

COMP-5413-FB (Topics in computer vision)

Final Project

Topic: Attendance taker by scanning faces

Faculty: Dr. Shan Du

Team Members:

Jaykumar Nariya (1116571)

Parth Valani (1116576)

ABSTRACT

The face is one of the easiest ways to distinguish the individual identity of each other. Face recognition is a personal identification system that uses personal characteristics of a person to identify the person's identity.

Human face recognition procedure basically consists of two phases, namely face detection, where this process takes place very rapidly in humans, except under conditions where the object is located at a short distance away, the next is the introduction, which recognize a face as individuals. Stage is then replicated and developed as a model for facial image recognition (face recognition) is one of the much-studied biometrics technology and developed by experts.

Human face detection and recognition play important roles in many applications such as video surveillance and face image database management. In our project, we have studied worked on both face recognition and detection techniques and developed algorithms for them.

In face recognition the algorithm used is LBPH (Local Binary Pattern Histogram) in which we recognize an unknown test image by comparing it with the known training images stored in the database as well as give information regarding the person recognized. These techniques works well under robust conditions like complex background, different face positions.

Table of Content

1.INTRODUCTION

- 1.1. Introduction
- 1.2. Objective
- 1.3. Literature review
- 1.4. Methodologies

2. CHALLANGES

3. LOCAL BINARY HISTOGRAM PATTERNS

- 3.1. Texture Descriptor
- 3.2. Local Binary Patterns Extension
- 3.3. Uniform Patterns

4. LOCAL BINARY PATTERNS APPLIED TO FACE DETECTION

- 4.1. Image Block Scanning Method
- 4.2. Feature Extraction for Two Stages Face Detection Scheme
 - 4.2.1 Coarse Stage Feature Extraction
 - 4.2.2 Fine Stage Features Extraction
- 4.3. Face Detection Classification
- 4.4. Application to Face Detection
- 4.5. SVM applied to face detection classification
 - 4.5.1. PCA before applying SVM
- 4.6. Face Detection System Improvements
 - 4.6.1. Training Stage
 - 4.6.1.1. Bootstrap strategy to non-face pattern reallocation
 - 4.6.2. Image Preprocessing
 - 4.6.2.1. Test
- 4.7. Overlapping detection removal
 - 4.7.1. Overlapping Detection Removal Technique

5. LOCAL BINARY PATTERNS APPLIED TO FACE RECOGNITION

- 5.1. Image Preprocessing
- 5.2. Face Descriptor Extraction
- 5.3. Face Identification
 - 5.3.1. Face Regions Analysis
- 5.4. Face Recognition Classification

5.5. LBPH(Local Binary Pattern Histogram)

5.5.1. Objective

5.5.2. Introduction

5.5.3. Step-by-Step

5.5.3.1. Parameters

5.5.3.2. Training the algorithm

5.5.3.3. Applying the LBP Operation

5.5.3.4. Extracting the histogram

5.5.3.5. Performing Face Recognition

6. CODE

7. EXPERIMENTAL RESULTS

8. COMPARING LBP WITH OTHER DESCRIPTOR

9. CONCLUSION

10. REFERENCES

CHAPTER – 1. INTRODUCTION

Face detection and recognition are playing a very important role in our current society, due to their use for a wide range of applications such as surveillance, banking and multimedia equipment as cameras and video game consoles to name just a few.

A face recognition system can be used in two modes: verification (or authentication) and identification. A face verification system involves confirming or denying the identity claimed by a person (one-to-one matching).

On the other hand, a face identification system attempts to establish the identity of a given person out of a pool of N people (one-to- N matching). When the identity of the person may not be in the database, this is called open set identification. While verification and identification often share the same classification algorithms, both modes target distinct applications.

In verification mode, the main applications concern access control, such as computer or mobile device log-in, building gate control, digital multimedia data access. Over traditional security access systems, face verification has many advantages: the biometric signature cannot be stolen, lost or transmitted, like for ID card, token, badges or forgotten like passwords or PIN codes. In identification mode, potential applications mainly involve video surveillance (public places, restricted areas), information retrieval (police databases, multimedia data management) or human computer interaction (video games, personal settings identification).

