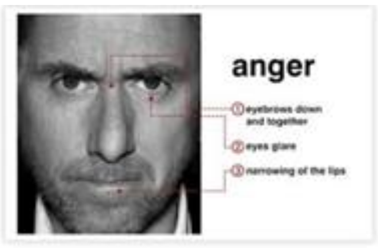


Figure 6(a): Anger [3]

2. **Happiness**
   - **a.** Muscle around the eyes are tightened
   - **b.** Crow’s feet wrinkles appears around eyes
   - **c.** Cheeks are raised
   - **d.** Lip corners are raised diagonally

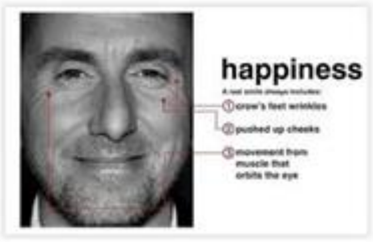

happiness
A real smile always includes:
① crow's feet wrinkles
② pushed up cheeks
③ movement from muscle that orbits the eye

Figure 6(b): Happiness [3]

3. **Sadness**
   - **a.** Inner corner of eyebrows are raised
   - **b.** Eyelids are loose
   - **c.** Lip corners are pulled down.

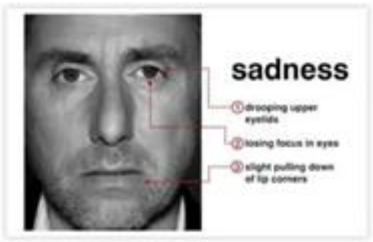


Figure 6(c): Sadness [3]

4. **Fear**
   - a. Eyebrows are pulled up and together
   - b. Upper eyelids are pulled up and
   - c. Mouth is stretched

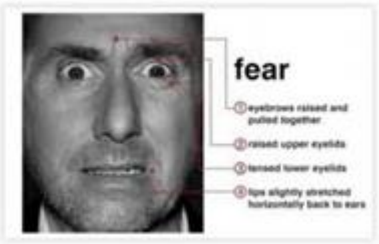

fear
① eyebrows raised and pulled together
② raised upper eyelids
③ tensed lower eyelids
④ lips slightly stretched horizontally back to ears

Figure 6(d): Fear [3]

5. **Disgust**
    a. Eyebrows are pulled down
    b. Nose is wrinkled
    c. Upper lip is pulled up

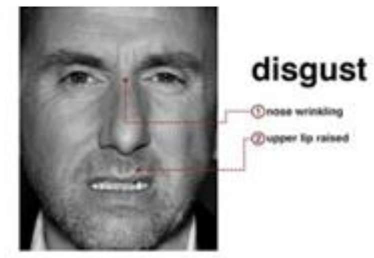


Figure 6(e): Disgust [3]

6. **Surprise**
    a. Entire eyebrows are pulled up
    b. Eyelids are also pulled up
    c. Mouth is widely open.

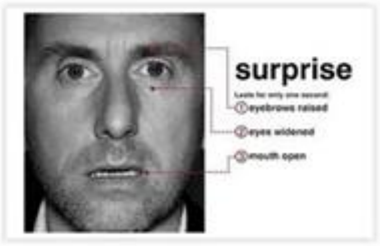


Figure 6(f): Surprise [3]

7. **Contempt**
    a. Lip corner tightened and raised on only one side of face

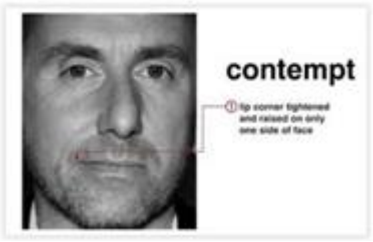


Figure 6(g): Contempt [3]

Figure 6: Different Emotions, Image courtesy [3]

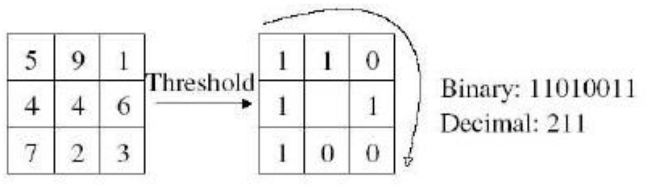
Chapter 3 - Proposed Algorithms

Face detection is a necessary first-step in face recognition systems, with the purpose of localizing and extracting the face region from the background. A wide variety of techniques have been proposed, ranging from simple edge-based algorithms to composite high-level approaches utilizing advanced pattern recognition methods. The solution to the problem involves segmentation, extraction, and verification of faces and possibly facial features from an uncontrolled background [7].

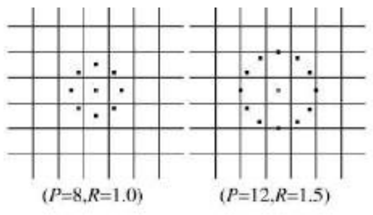
3.1 Local Binary Patterns

An efficient algorithm for mobile products is facial recognition using LBP (Local Binary Pattern). LBP features have performed very well in various applications, including texture classification and segmentation, image retrieval and surface inspection [8].

LBP operator (Ojala et al. 1996) forms labels for the image pixels by thresholding the 3 x 3 neighbourhood of each pixel with the centre value and considering the result as a binary number.



The basic LBP operator



Circular (8,1) Neighbourhood   Circular (12,1.5) neighbourhood

Figure 7: Image courtesy [9]

Concept:

The LBP feature vector, in its simplest form, is created in the following manner:

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbours (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.

- Where the centre pixel's value is greater than the neighbour’s value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).
- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the centre). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window[10]

The 256-bin histogram of the labels computed over an image can be used as a texture descriptor. Each bin of histogram (LBP code) can be regarded as a micro-texton. Local primitives which are codified by these bins include different types of curved edges, spots, flat areas, etc [8].

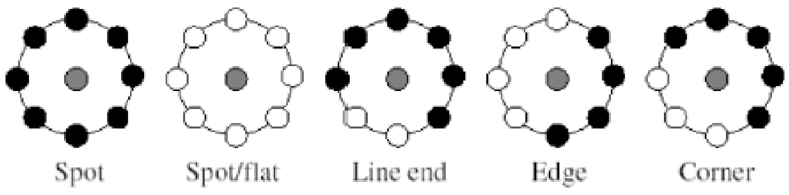


Figure 8: Examples of texture primitives [9]

Further extension of LBP is to use uniform patterns. A Local Binary Pattern is called uniform if it contains at most two bitwise transitions from 0 to 1 or vice versa when the binary string is considered circular. For example, 00000000, 001110000 and 11100001 are uniform patterns. In the LBP operator $LBP^{u2}_{P,R}$. the subscript represents using the operator in a (P,R) neighbourhood; the superscript u2 indicates using only uniform patterns and labelling all remaining patterns with a single label. A histogram of a labelled image f(x, y) can be defined as

$$H_i = \sum_{x,y} I(f_l(x,y) = i), \qquad i = 0, \ldots, n-1$$

Where n is the number of different labels produced by the LBP operator and

$$I(A) = \begin{cases} 1 & A \text{ is true} \\ 0 & A \text{ is false} \end{cases}$$

The use of LBP is that it is illumination invariant because with increase in illumination density the pixel values go up but the relative values remains the same therefore binary patterns remains the same.

3.2 Cascade of “weak-classifiers” using Haar features

For finding face in unconstrained scenes, we propose our further study in face detection and feature extraction using a cascade of “weak-classifiers”, using Haar features as proposed by Viola & Jones. This method after excessive training yields impressive results. This approach is now the most commonly used algorithm for face detection. A basic implementation is included in OpenCV.

The key contribution by Viola & Jones includes:

1) The introduction of a new image representation called the “Integral Image” which allows the features used by our detector to be computed very quickly.
2) A learning algorithm, based on AdaBoost, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers
3) A method for combining increasingly more complex classifiers in a “cascade” which allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions.

In the domain of face detection the system yields detection rates comparable to the best previous systems. Used in real-time applications, the detector runs at 15 frames per second without resorting to image differencing or skin color detection.

The integral image can be computed from an image using a few operations per pixel. Once computed, any one of these Harr-like features can be computed at any scale or location in constant time.

Within any image sub window the total number of Harr-like features is very large, far larger than the number of pixels. In order to ensure fast classification, the learning process must exclude a large majority of the available features, and focus on a small set of critical features.The weak learner is constrained so that each weak classifier returned can depend on only a single feature. As a result each stage of the boosting process, which selects a new weak classifier, can be viewed as a feature selection process.

A method for combining successively more complex classifiers in a cascade structure dramatically increases the speed of the detector by focusing attention on promising regions of the image.

3.2.1 Features

This method classifies images based on the value of simple features. The motivation of choosing is that features can act to encode ad-hoc domain knowledge that is difficult to learn using a finite quantity of training data. The feature based system operates much faster than a pixel-based system.

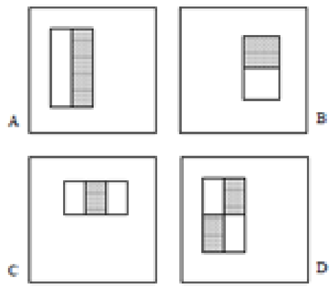


Figure 9: Example of rectangle features shown relative to the enclosing detection window. The sum of pixels which lie within the white rectangles are subtracted from the sum of pixels in grey rectangles. (A) and (B) shows two rectangle features. (C) shows three rectangle features and (D) a four rectangle feature [11].

There are specifically three kinds of features. The value of a two-rectangle feature is the difference between the sums of the pixels within two rectangular regions. The regions have the same size and shape and are horizontally or vertically adjacent (see Figure 9). A three-rectangle feature computes the sum within two outside rectangles subtracted from the sum in a centre rectangle. Finally a four-rectangle feature computes the difference between diagonal pairs of rectangles.

Given that the base resolution of the detector is 24x24, the exhaustive set of rectangle features is quite large, over 180,000.

3.2.2 Integral Image

Rectangle features can be computed very rapidly using an intermediate representation for the image called the integral image.

The integral image at location (x,y) contains the sum of the pixels above and to the left of (x,y) , inclusive.

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

Using the integral image any rectangular sum can be computed in four array references:

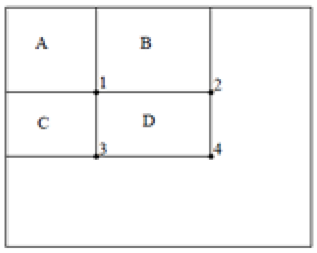


Figure 10: The sum of pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is A+B, at 3 is A+C and at location 4 is A+B+C+D. The sum within D can be computed as 4+1-(2+3) [11].

Since the two-rectangle features defined above involve adjacent rectangular sums they can be computed in six array references, eight in the case of the three-rectangle features, and nine for four-rectangle features.

3.2.3 Learning Classification Functions

Here, a variant of AdaBoost is used both to select a small set of features and train the classifier. The weak learning algorithm is designed to select the single rectangle feature which best separates the positive and negative examples.For each feature, the weak learner determines the optimal threshold classification function, such that the minimum number of examples is misclassified.

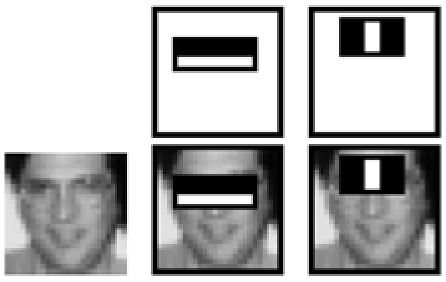


Figure 11: The first and second feature selected by AdaBoost. The first feature measures the difference in intensity between the region of the eyes and region across the upper cheeks. The second feature compares the intensities in the eye regions to the intensity across the bridge of nose [11].

3.2.4 The Additional Cascade

Here, there is a cascade of classifiers which achieves increased detection performance while radically reducing computation time. Smaller, and therefore more efficient, boosted classifiers can be constructed which reject many of the negative sub-windows while detecting almost all positive instances. The overall form of the detection process is that of a degenerate decision tree, what we call a “cascade”.

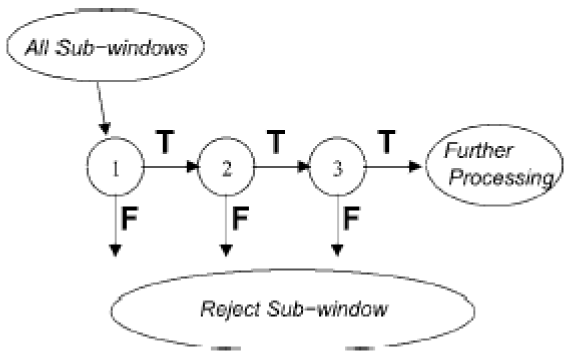


Figure 12: Schematic depiction of the detection cascade [11].

Much like a decision tree, subsequent classifiers are trained using those examples which pass through all the previous stages. As a result, the second classifier faces a more difficult task than the first [11].

Haar-like feature provides good performance in extracting textures and cascading architecture and integral image representation make it computationally efficient [8].

Chapter 4 - Platform and Libraries

This chapter outlines the details of the platform and the libraries used.

4.1 Platforms and libraries used

1. OpenCV
2. Android Studio

4.2 OpenCV

OpenCV (Open Source Computer Vision) is an open source computer vision and machine learning software library. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. The library has various algorithms that can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc [14].

4.3 Android Studio

Android powers hundreds of millions of mobile devices and is the largest installed base of any mobile platform and growing fast. Android provides a world-class platform for creating apps and games for Android users, as well as an open marketplace for distributing to them instantly. Android Studio is the official Integrated Development Environment (IDE) for Android app development. It is the fastest way to build high quality, performant apps for the Android platform, including phones and tablets, Android Auto, Android Wear, and Android TV [15]. OpenCV4Android SDK package enables development of Android applications with use of OpenCV library. The face detection functionality of OpenCV libraries on Android supports two modes of execution: available by default Java wrapper for the cascade classifier, and manually crafted JNI call to a native class which supports tracking. Currently, we have implemented mode one usage of it.