In addition, most consumer electronic devices such as mobile phones, laptops, video game consoles and even televisions include a small camera enabling a wide range of image processing functionalities including face detection and recognition applications.

For instance, a renowned TV manufacturer has built-in a camera to some of their television series to make a new feature called Intelligent Presence Sensor possible. The users' presence is perceived by detecting faces, motion, position and even age, in the area in front of the television and after a certain time with no audience, the set turns off automatically, thus saving both energy and TV life.

On the other hand, other demanding applications for face detection and recognition

are in the field of automatic video data indexing to cope with the increase of digital data storage. For example, to assist in the indexing of huge television contents databases by automatically labeling all videos containing the presence of a given individual.

In a similar way, face detection and recognition techniques are helpful for Web Search Engines or consumers' picture organizing applications in order to perform automatic face image searching and tagging. For instance, Google's Picasa digital image organizer has a built-in face recognition system that can associate faces with people, so that queries can be run on pictures to return all images with a specific group of people together.

Another example is iPhoto, a photo organizer distributed with iLife that uses face detection to identify faces of people in photos and face recognition to match faces that look like the same person. After four decades of research and with today's wide range of applications and new possibilities, researchers are still trying to find the algorithm that best works in different illuminations, environments, over time and with minimum error.

This work pretends to explore the potential of a texture descriptor proposed by researches from the University of Oulu in Finland [1]-[7]. This texture face descriptor is called Local Binary Patterns (LBP) and the main motivations to study it in this work are:

- Low computation complexity of the LBP operator.
- It can be applied for both detection and recognition.
- Robustness to pose and illumination changes.

Local Binary Patterns were first used in order to describe ordinary textures where the spatial relation was not as significant as it is for face images. A face can be seen as a composition of micro textures depending on the local situation. The LBP is basically divided into two different descriptors: a global and a local. The global is used for discriminating the most nonface objects (blocks), whereas the second provides specific and detailed face information which can be used not only to select faces, but also to provide face information for recognition.

So the whole descriptor consists of a global texture and a local texture representation calculated by dividing the image into blocks and computing the texture histogram for each one. The results will be concatenated in a general descriptor vector. In such

representation, the texture of facial regions is encoded by the LBP while the shape is recovered by the concatenation of different local histograms.

Afterwards, the feature vector will be used to feed an adequate classifier or discriminative scheme to decide the fakeness (a measure to determine the similarity of an object to a human face) of the input image or the identity of the input face in case of face recognition.

In the following chapters, a complete system using Local Binary Patterns in both face detection and recognition stages will be detailed, implemented and evaluated.

Chapter 2 provides the most important challenges to better understand the research motivations.

Chapter 3 deals with the Local Binary Patterns operator, detailing how to compute it, interpret it and why it is useful for face description.

In Chapter 4 and 5, the theory for LBP application to Face Detection and Recognition respectively is discussed. The fourth chapter exposes the detection scheme proposal, different classification methods and finally some system improvements such as the use of skin segmentation pre-stage for detection algorithm optimization. In a similar way, the fifth chapter describes the image pre-processing phase for face recognition and the proposed identification scheme.

Before beginning with the empirical phase, Chapter 6 is Experiment Code. Chapter 7 is Experiment result, Chapter 8 and Chapter 9 Comparison with others and conclusion.

Finally, the Appendix section contains useful additional information to complete and to better understand this research work, such as more detailed information about the image databases, results tables and a description of the used algorithms.

1.2 Objectives

This software provides a reliable way to take attendance through face detection automatically. This forum was designed for automation in Education Department with a goal of innovation in existing fields.

Our aim is to avoid the discrepancy of attendance in the schools and universities.

This will not only help teachers and students to make the attendance taking process easier but also help them to keep the records in the databases through creating excel sheets of respective faculty.

However, this will not only avoid the controversies but also increase the ratio in attendance comparatively to the current one.

The Project stands for Taking attendance through scanning faces. This project is implemented to take a one step in a innovation of Education Department.

This Project can also be used in other Government Department and Private Sectors. Which can decrease conspiracies about presence in the offices or Schools or Colleges.