Chapter 5 - Implementation and Application: Phase I

We have implemented facial feature recognition by integrating OpenCV library with Android Studio. The application needs the support of OpenCV manager.

5.1 Application Overview

The overview of the application to be implemented is as shown below.

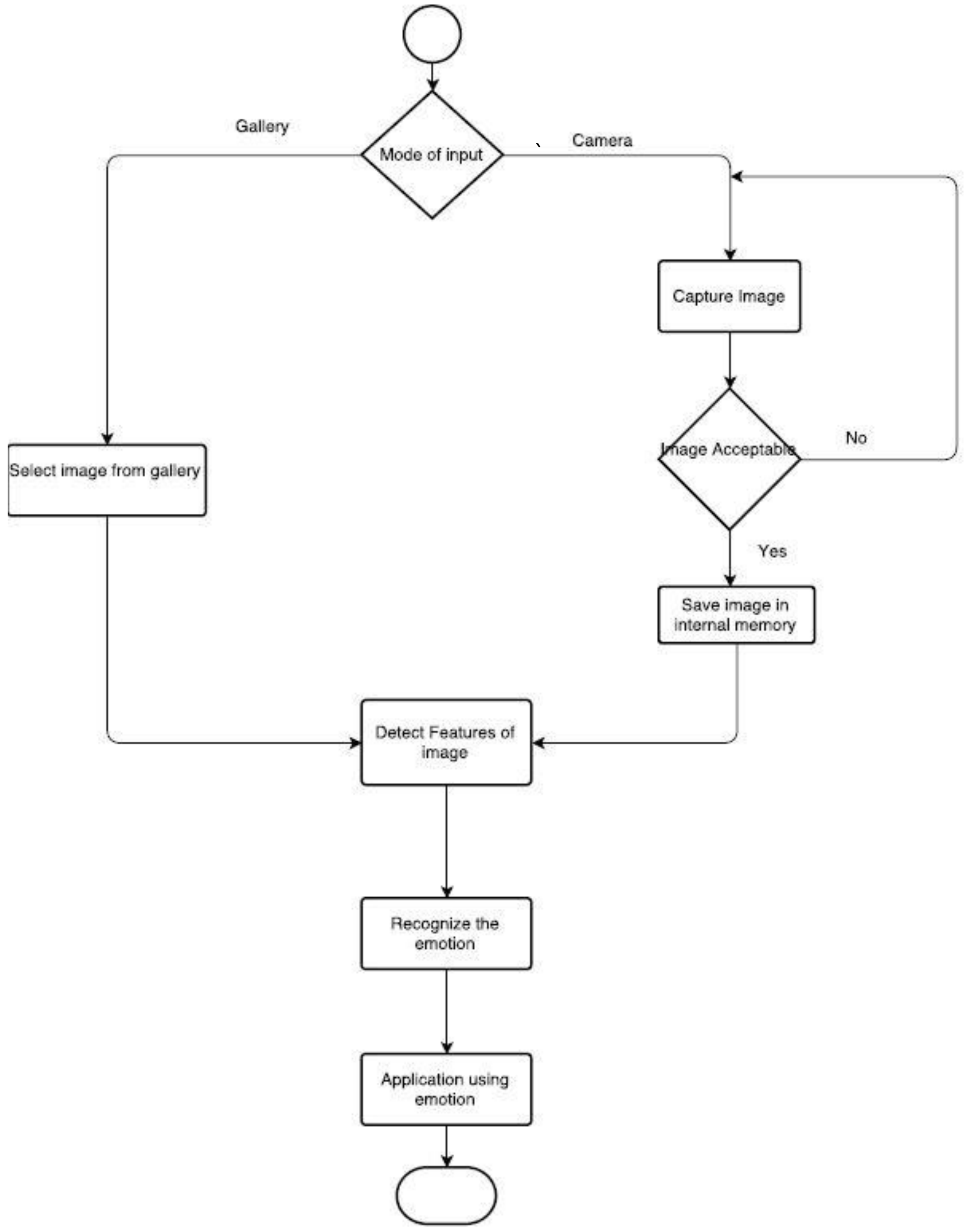


Figure 13: An overview of a system using emotion recognition

The application allows the user to specify the mode of input. The user can select from three modes –

1. from Camera
2. from Gallery
3. Live Detection

If the first option is selected, then the device's camera is opened and the user can click a photo. If the user selects the second option then the device's gallery is opened and the user is able to choose a photo from it. The user has an option to save the taken photograph or can take another one instead. The user also has an option to detect emotion on live camera feed (using the front camera).

After the mode of picture input is selected, the face detection and feature extraction process is done and the emotion is recognized. Finally, the emotion thus detected can be used in various application domains, few of which are enlisted in this report.

5.2 Feature Extraction

The flow of the feature extraction part of the android application is depicted by the following flow diagram (Fig 14).

The feature extraction part of the application uses OpenCV with Android. The application starts by loading the OpenCV manager and the cascade files for the features that are to be extracted. Upon their successful loading, the application continues to detect the faces in the present camera frame using LBP/HAAR cascades.

The region of interest for the respective features is set and this area is passed to the function "get_template" for learning. The frame count is used to decide the no of learning frames and then the final template is used for matching.

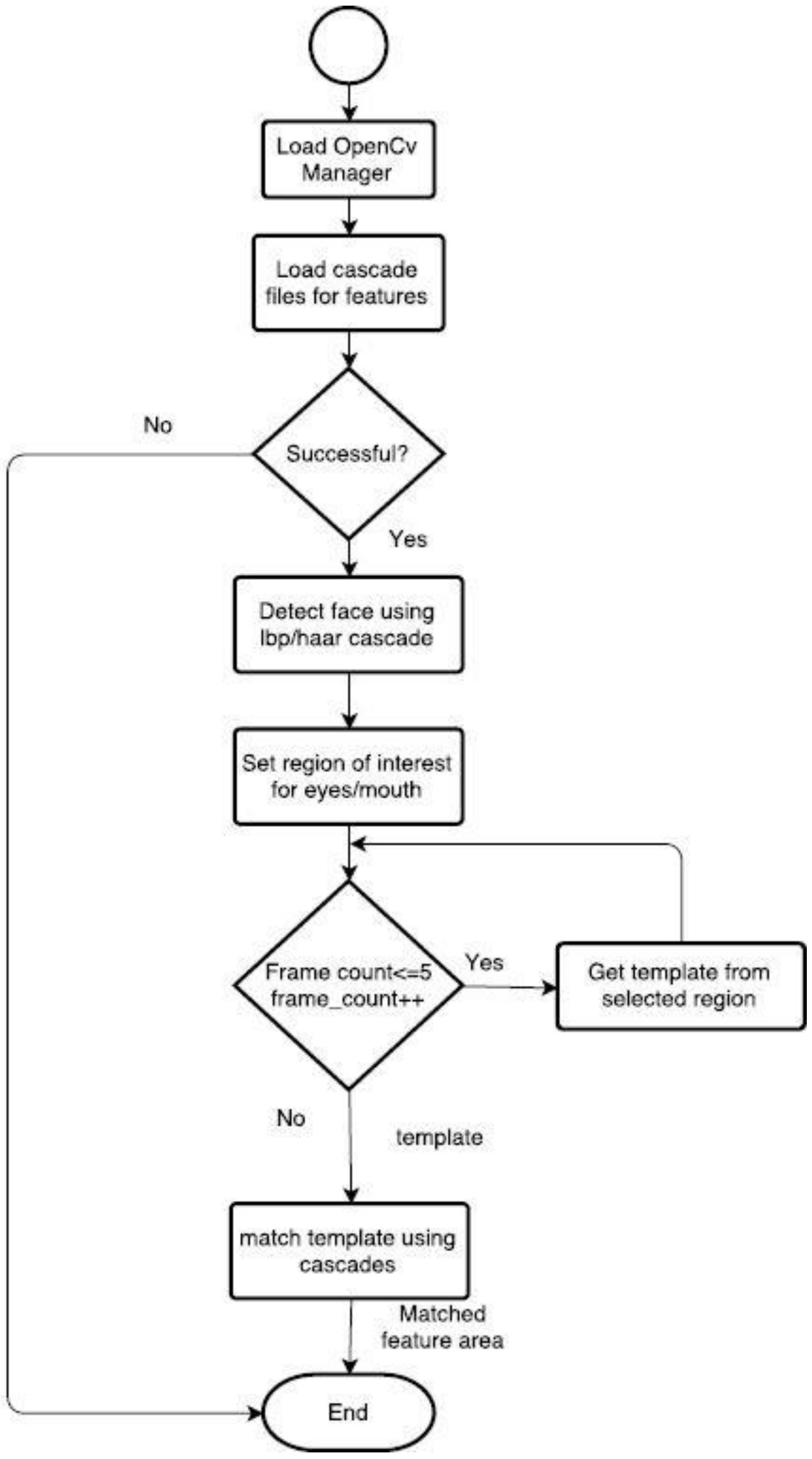

Load OpenCv Manager
Load cascade files for features
Successful?
No
Yes
Detect face using lbp/haar cascade
Set region of interest for eyes/mouth
Frame count<=5 frame_count++
Yes
Get template from selected region
No
template
match template using cascades
Matched feature area
End

Figure 14: Flow diagram of feature extraction implementation

5.3 Applications

1. Affective Car Interfaces with emotion recognition:
A potential application for multi-media technologies is affective intelligent car interfaces for enhanced driving safety. Certain physiological signals (galvanic skin response, heartbeat, and temperature) map to certain driving-related emotions and states (Frustration/Anger, Panic/Fear, and Boredom/Sleepiness). Drivers do emote while they are driving in their cars and their driving is affected by their emotions.

The inability to manage one's emotions while driving is often identified as one of the major causes for accidents. Anger is one of the emotions that negatively affect one's driving. When drivers become angry, they start feeling self-righteous about events and anger impairs their normal thinking and judgment, their perception is altered, thus leading to the misinterpretation of events. Other states that lead to negative effects are frustration, anxiety, fear, and stress. In order to be a safer driver on the highways, a person needs to be better aware of her emotions and possess the ability to manage them effectively. Once drivers are aware of their emotional states it becomes easier for them to respond to the situation in a safe manner, but drivers can often lack in awareness. For example, when the intelligent system recognizes the anger or rage of a driver it might suggest the driver to perform a breathing exercise. Similarly, when the system recognizes driver's sleepiness, it might change the radio station for a different tune or roll down the window for fresh air. Having a natural communication between the system and the driver, and taking the precautions mentioned above automatically, depending on the drivers' personal preferences, would increase the drivers' feeling that there exists a real person in the car with them to assist them while they drive [16].

2. Business application – Achieving customer satisfaction using real-time emotion detection in Restaurants:
Customer satisfaction plays an important role within your business. Not only is it the leading indicator to measure customer loyalty, identify unhappy customers, reduce churn and increase revenue; it is also a key point of differentiation that helps you to attract new customers in competitive business environments.

The facial expressions of customers can be used as a measure of satisfaction, especially in the hospitality industry. In case of a restaurant, the CCTV camera footage can be used as a real-time video feed for the emotion recognition application. This application will track expressions of the customers and will recognize the emotions they portray. Every time a negative emotion like disgust, sadness or anger surfaces on the customer's face, the application will then send this data along with a suitable suggestion to the concerned authority, in order to attend to the said customer's needs.

5.4 Screenshots

The following are the screenshots of the application implemented.

Figure 15 shows the application's icon on the device screen and also the OpenCV manager required to use OpenCV libraries on android.

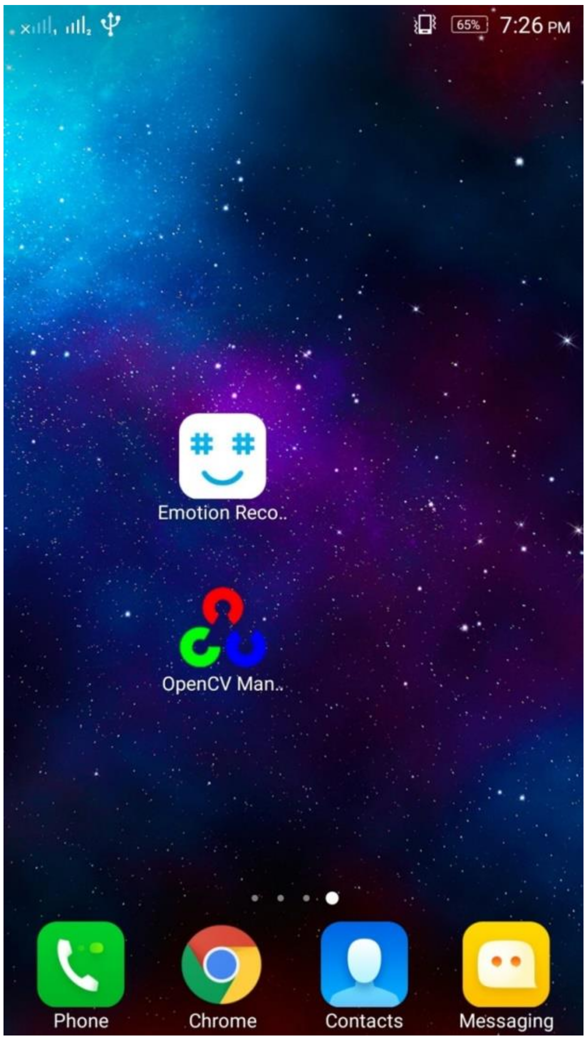


Figure 15: Emotion Recognition icon on the device screen

Figure 16, shows the start screen of the application. On clicking the emoji icon on the screen, the user will be navigated to the next menu screen.

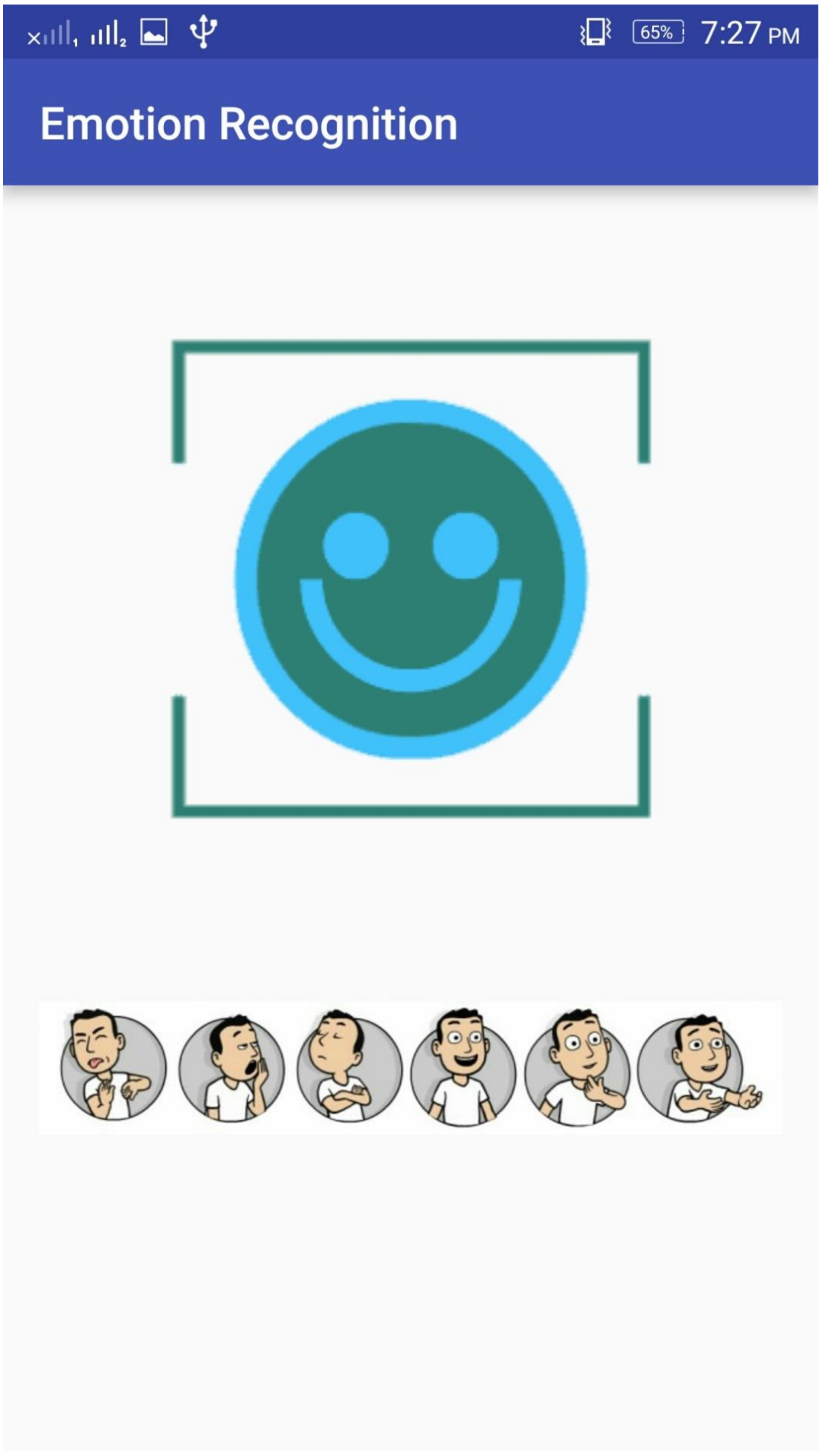


Figure 16: Application start page

Figure 17, shows the menu screen for selecting the input mode. The three options made available to the users are: OpenCV Camera, Capture Image, Open Gallery.

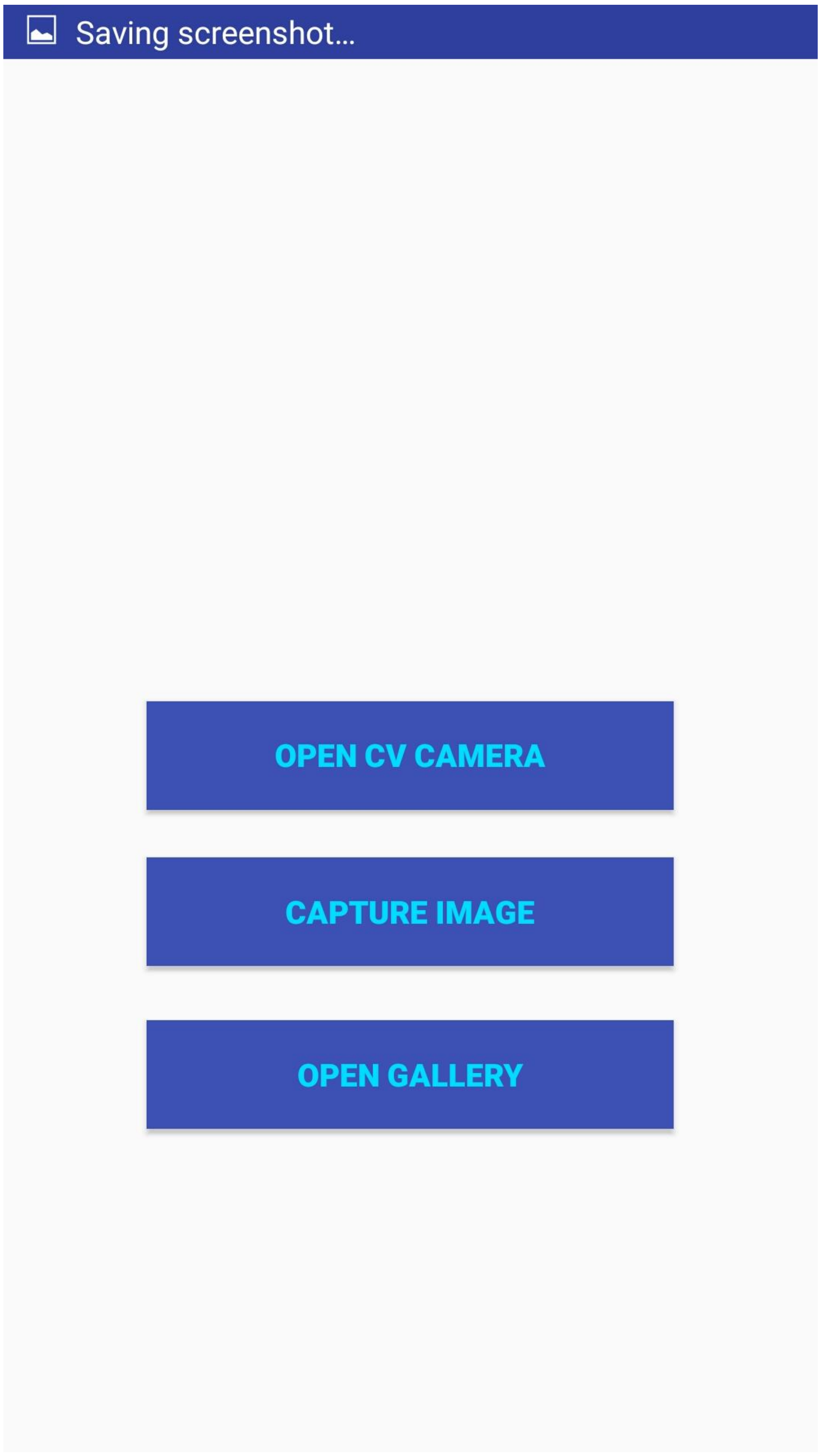


Figure 17: Selecting input mode

Figure 18, shows the actual detection of faces and the region of features (eyes, mouth) of the subject. It is the real-time feature detection using feed from camera.

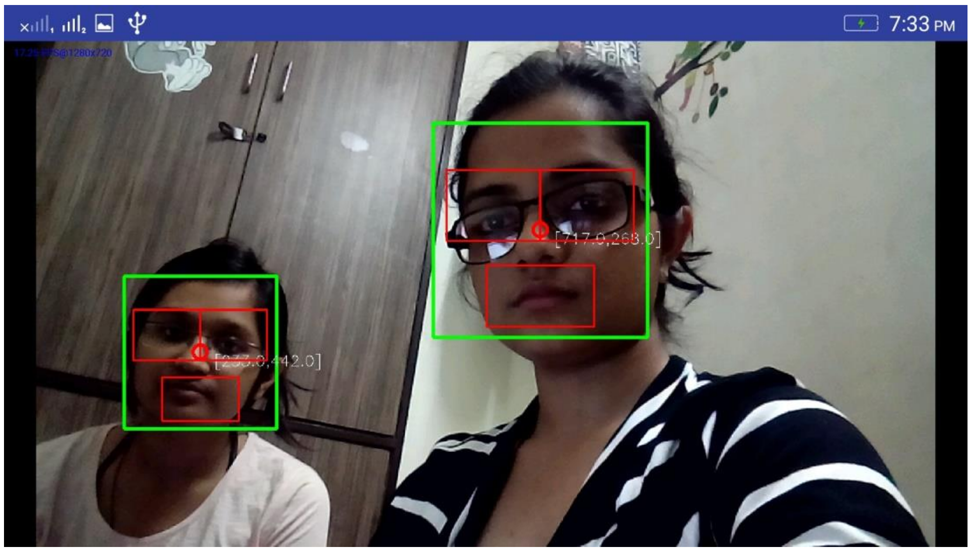


Figure 18: Real-time feature detection of the camera feed

Chapter 6 - Implementation: Phase II

In this implementation we use Haar cascade for extracting facial features and fisherface for classification.

6.1 Theory

Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. The theory of Haar feature-based cascade classifier is detailed in section 3.2. We have used HAAR filter from OpenCV to automate face finding.

Fisher faces:

For creating the training and classification set to teach the classifier what an emotion looks like, dataset organisation is important.

One way to represent the input data is by finding a subspace which represents most of the data variance. This can be obtained with the use of Principal Components Analysis (PCA). When applied to face images, PCA yields a set of eigenfaces. These eigenfaces are the eigenvectors associated to the largest eigenvalues of the covariance matrix of the training data. The eigenvectors thus found correspond to the least-squares (LS) solution. This is indeed a powerful way to represent the data because it ensures the data variance is maintained while eliminating unnecessary existing correlations among the original features (dimensions) in the sample vectors.

When the goal is classification rather than representation, the LS solution may not yield the most desirable results. In such cases, one wishes to find a subspace that maps the sample vectors of the same class in a single spot of the feature representation and those of different classes as far apart from each other as possible. The techniques derived to achieve this goal are known as discriminant analysis (DA).

The most known DA is Linear Discriminant Analysis (LDA), which can be derived from an idea suggested by R.A. Fisher in 1936. When LDA is used to find the subspace representation of a set of face images, the resulting basis vectors defining that space are known as Fisherfaces [17].

Using class specific linear methods for dimensionality reduction and simple classifiers in the reduced feature space, one may get better recognition rates. Fisher's Linear Discriminant (FLD) is an example of a class specific method, it tries to "shape" the scatter in order to make it more reliable for classification [18].

6.2 Libraries used

For recognising emotions on images we used OpenCV with Python.

For the purpose of face detection, we have used the Haar Cascade provided by OpenCV. The Haar cascades that come with OpenCV are located in the "/data/haarcascades" directory of your OpenCV installation. We have used haarcascade_frontalface_default.xml for detecting the face. So, we load the cascade using the cv2.CascadeClassifier function which takes the path to the cascade xml file.

OpenCV has few 'face recognizer' classes (Eigenfaces, Fisherfaces, Local Binary Patterns Histograms) that we can also use for emotion recognition. The face recognizer object has functions like "FaceRecognizer.train" to train the recognizer and "FaceRecognizer.predict" to recognize a face. The algorithm for Fisherfaces in OpenCV can be used using createFisherFaceRecognizer().

6.3 Procedure

The following is the procedure followed in this implementation:

- The first step was to organize the dataset. The dataset contains emotion sequences. Each image sequence consists of the forming of an emotional expression, starting with a neutral face and ending with the emotion. So, from each image sequence two images are extracted; one neutral (the first image) and one with an emotional expression (the last image).
- In order for the classifier to give its best results, all the images must be of the same size and must contain only a face. We need to find the face on each image, convert to grayscale, crop it and save the image to the dataset. We use 4 in-built HAAR filters from OpenCV to automate finding, to make sure that we detect as many faces as possible.
- Next step is to teach the classifier what certain emotions look like. The dataset is split into training and prediction sets and then we use the training set to teach the classifier to

recognize the to-be-predicted labels, and use the classification set to estimate the classifier performance.

- Finally, after training the classifier, prediction of the images is done and the percentage performance of the classifier is calculated [22].
- This trained classifier is then used to classify emotions on human faces, by capturing human faces through live video feed.