1.3 Literature Review

The availability of numerous commercial face recognition systems attests to the significant progress achieved in the research field. Despite these achievements, face recognition continues to be an active topic in computer vision research. This is because current systems perform well under relatively controlled environments but tend to suffer when variations in different factors (such as pose, illumination etc.) are present. Therefore, the goal of the ongoing research is to increase the robustness of the systems against different factors. Ideally, we aim to develop a face recognition system which mimics the remarkable capabilities of human visual perception. Before attempting to reach such a goal, one needs to continuously learn the strengths and weaknesses of the proposed techniques in order to determine new directions for future improvements. In this work, we present a novel approach to face recognition which considers both shape and texture information to represent face images. The face area is first divided into small regions from which Local Binary Pattern (LBP) histograms are extracted and concatenated into a single, spatially enhanced feature histogram efficiently representing the face image. The recognition is performed using a nearest neighbor classifier in the computed feature space with Chi square as a dissimilarity measure. Extensive experiments clearly show the superiority of the proposed scheme over all considered methods (PCA, Bayesian Intra/extrapersonal Classifier and Elastic Bunch Graph Matching) on FERET tests which include testing the robustness of the method against different facial expressions, lighting and aging of the subjects. In addition to its efficiency, the simplicity of the proposed method allows for very fast feature extraction.

1.4 Methodologies

We use Local binary pattern as Feature Extraction and cascade classifier as model to recognize face either its faculty or student. Because of below reasons:

1. The most important property of the LBP operator in real-world applications is its tolerance against illumination changes.
2. The advantage of using cascade is that it is extremely fast for Training as real time training is most important in this feature.
3. The process took around 25 images/sec.
4. No GPU required to train model.
5. Cascade is basically a classifier which is used to detect the object for which it has been trained from the source.
6. Cascade is trained by superimposing the positive image over a set of negative images.
7. Better results are obtained by using high quality images and increasing the amount of stages for which the classifier is trained.

As we all know that we can use any pretrained model like any of CNNs for better accuracy. But in this project, we have to generate student data on real time and train that model real time is very important. Also, Cascade classifiers compared to CNNs, have the advantage of having the ability to perform detection and classification simultaneously, which eliminates the need to design an additional detection algorithm that extracts the image from the tool and enters it into a classifier and cascade classifiers do not require changes in their structure to improve the quality of recognition, as it does when defining the architecture of a CNN, but a variation in their parameters to change the number of stages and positive and negative training images, making it easier to implement than a CNN.

Time of training is bigger factor in this project. That's why we choose first cascade classifier to recognize face.

CHAPTER - 2

CHALLENGES

In order to better understand the face detection and recognition task and its difficulties, the following factors must be taken into account, because they can cause serious performance degradation in face detection and recognition systems:

- Pose: face images vary due to relative position between camera and face and some facial features like the eyes or the nose can be partially or totally hidden.
- Illumination and other image acquisition conditions: face aspect in an image can be affected by factors such as illumination variations, in its source distribution and intensity, and camera features such as sensor response and lenses. Differences produced by these factors can be greater than differences between individuals.
- Occlusions: faces can be partially occluded by other objects (beards, moustaches and glasses) and even by faces from other people.
- Facial expressions: facial appearance can be directly affected by the facial expression of the individual.

Moreover, the following factors are specific from face detection task and basically related to computational efficiency that demands, for example, to spend as shortest time as possible exploring non-face image areas.

- Faces per image: normally faces rarely appear in images. 0 to 10 faces are to be found in an image.
- Scanning areas: the sliding window for image exploration evaluates a huge number of location and scale combinations and directly depends on the selected:
 - scanning step (for location exploration)
 - down sampling ratio (for scale

exploration) On the other hand, face recognition

specific challenges:

- Images taken years apart: facial appearance can considerably change over the years, driving to *False Reject* errors that can provoke the access deny of legitimate users.
- Similar identities: biometric recognition systems are susceptible of *False Accept* errors where an impostor with a similar identity can be accepted as a legitimate user.

In some case, the training data used for face detection and recognition systems are frontal view face images, but obviously, some difficulties appear when trying to analyze faces with different pose angles. Therefore, some processing schemes introduce rotated images in their training data set and other try to identify particular features of the face such as the eyes, mouth and nose, in order to improve positive ratios.