6.4 Code Snippet

Script 1: process_dataset.py

```
#process_dataset.py
#script 1
"""
Classify the images based on the text file associated with each image describing the emotion
depicted in the image and segregate them into different folders.
"""
import glob
from shutil import copyfile

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #Define emotion order
participants = glob.glob("source_emotion\\*") #Returns a list of all folders with participant numbers

for x in participants:
    part = "%s" %x[-4:] #store current participant number
    print("Part: %s",part)
    for sessions in glob.glob("%s\\*" %x): #Store list of sessions for current participant
        for files in glob.glob("%s\\*" %sessions):
            current_session = files[20:-30]
            #print("Current Session: %s",current_session)
            file = open(files, 'r')

            emotion = int(float(file.readline()))

            sourcefile_emotion = glob.glob("source_images\\%s\\%s\\*" %(part, current_session))[-1] #selecting the last image containing the emotion
            sourcefile_neutral = glob.glob("source_images\\%s\\%s\\*" %(part, current_session))[0] #selecting the first image containing the neutral face

            dest_neut = "sorted_set\\neutral\\%s" %sourcefile_neutral[25:] #Generating path to put neutral image
            dest_emot = "sorted_set\\%s\\%s" %(emotions[emotion], sourcefile_emotion[25:]) #Generating path to put image containing emotion

            copyfile(sourcefile_neutral, dest_neut)
            copyfile(sourcefile_emotion, dest_emot)
```

Script 2: normalize_image.py

```
#normalize_image.py
#script2

'''
    a.  Load the cascade classifiers from their respective xml files
    b.  For every emotion do:
        i.      Read the image file and convert to grayscale
        ii.     Use the classifiers to detect the face
        iii.    Go over detected faces, stop at first detected face, return empty if no face.
        iv.     Cut the face to make every image of same size
        v.      Write the same dimensional images into the dataset folder, to be used further.

'''

import cv2
import glob

faceDet = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
faceDet2 = cv2.CascadeClassifier("haarcascade_frontalface_alt2.xml")
faceDet3 = cv2.CascadeClassifier("haarcascade_frontalface_alt.xml")
faceDet4 = cv2.CascadeClassifier("haarcascade_frontalface_alt_tree.xml")

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #list of emotions

def detect_faces(emotion):
    files = glob.glob("sorted_set\\%s\\*" %emotion) #list of all images with emotion

    filenumber = 0
    for f in files:
        frame = cv2.imread(f) #Opening image
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) #Converting image to grayscale

        #Detect face using 4 different classifiers
        face = faceDet.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCALE_IMAGE)
        face2 = faceDet2.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCALE_IMAGE)
        face3 = faceDet3.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCALE_IMAGE)
        face4 = faceDet4.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=10, minSize=(5, 5), flags=cv2.CASCADE_SCALE_IMAGE)
        #Go over detected faces, stop at first detected face, return empty if no face.
        if len(face) == 1:
            facefeatures = face
        elif len(face2) == 1:
            facefeatures == face2
        elif len(face3) == 1:
            facefeatures = face3
        elif len(face4) == 1:
            facefeatures = face4
        else:
            facefeatures = ""

        #Cut and save face
        for (x, y, w, h) in facefeatures: #get coordinates and size of rectangle containing face
            print "face found in file: %s" %f
            gray = gray[y:y+h, x:x+w] #Cut the frame to size

            try:
                out = cv2.resize(gray, (350, 350)) #Resize face so all images have same size
                cv2.imwrite("dataset\\%s\\%s.jpg" %(emotion, filenumber), out) #Write image
            except:
               pass
        filenumber += 1 #Increment image number

for emotion in emotions:
    detect_faces(emotion) #Call functions
```

Script 3: model_training.py

```
#model_training.py
#script 3
'''
a.      Get_files function:
                i.Get the files and split them into training and prediction data sets.

b.      Make_sets function:
                i.Make different sets according to the get_files function
                ii.For each file in the training or prediction set, read the image and convert to gray.
                iii.Make a list of the training (or prediction) images and the labels.

c.      Run_recognizer function:
                i.Call the make_files function.
                ii.Train the data in the lists using fishface.train()
                iii.Use the predict function for prediction
                iv.Find the percentage correctness

d.      Start from Run_recognizer function and do the process desired number of times.

'''
import cv2
import glob
import random
import numpy as np

emotions = ["neutral", "anger", "contempt", "disgust", "fear", "happy", "sadness", "surprise"] #list of emotions
fishface = cv2.createFisherFaceRecognizer() #Initialization of fisher face classifier

data = {}

def get_files(emotion): #Define function to get file list, randomly shuffle it and split 80/20
    files = glob.glob("dataset\\%s\\*" %emotion)
    #print("files are: %s", files)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    return training, prediction


def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []
    for emotion in emotions:
        training, prediction = get_files(emotion)

        #Append data to training and prediction list, and generate labels 0-7
        for item in training:
            image = cv2.imread(item) #open image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #convert to grayscale
            training_data.append(gray) #append image array to training data list
            training_labels.append(emotions.index(emotion))

        for item in prediction: #repeat above process for prediction set
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            prediction_data.append(gray)
            prediction_labels.append(emotions.index(emotion))

    return training_data, training_labels, prediction_data, prediction_labels
```

```
def run_recognizer():
    training_data, training_labels, prediction_data, prediction_labels = make_sets()

    print "training fisher face classifier"
    print "size of training set is:", len(training_labels), "images"
    fishface.train(training_data, np.asarray(training_labels))

    print "predicting classification set"
    cnt = 0
    correct = 0
    incorrect = 0
    for image in prediction_data:
        pred, conf = fishface.predict(image)
        if pred == prediction_labels[cnt]:
            correct += 1
            cnt += 1
        else:
            incorrect += 1
            cnt += 1
    return ((100*correct)/(correct + incorrect))

if __name__ == '__main__':
    metascore = []
    for i in range(0,10):
        correct = run_recognizer()
        print "got", correct, "percent correct!"
        metascore.append(correct)

    print "\n\nend score:", np.mean(metascore), "percent correct!"
    fishface.save('models\emotion_detection_model.xml') #save the training files to the model
```

Script 4: emotion_detect.py

```
#emotion_detect.py
#script 4
"""
a.      nparray_as_image and image_as_nparray convert image to nparray and vice versa
b.      _load_emoticons loads the graphic image corresponding to every emotion
c.      The model is loaded and trained data is stored in xml file
d.      show_webcam_and_run is used to launch the camera of the device and predict the emotion
        i.      this function calls find_faces which calls _locate_faces to detect all faces in the image,
        cuts the faces and then normalizes the faces (i.e. to gray and resizing) and then zips them and returns it
        ii.     finally it predicts the emotion of the normalized faces using the trained model and draws the related
        emoticon on the respective faces

This module is the main module in this package. It loads emotion recognition model from a file,
shows a webcam image, recognizes face and it's emotion and draw emotion on the image.
"""

import cv2
from cv2 import WINDOW_NORMAL
import numpy as np
#import cv2
if cv2.__version__ == '3.1.0':
    from PIL import Image
else:
    from PIL import Image


faceCascade = cv2.CascadeClassifier('models/haarcascade_frontalface_default.xml')
def image_as_nparray(image):
    return np.asarray(image)


def nparray_as_image(nparray, mode='RGB'):
    return Image.fromarray(np.asarray(np.clip(nparray, 0, 255), dtype='uint8'), mode)

def load_image(source_path):
    source_image = cv2.imread(source_path)  #reading the image from source
    return cv2.cvtColor(source_image, cv2.COLOR_BGR2GRAY)       #converting image to grayscale
```

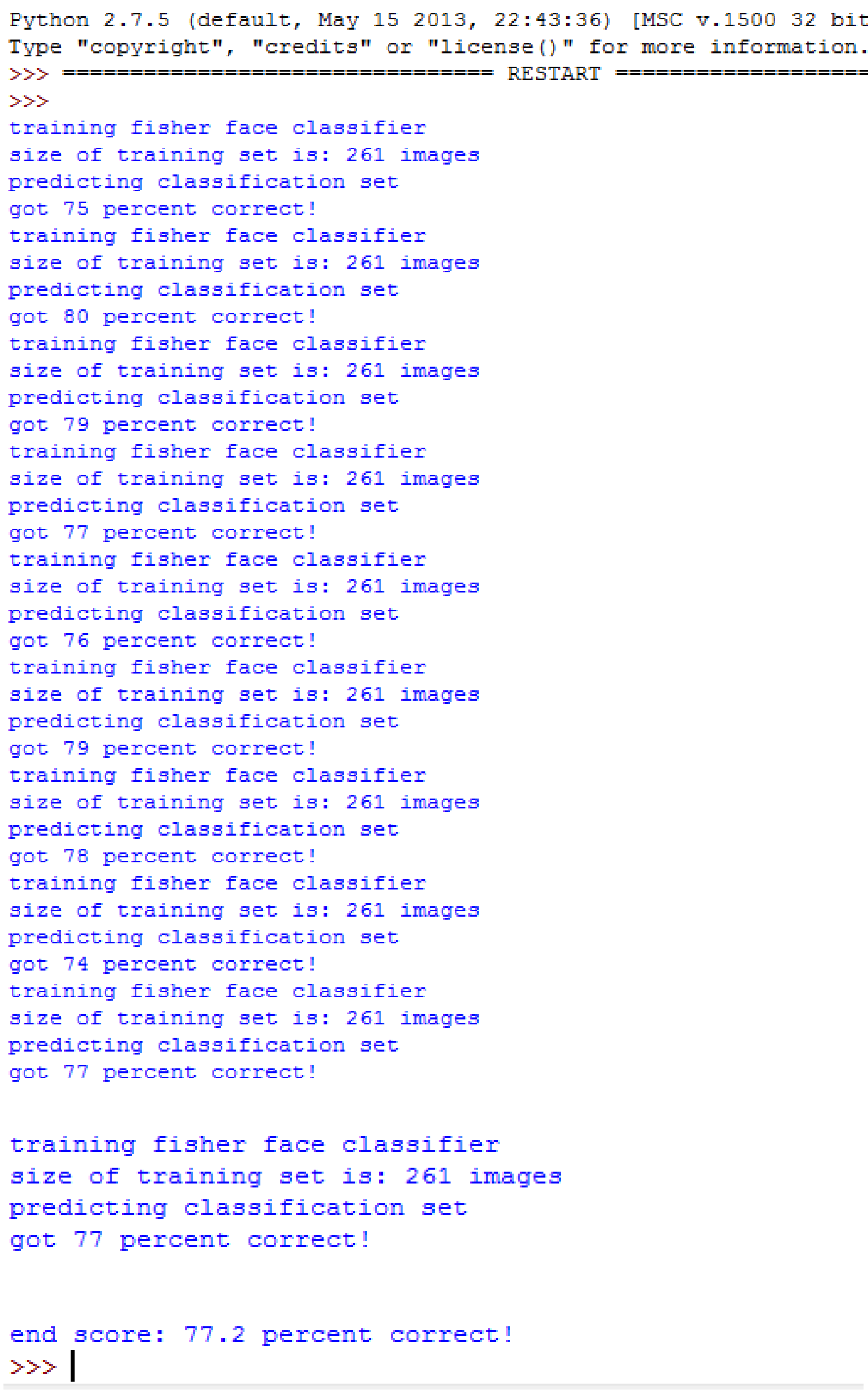
```
def _normalize_face(face):
    face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY) #normalizing the face image
    face = cv2.resize(face, (350, 350))

    return face;
def _locate_faces(image):
        #locating multiple faces in the image
    faces = faceCascade.detectMultiScale(
        image,
        scaleFactor=1.1,
        minNeighbors=15,
        minSize=(70, 70)
    )
    return faces  # list of (x, y, w, h)

def find_faces(image):
        #finding faces in the frame and returning normalized face images
    faces_coordinates = _locate_faces(image)
    cutted_faces = [image[y:y + h, x:x + w] for (x, y, w, h) in faces_coordinates]
    normalized_faces = [_normalize_face(face) for face in cutted_faces]
    return zip(normalized_faces, faces_coordinates)

def draw_with_alpha(source_image, image_to_draw, coordinates):
        #Draws a partially transparent image over another image.
    x, y, w, h = coordinates
    image_to_draw = image_to_draw.resize((h, w), Image.ANTIALIAS)
    image_array = image_as_nparray(image_to_draw)
    for c in range(0, 3):
        source_image[y:y + h, x:x + w, c] = image_array[:, :, c] * (image_array[:, :, 3] / 255.0) \
                                            + source_image[y:y + h, x:x + w, c] * (1.0 - image_array[:, :, 3] / 255.0)



def _load_emoticons(emotions):
    return [nparray_as_image(cv2.imread('graphics/%s.png' % emotion, -1), mode=None) for emotion in emotions]
```

```
def _normalize_face(face):
    face = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY) #normalizing the face image
    face = cv2.resize(face, (350, 350))

    return face;
def _locate_faces(image):
        #locating multiple faces in the image
    faces = faceCascade.detectMultiScale(
        image,
        scaleFactor=1.1,
        minNeighbors=15,
        minSize=(70, 70)
    )
    return faces  # list of (x, y, w, h)

def find_faces(image):
        #finding faces in the frame and returning normalized face images
    faces_coordinates = _locate_faces(image)
    cutted_faces = [image[y:y + h, x:x + w] for (x, y, w, h) in faces_coordinates]
    normalized_faces = [_normalize_face(face) for face in cutted_faces]
    return zip(normalized_faces, faces_coordinates)

def draw_with_alpha(source_image, image_to_draw, coordinates):
        #Draws a partially transparent image over another image.
    x, y, w, h = coordinates
    image_to_draw = image_to_draw.resize((h, w), Image.ANTIALIAS)
    image_array = image_as_nparray(image_to_draw)
    for c in range(0, 3):
        source_image[y:y + h, x:x + w, c] = image_array[:, :, c] * (image_array[:, :, 3] / 255.0) \
                                            + source_image[y:y + h, x:x + w, c] * (1.0 - image_array[:, :, 3] / 255.0)



def _load_emoticons(emotions):
    return [nparray_as_image(cv2.imread('graphics/%s.png' % emotion, -1), mode=None) for emotion in emotions]
```

```
def emo_detect(model, emoticons, window_size=None, window_name='webcam', update_time=10):
    #Shows webcam image, detects faces and its emotions in real time and draw emoticons over those faces.

    cv2.namedWindow(window_name, WINDOW_NORMAL)
    if window_size:
        width, height = window_size
        cv2.resizeWindow(window_name, width, height)

    vc = cv2.VideoCapture(0)
    #print "helllo"
    if vc.isOpened():
        read_value, webcam_image = vc.read()
        #print "ere too"
    else:
        print("webcam not found")
        return

    while read_value:
        for normalized_face, (x, y, w, h) in find_faces(webcam_image):
            prediction = model.predict(normalized_face)  # do prediction
            if cv2.__version__ != '3.1.0':
                prediction = prediction[0]

            image_to_draw = emoticons[prediction]
            draw_with_alpha(webcam_image, image_to_draw, (x, y, w, h))

        cv2.imshow(window_name, webcam_image)
        read_value, webcam_image = vc.read()
        key = cv2.waitKey(update_time)

        if key == 27:  # exit on ESC
            break

    cv2.destroyWindow(window_name)


if __name__ == '__main__':
    emotions = ['neutral', 'anger', 'disgust', 'happy', 'sadness', 'surprise']
    emoticons = _load_emoticons(emotions)

    # load model
    if cv2.__version__ == '3.1.0':
        fisher_face = cv2.face.createFisherFaceRecognizer()
    else:
        fisher_face = cv2.createFisherFaceRecognizer()
    fisher_face.load('models/emotion_detection_model.xml')

    # use learnt model
    window_name = 'WEBCAM (press ESC to exit)'
    emo_detect(fisher_face, emoticons, window_size=(1600, 1200), window_name=window_name, update_time=8)
```