After giving a general overview of the different methods, this work has been focused on a technique based in the use of the LBP operator which can be used for both face detection and recognition, motivated by its robustness to pose and illumination changes and due to its low computation complexity. Next chapter will explain in detail how LBP can be used for both tasks.

CHAPTER - 3

LOCAL BINARY PATTERNS

This chapter introduces a discriminative feature space that can be applied for both face detection and recognition challenges, motivated by its invariance with respect to monotonic gray scale and the fact that it can be extracted in a single scan through the whole image.

Local Binary Patterns (LBP) is a texture descriptor that can be also used to represent faces, since a face image can be seen as a composition of micro-texture-patterns.

Briefly, the procedure consists of dividing a facial image in several regions where the LBP features are extracted and concatenated into a feature vector that will be later used as facial descriptor.

3.1. Texture Descriptor

The LBP originally appeared as a generic texture descriptor. The operator assigns a label to each pixel of an image by thresholding a 3x3 neighborhood with the centre pixel value and considering the result as a binary number. In different publications, the circular 0 and 1 resulting values are read either clockwise or counter clockwise. In this research, the binary result will be obtained by reading the values clockwise, starting from the top left neighbor, as can be seen in the following figure.

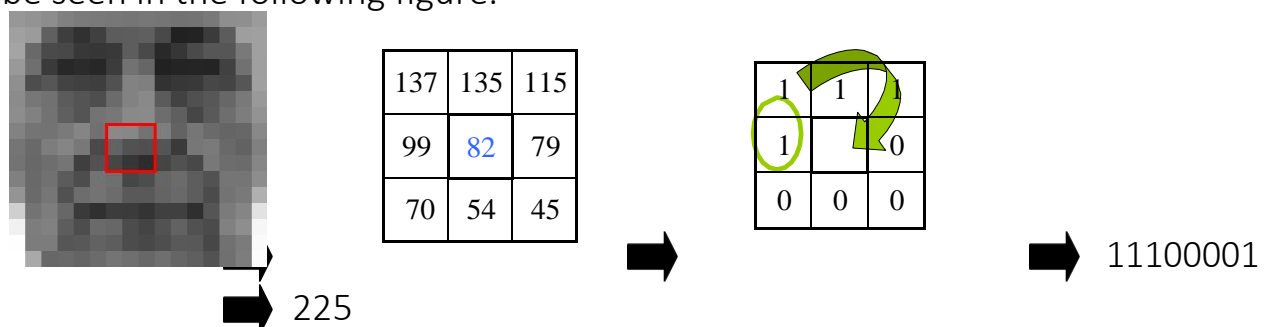


Figure 1. LBP labeling: binary label is read clockwise starting from top left neighbor.

In other words, given a pixel position (x_c, y_c) , LBP is defined as an ordered set of binary comparisons of pixel intensities between the central pixel and its surrounding pixels.

3.2. Local Binary Patterns Extension

In order to treat textures at different scales, the LBP operator was extended to make use of neighborhoods at different sizes. Using circular neighborhoods and bilinear interpolation of the pixel values, any radius and number of samples in the neighborhood can be handled. Therefore, the following notation is defined:

(P, R) which means P sampling points on a circle of R

radius. The following figure shows some examples of different sampling points and radius:

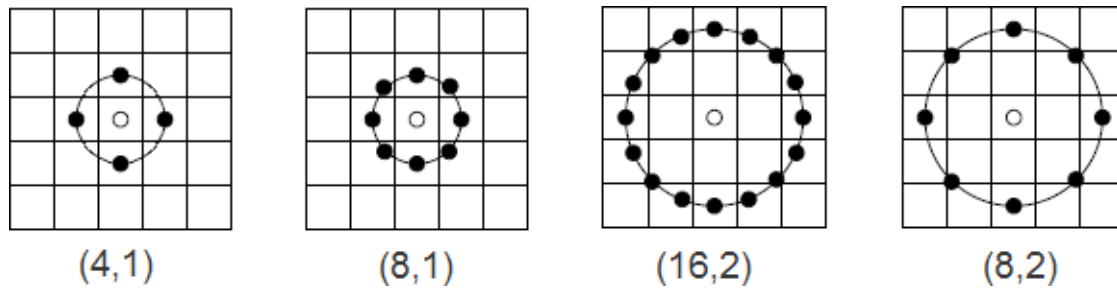


Figure 2. LBP different sampling point and radius examples.