6.5 Observation

With 8 emotions, the classifier gave 77.2% correct result in predicting the dataset.



```
Python 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 75 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 80 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 79 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 77 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 76 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 79 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 78 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 74 percent correct!
training fisher face classifier
size of training set is: 261 images
predicting classification set
got 77 percent correct!

training fisher face classifier
size of training set is: 261 images
predicting classification set
got 77 percent correct!


end score: 77.2 percent correct!
>>>
```

6.6 Screenshots

The following are the screenshots of the application implementation.

Figure 19, shows the neutral and angry emotion detection of two subjects. A live video feed is taken and the detected emotion is mentioned below in the video feed itself.

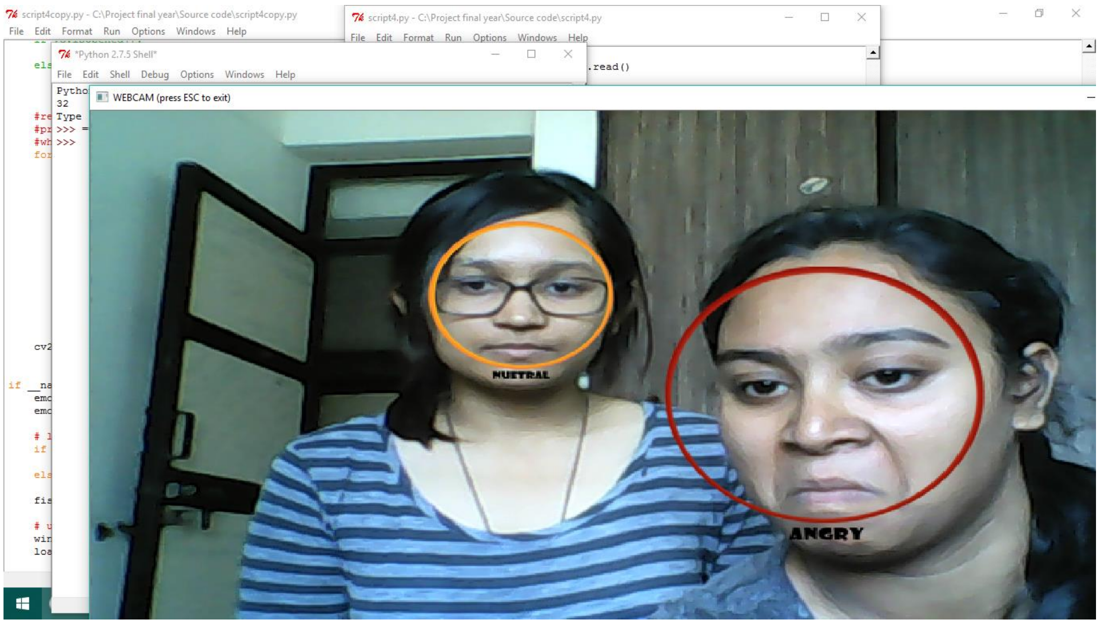

script4copy.py - C:\Project final year\Source code\script4copy.py
script4.py - C:\Project final year\Source code\script4.py
Python 2.7.5 Shell
WEBCAM (press ESC to exit)
NUETRAL
ANGRY

Figure 19: Neutral and angry face detection of two subjects

Figure 20, depicts the detection of emotions on faces of four subjects using the above implementation method. The detected emotions include surprise, anger and neutral emotions.

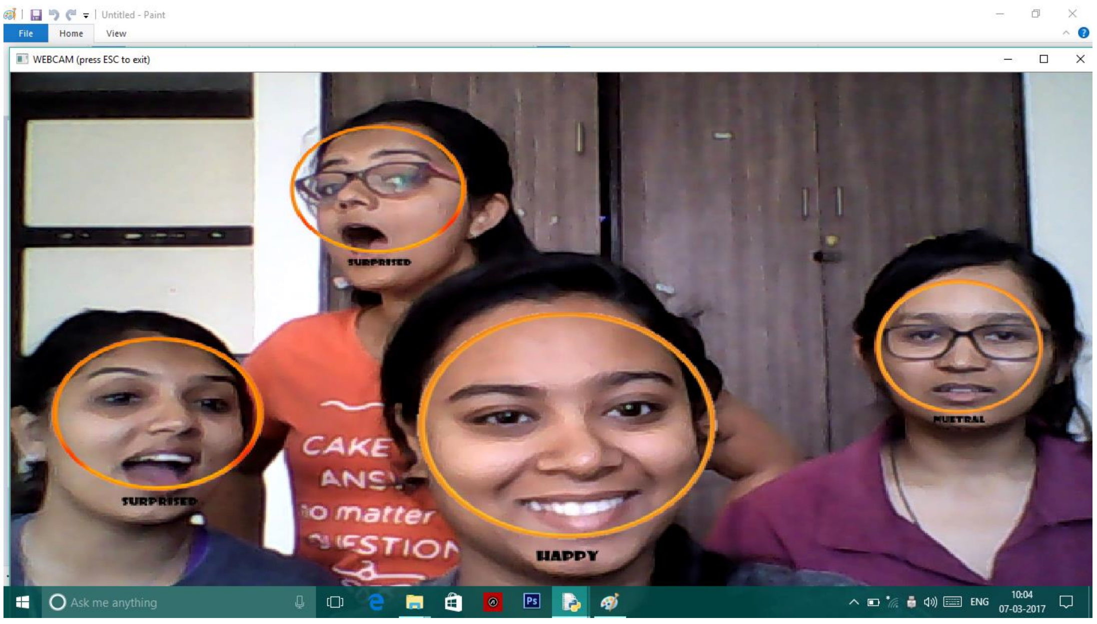


Figure 20: Surprised, angry and neutral face detection of four subjects

Figure 21, depicts the detection of emotions on faces of four subjects using the above implementation method. The detected emotions include surprise, anger and neutral emotions.

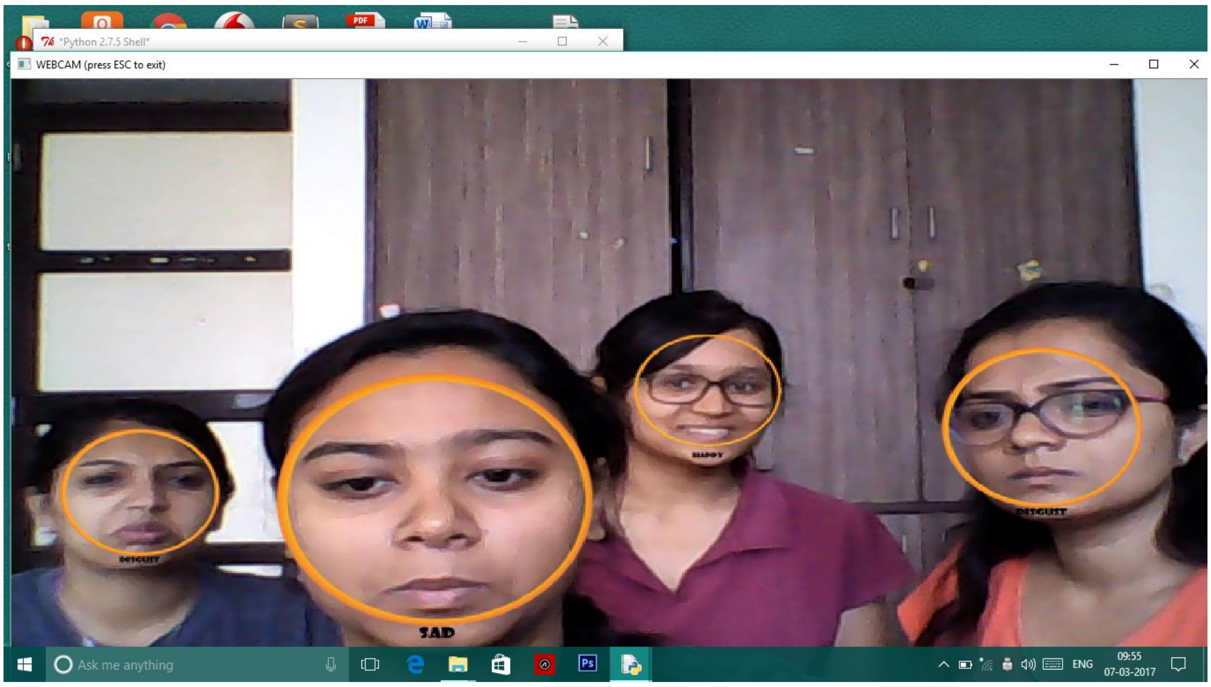


Figure 21: Disgust, happy and sad face detection of four subjects

Chapter 7 - Implementation: Phase III

In this implementation we use facial landmarks and Linear SVM, a machine learning algorithm for emotion recognition. This classifier is more robust and powerful than the earlier used fisherface classifier.

7.1 Theory

Automatically analysing facial expressions in video sequences is a challenging task due to the fact that current techniques for the detection and tracking of facial expressions are sensitive to head pose, occlusion, pose, and variations in lighting conditions. Thus, we have implemented a method based on automatic facial landmark detection.

Facial Landmark detection:

Facial landmarks are a set of salient points, usually located on the corners, tips or mid points of the facial components. The landmark points display the largest displacements and deformations of the facial components during dynamic changes of the expressions [19].

Facial Landmark detection aims to facilitate locating point correspondence between images or between images and a known model where natural features, such as the texture shape or location information, are not present in sufficient quantity and uniqueness. Detecting and tracking landmark points in video sequences enables computers to recognize affective states of humans, as well as the abilities to interpret and respond appropriately to users' affective feedback. Facial landmark algorithm analyses the relative position, size, and/or shape of the eye corners, nose, chin tip, and mouth corners [20].

Support Vector Machine:

Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification and regression. Here, each data item is plotted as a point in n-dimensional space (where n is number of features) with the value of each feature being the value of a particular coordinate. Then, classification is performed by finding the hyper-plane that differentiates the two classes very well.

SVMs exhibit good classification accuracy even when only a modest amount of training data is available, making them particularly suitable to a dynamic, interactive approach to expression recognition. They also avoid the "curse of dimensionality" by placing an upper bound on the margin between the different classes, making it a practical tool for large, dynamic datasets. The feature space may even be reduced further by selecting the most distinguishing features through minimization of the feature set size.

SVMs plot the training vectors in high-dimensional feature space, and label each vector with its class. A hyperplane is drawn between the training vectors that maximize the distance between the different classes.

Pros:

- It works really well with clear margin of separation
- It is effective in high dimensional spaces.
- It is effective in cases where number of dimensions is greater than the number of samples.
- It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

Cons:

- It doesn't perform well, when we have large data set because the required training time is higher
- It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is related SVC method of Python scikit-learn library [21].

7.2 Libraries used

For implementation, here we have used Python (2.7), OpenCV, SKLearn, Dlib.

Dlib is a C++ toolkit containing machine learning algorithms and tools that facilitate creating complex software in C++ to solve real world problems.

For face landmark detection, "shape_predictor_68_face_landmarks.dat" is the trained model for 68 landmarks to perform detection on input image.

7.3 Procedure

The facial landmark dots that appear as a result of running facial landmark detection method, acts as features to feed the classifier that will help us to divide data into categories.

Here the dataset used is the dataset that resulted from the previous implementation, after normalizing the original Cohn-Kanade dataset [23].

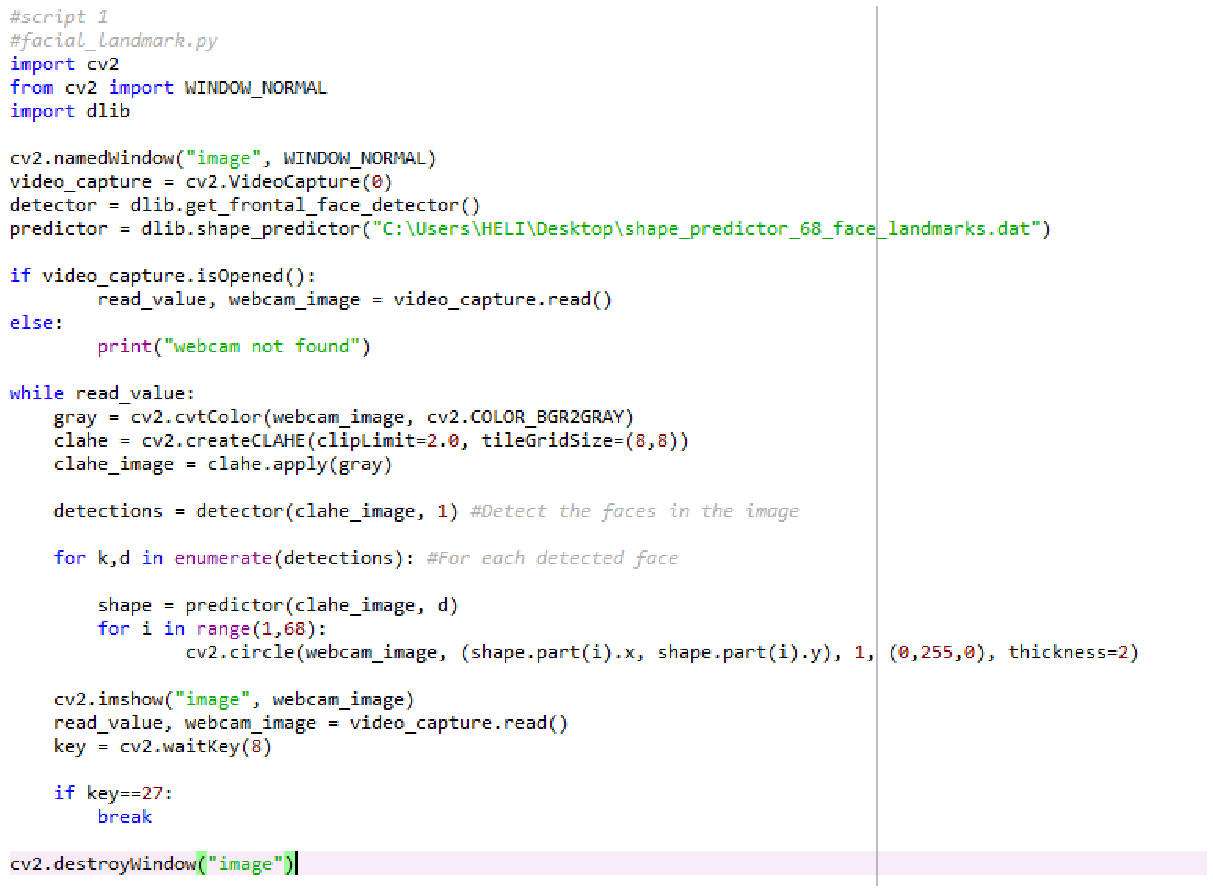
The following is the procedure followed in this implementation:

- First, extract the coordinates of all face landmarks. These coordinates are the first collection of features
- Next calculate the position of all points relative to each other. For this, calculate the mean of both axes, which results in the point coordinates of the sort-of "centre of gravity" of all face landmarks. Then, get the position of all points relative to this central point.
- The faces may be tilted, which might confuse the classifier. To correct for this rotation it is assumed that the bridge of the nose in most people is more or less straight, and so, all calculated angles are offset by the angle of the nose bridge. This rotates the entire vector array so that tilted faces become similar to non-tilted faces with the same expression [24].
- Finally, the existing dataset is split into training and predictions set with corresponding labels, and then train the classifier (SVM) which stores the resulting training data into an xml file for future use.

This trained classifier is then used to predict emotions of humans, through video feed.

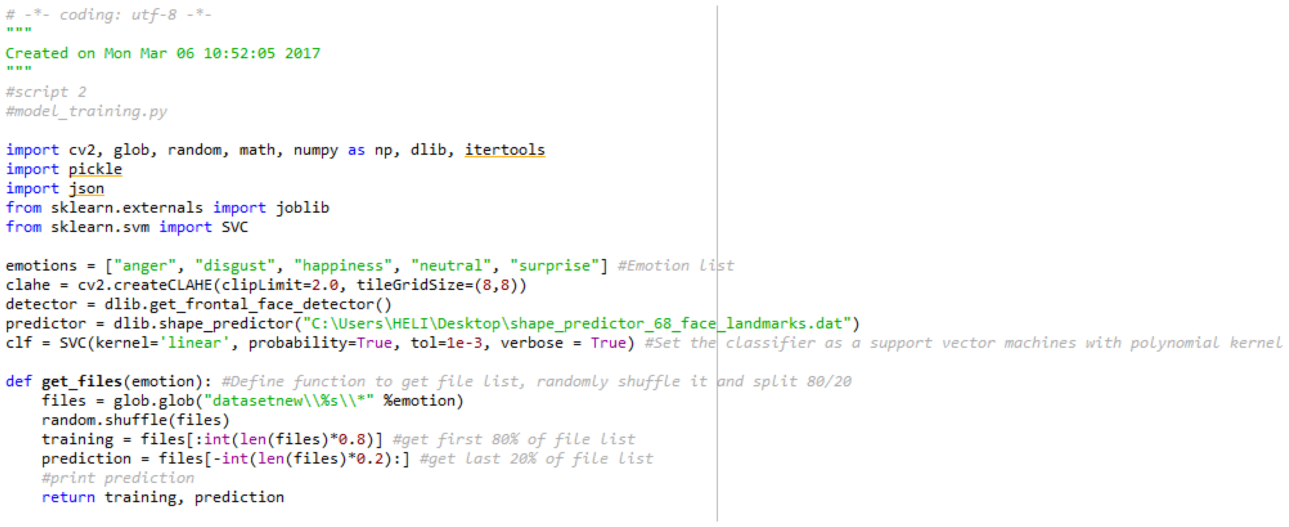
7.4 Code Snippet

Script 1: facial_landmark.py

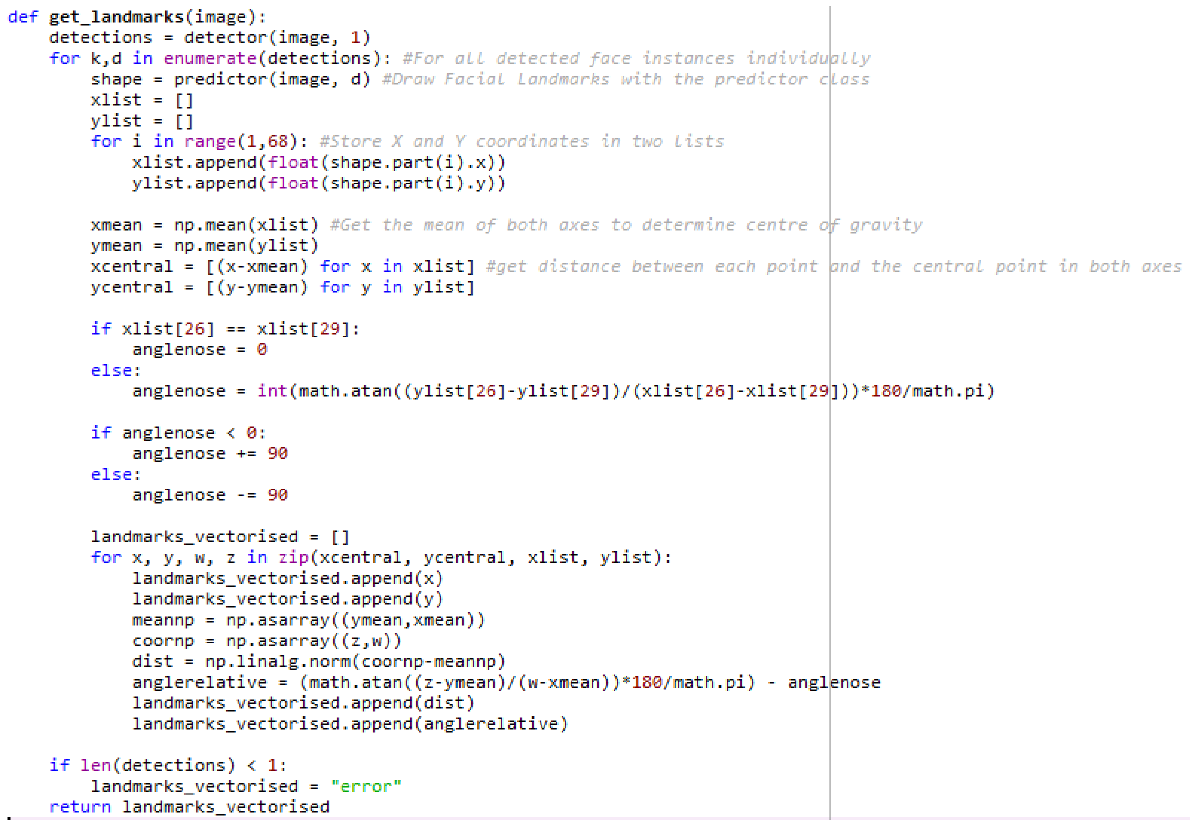


```python
#script 1
#facial_landmark.py
import cv2
from cv2 import WINDOW_NORMAL
import dlib

cv2.namedWindow("image", WINDOW_NORMAL)
video_capture = cv2.VideoCapture(0)
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("C:\Users\HELI\Desktop\shape_predictor_68_face_landmarks.dat")

if video_capture.isOpened():
        read_value, webcam_image = video_capture.read()
else:
        print("webcam not found")

while read_value:
    gray = cv2.cvtColor(webcam_image, cv2.COLOR_BGR2GRAY)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    clahe_image = clahe.apply(gray)

    detections = detector(clahe_image, 1) #Detect the faces in the image

    for k,d in enumerate(detections): #For each detected face

        shape = predictor(clahe_image, d)
        for i in range(1,68):
                cv2.circle(webcam_image, (shape.part(i).x, shape.part(i).y), 1, (0,255,0), thickness=2)

    cv2.imshow("image", webcam_image)
    read_value, webcam_image = video_capture.read()
    key = cv2.waitKey(8)

    if key==27:
        break

cv2.destroyWindow("image")
```