In $LBP(4,1)$ case, the reason why the four points selected correspond to vertical and horizontal ones, is that faces contain more horizontal and vertical edges than diagonal ones.

When computing pixel operations taking into account $N \times N$ neighborhoods at the boundary of an image, a portion of the $N \times N$ mask is off the edge of the image. In such situations, different padding techniques are typically used such as zero-padding, repeating border elements or applying a mirror reflection to define the image borders. Nevertheless, in LBP operator case, the critical boundary, defined by the radius R of the circular operation, is not solved by using a padding technique, instead of that, the operation is started at image

pixel (R, R). The advantage is that the final LBP labels histogram will be not influenced by the borders, although the resulting LBP labels image size will be

reduced to (Width-R)x(Height-R) pixels.

3.3. Uniform Patterns

Ojala [7], in his work *Multiresolution gray-scale and rotation invariant texture classification with Local Binary Patterns*, reveals that it is possible to use only a subset of the 2^P local binary patterns to describe textured images. This subset is called uniform patterns or fundamental patterns.

A LBP is called uniform if the circular binary pattern (clockwise) contains maximal 2 transitions from 0 to 1 and vice versa.

Circular Pattern	Binary	# of bitwise transitions	Uniform pattern?
11111111		0	Yes
00001111		1	Yes
01110000		2	Yes
11001110		3	No
11001001		4	No

Table 1. Example of uniform and non uniform Local Binary Patterns

Each of these patterns has its own bin in the LBP histogram. The rest of the patterns with more than 2 transitions will be accumulated into a single bin. In our experiments the non uniform patterns are accumulated into bin 0.

CHAPTER - 4

LOCAL BINARY PATTERNS APPLIED TO FACE DETECTION

In this chapter, *Local Binary Patterns* application to face detection is considered, as the facial representation in a face detection system.

In order to improve algorithm efficiency and speed, two stages face detection scheme will be introduced in next *section 4.2*: a first coarse stage to preselect face candidates and a second fine stage to finally determinate the faceness of the input image.

Following *Figure 6* introduces the generic face detection scheme proposed for this research project, including:

- Training stage (top box): faces and non-faces training samples are introduced in the system, and feature vectors for 1st coarse and 2nd fine stages are calculated in parallel and later concatenated in a unique *Enhanced Features Vector* of 203-bin to describe each face and non-face image sample. Then, all these results will be used to generate a mean value model for each class.
- Test stage (bottom box): for each new test image, skin segmentation pre-processing is firstly applied to improve face detection efficiency. Then the result will feed 1st coarse classification stage and only face candidates will go through 2nd fine classification stage. Just test images with positive results in both stages will be classified as faces.