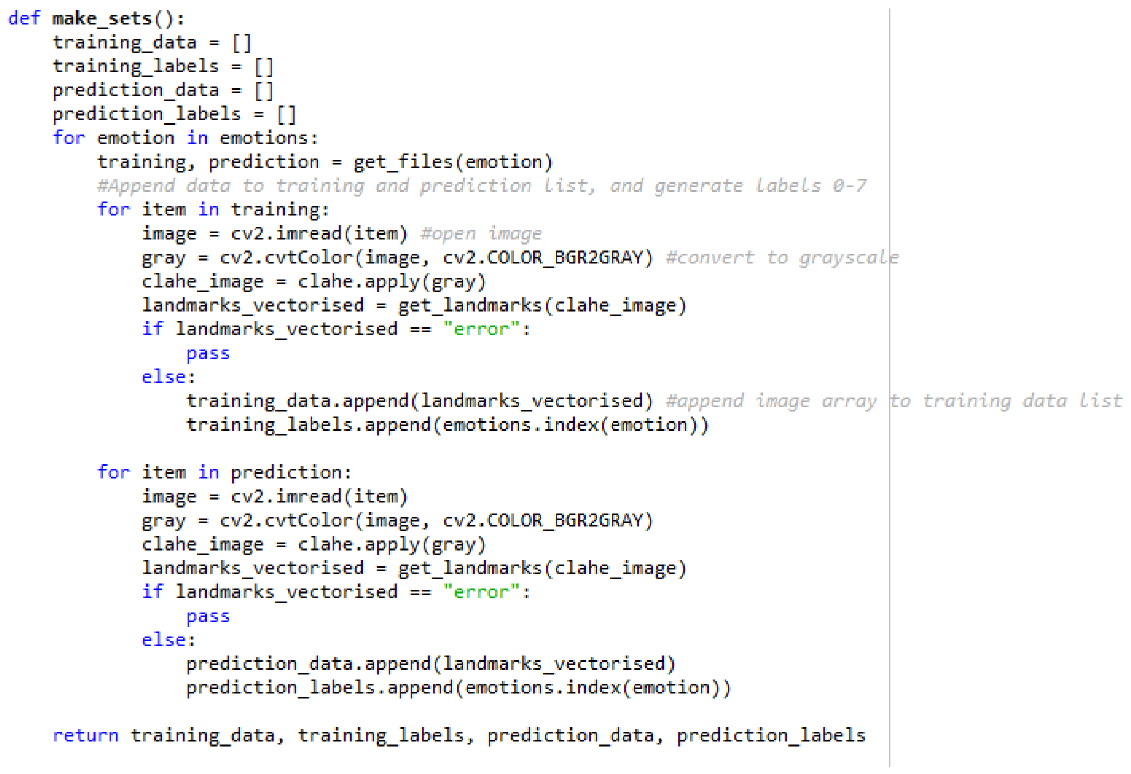
Script 2: model_training.py



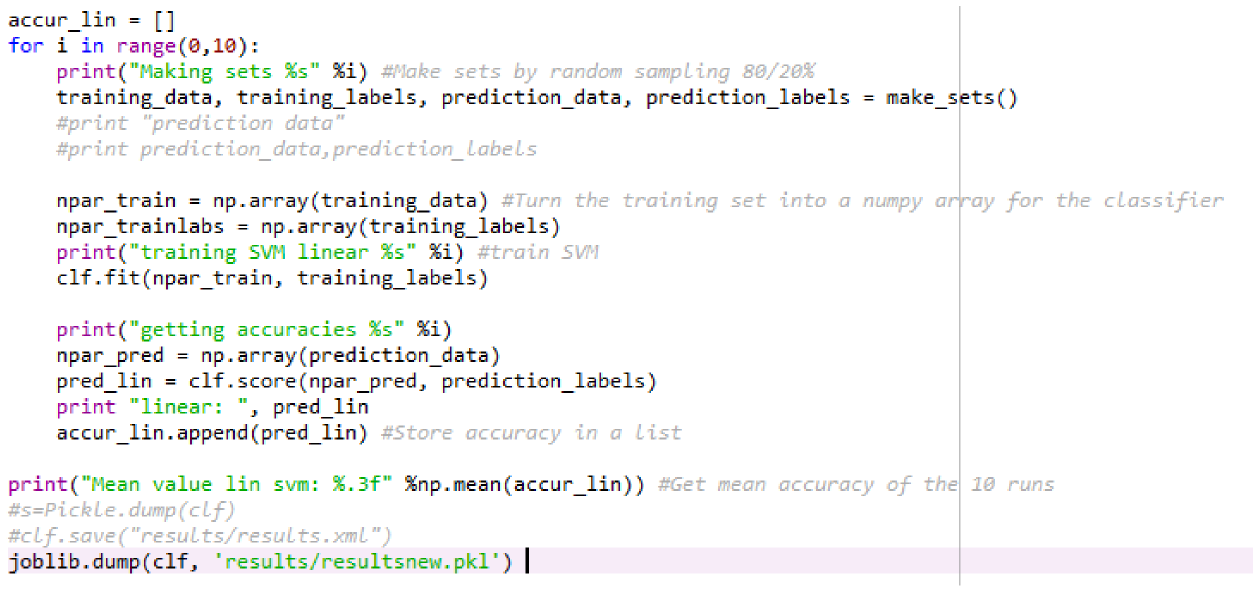
```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 06 10:52:05 2017
"""
#script 2
#model_training.py

import cv2, glob, random, math, numpy as np, dlib, itertools
import pickle
import json
from sklearn.externals import joblib
from sklearn.svm import SVC

emotions = ["anger", "disgust", "happiness", "neutral", "surprise"] #Emotion list
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("C:\Users\HELI\Desktop\shape_predictor_68_face_landmarks.dat")
clf = SVC(kernel='linear', probability=True, tol=1e-3, verbose = True) #Set the classifier as a support vector machines with polynomial kernel

def get_files(emotion): #Define function to get file list, randomly shuffle it and split 80/20
    files = glob.glob("datasetnew\\%s\\*" %emotion)
    random.shuffle(files)
    training = files[:int(len(files)*0.8)] #get first 80% of file list
    prediction = files[-int(len(files)*0.2):] #get last 20% of file list
    #print prediction
    return training, prediction
```



```
def get_landmarks(image):
    detections = detector(image, 1)
    for k,d in enumerate(detections): #For all detected face instances individually
        shape = predictor(image, d) #Draw Facial Landmarks with the predictor class
        xlist = []
        ylist = []
        for i in range(1,68): #Store X and Y coordinates in two lists
            xlist.append(float(shape.part(i).x))
            ylist.append(float(shape.part(i).y))

        xmean = np.mean(xlist) #Get the mean of both axes to determine centre of gravity
        ymean = np.mean(ylist)
        xcentral = [(x-xmean) for x in xlist] #get distance between each point and the central point in both axes
        ycentral = [(y-ymean) for y in ylist]

        if xlist[26] == xlist[29]:
            anglenose = 0
        else:
            anglenose = int(math.atan((ylist[26]-ylist[29])/(xlist[26]-xlist[29]))*180/math.pi)

        if anglenose < 0:
            anglenose += 90
        else:
            anglenose -= 90

        landmarks_vectorised = []
        for x, y, w, z in zip(xcentral, ycentral, xlist, ylist):
            landmarks_vectorised.append(x)
            landmarks_vectorised.append(y)
            meannp = np.asarray((ymean,xmean))
            coornp = np.asarray((z,w))
            dist = np.linalg.norm(coornp-meannp)
            anglerelative = (math.atan((z-ymean)/(w-xmean))*180/math.pi) - anglenose
            landmarks_vectorised.append(dist)
            landmarks_vectorised.append(anglerelative)

    if len(detections) < 1:
        landmarks_vectorised = "error"
    return landmarks_vectorised
```



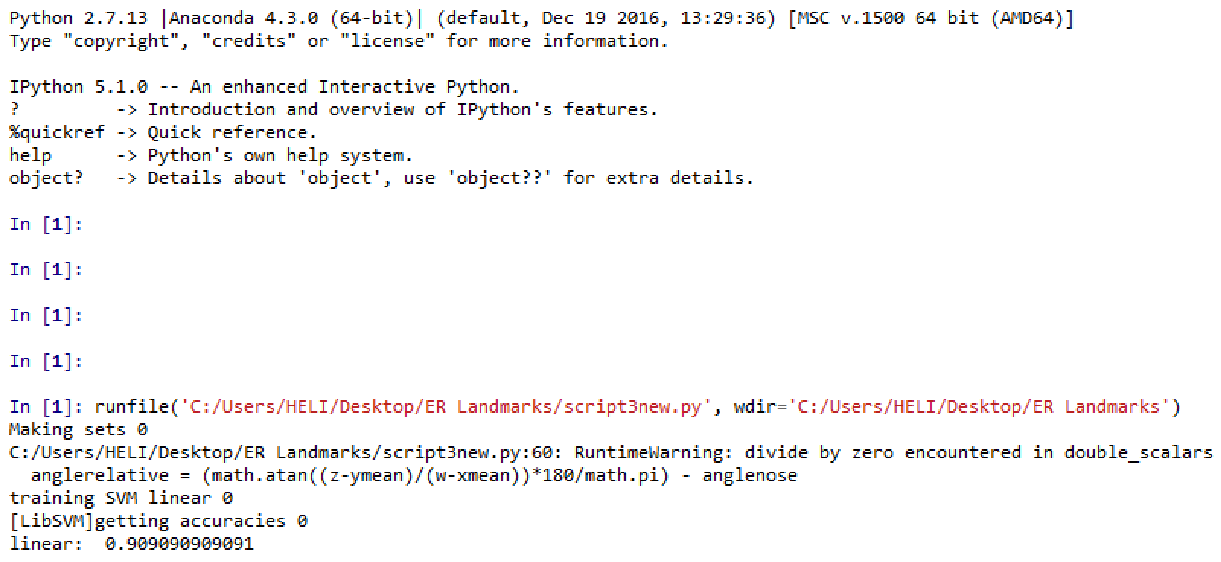
```
def make_sets():
    training_data = []
    training_labels = []
    prediction_data = []
    prediction_labels = []
    for emotion in emotions:
        training, prediction = get_files(emotion)
        #Append data to training and prediction list, and generate labels 0-7
        for item in training:
            image = cv2.imread(item) #open image
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) #convert to grayscale
            clahe_image = clahe.apply(gray)
            landmarks_vectorised = get_landmarks(clahe_image)
            if landmarks_vectorised == "error":
                pass
            else:
                training_data.append(landmarks_vectorised) #append image array to training data list
                training_labels.append(emotions.index(emotion))

        for item in prediction:
            image = cv2.imread(item)
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            clahe_image = clahe.apply(gray)
            landmarks_vectorised = get_landmarks(clahe_image)
            if landmarks_vectorised == "error":
                pass
            else:
                prediction_data.append(landmarks_vectorised)
                prediction_labels.append(emotions.index(emotion))

    return training_data, training_labels, prediction_data, prediction_labels
```



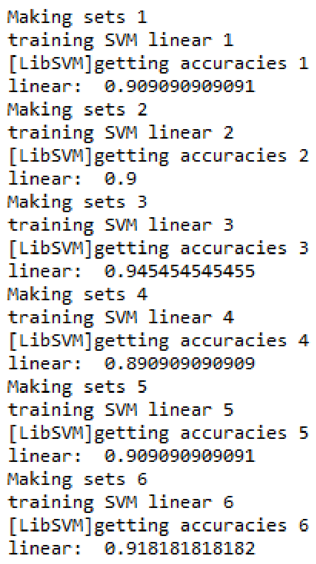
```
accur_lin = []
for i in range(0,10):
    print("Making sets %s" %i) #Make sets by random sampling 80/20%
    training_data, training_labels, prediction_data, prediction_labels = make_sets()
    #print "prediction data"
    #print prediction_data,prediction_labels

    npar_train = np.array(training_data) #Turn the training set into a numpy array for the classifier
    npar_trainlabs = np.array(training_labels)
    print("training SVM linear %s" %i) #train SVM
    clf.fit(npar_train, training_labels)

    print("getting accuracies %s" %i)
    npar_pred = np.array(prediction_data)
    pred_lin = clf.score(npar_pred, prediction_labels)
    print "linear: ", pred_lin
    accur_lin.append(pred_lin) #Store accuracy in a list

print("Mean value lin svm: %.3f" %np.mean(accur_lin)) #Get mean accuracy of the 10 runs
#s=Pickle.dump(clf)
#clf.save("results/results.xml")
joblib.dump(clf, 'results/resultsnew.pkl')
```

7.5 Observation

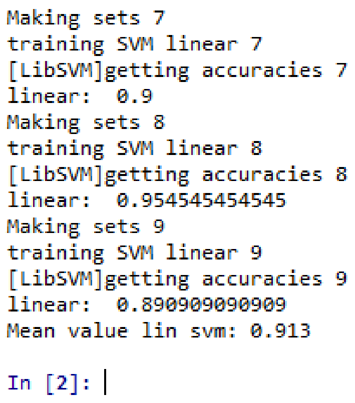
With 5 emotions, the classifier gave 91.3% correct result in predicting the dataset.



```
Python 2.7.13 |Anaconda 4.3.0 (64-bit)| (default, Dec 19 2016, 13:29:36) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:

In [1]:

In [1]:

In [1]:

In [1]: runfile('C:/Users/HELI/Desktop/ER Landmarks/script3new.py', wdir='C:/Users/HELI/Desktop/ER Landmarks')
Making sets 0
C:/Users/HELI/Desktop/ER Landmarks/script3new.py:60: RuntimeWarning: divide by zero encountered in double_scalars
  anglerelative = (math.atan((z-ymean)/(w-xmean))*180/math.pi) - anglenose
training SVM linear 0
[LibSVM]getting accuracies 0
linear:  0.909090909091
```



```
Making sets 1
training SVM linear 1
[LibSVM]getting accuracies 1
linear:  0.909090909091
Making sets 2
training SVM linear 2
[LibSVM]getting accuracies 2
linear:  0.9
Making sets 3
training SVM linear 3
[LibSVM]getting accuracies 3
linear:  0.945454545455
Making sets 4
training SVM linear 4
[LibSVM]getting accuracies 4
linear:  0.890909090909
Making sets 5
training SVM linear 5
[LibSVM]getting accuracies 5
linear:  0.909090909091
Making sets 6
training SVM linear 6
[LibSVM]getting accuracies 6
linear:  0.918181818182
```



```
Making sets 7
training SVM linear 7
[LibSVM]getting accuracies 7
linear:  0.9
Making sets 8
training SVM linear 8
[LibSVM]getting accuracies 8
linear:  0.954545454545
Making sets 9
training SVM linear 9
[LibSVM]getting accuracies 9
linear:  0.890909090909
Mean value lin svm: 0.913

In [2]: |
```

7.6 Screenshots

Figure 22, shows the output of script 1. The script successfully runs to display the facial landmarks of the subjects. These facial landmark dots are useful for emotion classification.

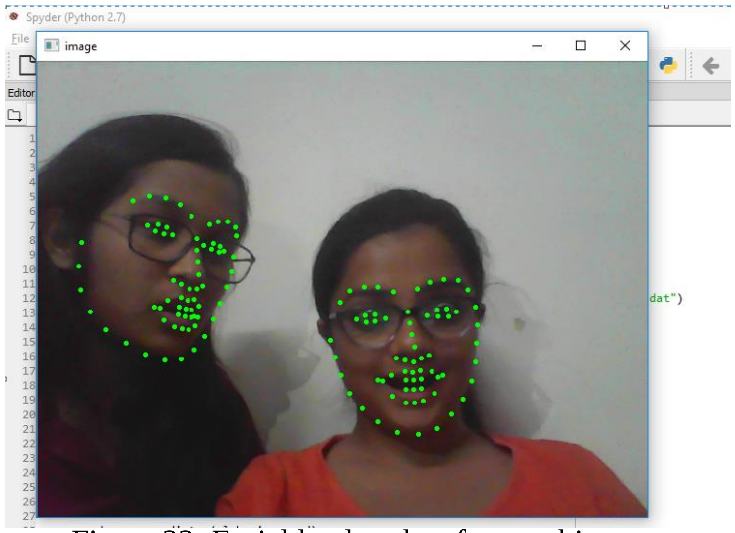
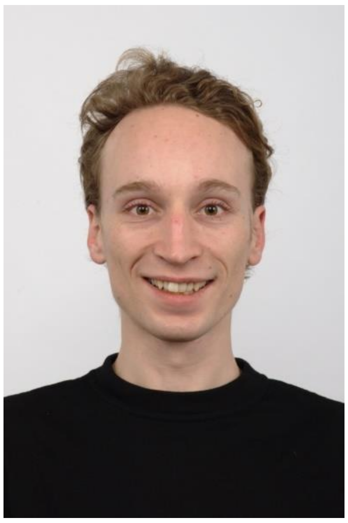


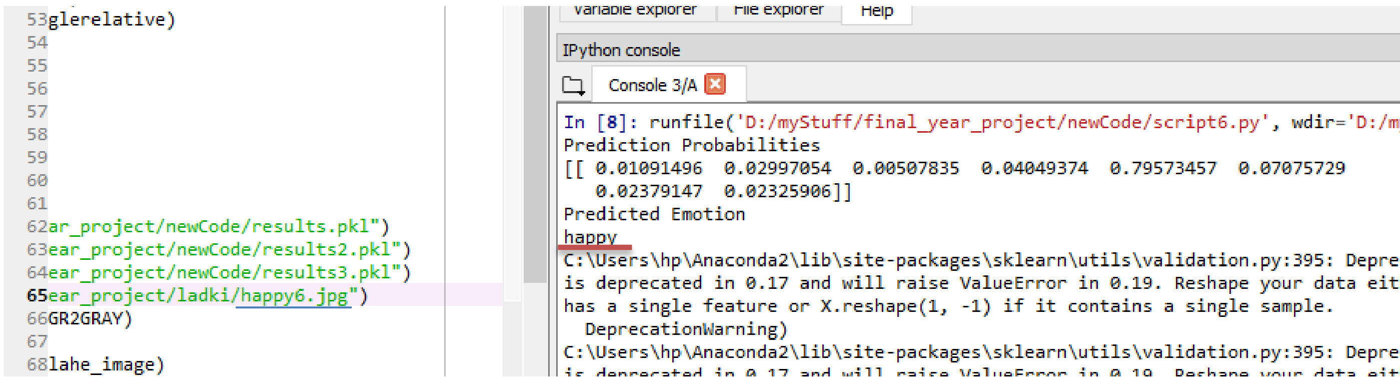
Figure 22: Facial landmarks of two subjects

Outputs using static image as input:

1) For happy face: Output is “happy”

The following are the screenshots of the image used for prediction of emotion and the emotion predicted. Also, the enlisted are the probabilities of the image falling into various emotion categories. The maximum probability is 0.79573457, which corresponds to class “happy”. Hence the emotion is classified to be Happy.





```
glerelative)








ar_project/newCode/results.pkl")
ear_project/newCode/results2.pkl")
ear_project/newCode/results3.pkl")
ear_project/ladki/happy6.jpg")
GR2GRAY)

lahe_image)
```

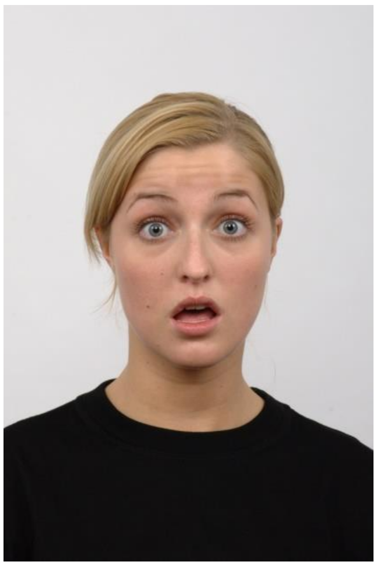
```
In [8]: runfile('D:/myStuff/final_year_project/newCode/script6.py', wdir='D:/m
Prediction Probabilities
[[ 0.01091496  0.02997054  0.00507835  0.04049374  0.79573457  0.07075729
   0.02379147  0.02325906]]
Predicted Emotion
happy
C:\Users\hp\Anaconda2\lib\site-packages\sklearn\utils\validation.py:395: Depre
is deprecated in 0.17 and will raise ValueError in 0.19. Reshape your data eit
has a single feature or X.reshape(1, -1) if it contains a single sample.
  DeprecationWarning)
C:\Users\hp\Anaconda2\lib\site-packages\sklearn\utils\validation.py:395: Depre
is deprecated in 0.17 and will raise ValueError in 0.19. Reshape your data eit
```

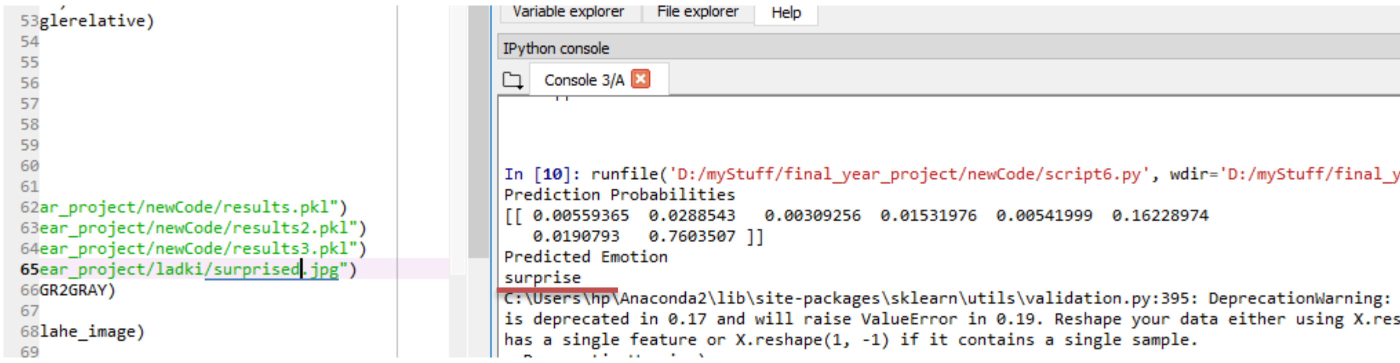
Also, the probability of image to fall in various categories is:

Anger: 0.01091496	Contempt: 0.02997054
Disgust: 0.00507835	Fear: 0.04049374
Happy: 0.79573457 (max)	Neutral: 0.07075729
Sadness: 0.02379147	Surprise: 0.02325906

2) For surprised face: Output is “surprise”

The following are the screenshots of the image used for prediction of surprise emotion and the emotion predicted. Also, the enlisted are the probabilities of the image falling into various emotion categories. The maximum probability is 0.7603507, which corresponds to class “surprise”. Hence the emotion is classified to be Surprise.





```
glerelative)








ar_project/newCode/results.pkl")
ear_project/newCode/results2.pkl")
ear_project/newCode/results3.pkl")
ear_project/ladki/surprised.jpg")
GR2GRAY)

lahe_image)
```

```
In [10]: runfile('D:/myStuff/final_year_project/newCode/script6.py', wdir='D:/myStuff/final_y
Prediction Probabilities
[[ 0.00559365  0.0288543   0.00309256  0.01531976  0.00541999  0.16228974
   0.0190793   0.7603507 ]]
Predicted Emotion
surprise
C:\Users\hp\Anaconda2\lib\site-packages\sklearn\utils\validation.py:395: DeprecationWarning:
is deprecated in 0.17 and will raise ValueError in 0.19. Reshape your data either using X.res
has a single feature or X.reshape(1, -1) if it contains a single sample.
```

Also, the probability of image to fall in various categories is:

Anger: 0.00559365	Contempt: 0.0288543
Disgust: 0.00309256	Fear: 0.01531976
Happy: 0.00541999	Neutral: 0.16228974
Sadness: 0.0190793	Surprise: 0.7603507 (max)

7.7 Comparison

For the standard set at 5 categories we managed to get 88.1% accuracy with FisherFace classifier. SVM approach yields 91.3% on the same data and hence the improvement.

Comparison between FisherFace (FLDA) and SVM:

1. LDA tries to maximise the distance between the means of the two groups, while SVM tries to maximise the margin between the two groups.

2. LDA is generative, SVM is discriminative.

3. SVM classification is an optimization problem, LDA has an analytical solution. The optimization problem for the SVM has a dual and a primal formulation that allows the user to optimize over either the number of data points or the number of variables, depending on which method is the most computationally feasible.

4. SVM focuses only on the points that are difficult to classify, LDA focuses on all data points. Such difficult points are close to the decision boundary and are called Support Vectors.

5. LDA assumes that the data points have the same covariance and the probability density is assumed to be normally distributed. SVM has no such assumption.

SVM makes no assumptions about the data at all, meaning it is a very flexible method. The flexibility on the other hand often makes it more difficult to interpret the results from a SVM classifier, compared to LDA.

Chapter 8 - Conclusion and Future Work

8.1 Conclusion

This report focuses on the various studies conducted on the subject of Face detection, Feature extraction and Emotion recognition and also on the implementation of an emotion recognition application. With implementation using Haar cascade and fisherface we obtained 88.1% accuracy. Whereas, implementation using facial landmarks and Linear SVM gave us 91.3% accuracy. Also, one advantage of the SVM classifier we use is that it is probabilistic. This means that it assigns probabilities to each category it has been trained for. Since, the dataset we have worked on is small, containing only about 1000 images spread over different categories, the experiments need to be performed with diverse dataset.

8.2 Future Work

We would like to extend this implementation by training the classifiers with more diverse datasets for improving the classification accuracies. Also, we would like to integrate this python project with Android Studio. Using python with OpenCV is easier than using Android with Open CV. Python supports more features as compared to android.

We are planning to integrate python with android studio using Kivy. Kivy is an open source cross platform Python library for developing android apk.

Buildozer is a generic Python packager for Android and iOS. Buildozer is a tool that automates the entire build process. It downloads and sets up all the prequisites for python-for-android, including the android SDK and NDK, then builds an apk that can be automatically pushed to the device.

Another option would be to use QPython. Qpython is a script engine which runs python programs on android device and also helps in developing android applications.

Chapter 9 - References

[1] Yang, Ming-Hsuan, David J. Kriegman, and Narendra Ahuja. "Detecting faces in images: A survey." IEEE Transactions on pattern analysis and machine intelligence 24.1 (2002): 34-58.

[2] Ion Marqu´es, “Face Recognition Algorithms”, University of the Basque Country, June 16, 2010, [http://www.ehu.eus/ccwintco/uploads/e/eb/PFC-IonMarques.pdf]

[3] “Facial Expression Pictures Chart & Facial Movements”, 3rd May, 2016, [https://imotions.com/blog/facial-expression-pictures]

[4] Hemalatha, G., and C. P. Sumathi. "A study of techniques for facial detection and expression classification." International Journal of Computer Science and Engineering Survey 5.2 (2014): 27.

[5] Jain, Anil K., Robert P. W. Duin, and Jianchang Mao. "Statistical pattern recognition: A review." IEEE Transactions on pattern analysis and machine intelligence 22.1 (2000): 4-37.

[6] B. Moghaddam, T. Jebara, and A. Pentland. “Bayesian face recognition. Pattern Recognition”, 33(11):1771–1782, November 2000.

[7] E. Hjelmas, B. K. Low. “Face detection: A Survey”, Department of Meteorology, University of Edinburgh, Scotland, United Kingdom April 17, 2001

[8]Jo Chang-yeon. “Face detection using LBP features”, December 12, 2008

[9] Shan, Caifeng, Shaogang Gong, and Peter W. McOwan. "Robust facial expression recognition using local binary patterns." Image Processing, 2005. ICIP 2005. IEEE International Conference on. Vol. 2. IEEE, 2005.

[10] Local Binary Patterns In Wikipedia. Retrieved October 20, 20016, [https://en.wikipedia.org/wiki/Local_binary_patterns]

[11] Paul Viola, Michael Jones. “Rapid object detection using a boosted cascade of simple features” Computer Vision and Pattern Recognition Conf, 2001. .

[12] Sakai, Toshiyuki, Makoto Nagao, and Shinya Fujibayashi. "Line extraction and pattern detection in a photograph." Pattern recognition 1.3 (1969): 233IN9237-236IN12248.

[13] Chen, Li-Fen, et al. "A new LDA-based face recognition system which can solve the small sample size problem." Pattern recognition 33.10 (2000): 1713-1726.

[14]About OpenCV, 2017 [http://opencv.org/about.html]

[15] Jamal Eason, Android developers blog, 19 September, 2016
[http://android-developers.blogspot.in/2016/09/android-studio-2-2.html]

[16] Christine L. Lisetti, Fatma Nasoz, “Affective Intelligent Car Interfaces with Emotion Recognition”, 11th Human Computer Interaction Int. Conf., Las Vegas, NV, USA, July 2005

[17] Aleix Martinez (2011) Fisherfaces. Scholarpedia, 6(2):4282.

[18]Belhumeur, Peter N., João P. Hespanha, and David J. Kriegman. "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection." IEEE Transactions on pattern analysis and machine intelligence 19.7 (1997): 711-720.

[19]Y. Tie and L. Guan. Automatic landmark point detection and tracking for human facial expressions. EURASIP Journal on Image and Video Processing, 8, 2013.DOI: 10.1186/1687-5281-2013-8

[20]C.Sindhuja, Dr.K.Mala. "LANDMARK IDENTIFICATION IN 3D IMAGE FOR FACIAL EXPRESSION RECOGNITION". 2015 International Conference on Computing and Communications Technologies (ICCCT’15). DOI: 10.1109/ICCCT2.2015.7292772.

[21] Sunil Ray, “Understanding Support Vector Machine algorithm from examples”, OCTOBER 6, 2015 [https://www.analyticsvidhya.com/blog/2015/10/understaing-support-vector-machine-example-code/]

[22] van Gent, P. (2016). Emotion Recognition With Python, OpenCV and a Face Dataset. A tech blog about fun things with Python and embedded electronics. Retrieved from: http://www.paulvangent.com/2016/04/01/emotion-recognition-with-python-opencv-and-a-face-dataset/

[23] Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG’00), Grenoble, France, 46-53.

[24] van Gent, P. (2016). Emotion Recognition Using Facial Landmarks, Python, DLib and OpenCV. A tech blog about fun things with Python and embedded electronics. Retrieved from: http://www.paulvangent.com/2016/08/05/emotion-recognition-using-facial-landmarks

Acknowledgement

We are highly indebted to Dr. M. A. Zaveri for his guidance and constant supervision as well as for providing necessary information regarding the project. Also, we would like to extend a sincere and heartfelt obligation towards all the personages who have helped us in this endeavour. Without their active guidance, help, co-operation and encouragement, we could not have made headway in the project.