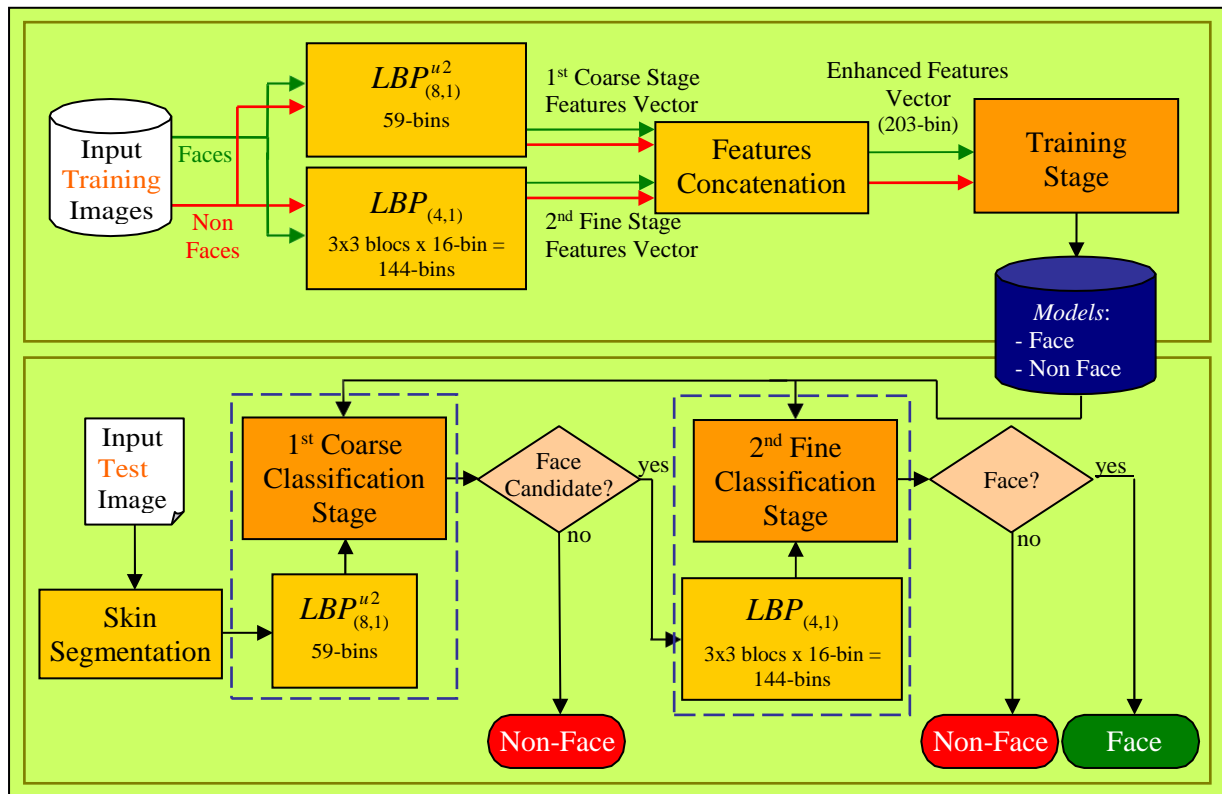


Figure 6. Face Detection Scheme

Above block diagram is intended to give an overview of the face detection system, although some processes has been overlooked to simplify it, such as the input image scanning and down-sampling (see section 4.1 and the final overlapping detection removal process (see section 4.4.2.2).

In each stage, a different part of a 203-bin LBP feature vector is used together with an appropriate classifier. This chapter also introduces different classification methods in order to study LBP face representation potential.

4.1. Image block scanning method

Input image exploration for face candidate searching is done by means of a window-sliding technique applied at a different image scales. As a result, faces can be detected at a different image locations and resolutions.

The following parameters describe the scanning method and have to be decided as a trade-off between face detector resolution and algorithm speed:

- Block size: square or rectangular block that determinates the resolution of the face detector.
- Moving scan step: number of pixels that defines the window-sliding step to

get each next block to be analyzed.

- Down-sampling rate: down scale factor for the scaling technique to reach all locations and scales in an image.

Notice that when searching in high resolution images, using a small down-sampling rate and a small moving scan step can considerably slow down the face detection algorithm. The influence of these parameters are evaluated in more detail in *Appendix III.1.1 Face Detection Computational Cost Analysis*.

4.2.Feature Extraction for Two Stages Face Detection Scheme

Considering the work of *A.Hadid* [5][6] about face detection using LBP, a face detector based on 2 stages is implemented. The main reason to select such a scheme is to improve speed and efficiency detection. Only face candidates extracted from first coarse stage will go through second fine stage (see *Figure 6*).

As it will be explained later in more detail, instead of using $N \times N$ pixel values (image size) to describe an image, this new facial representation is proposed to

achieve:

1. Dimensionality reduction: only 203 values needed and it is more efficient for low- resolution.
2. Efficiency in the representation of the face towards different challenges like illumination or pose variations.

Image size	Number of pixels	Reduction using 203 values
16x16	256	20,70 %
18x21	378	46,29 %
19x19	361	43,76 %

Table 5. Face descriptor length reduction for the image sizes used in this research

4.2.1. Coarse Stage Features Extraction

The first stage verifies if the global image appearance can be a face candidate.

Therefore,

2 $LBP_{(8,1)}$ is extracted from the whole image obtaining a 59-bin labels histogram to have an

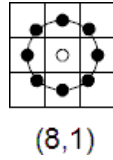


Figure 7. $LBP_{(8,1)}$

accurate description of the global image.

4.2.2. Fine Stage Features Extraction

Only positive results from previous step will be evaluated by means of a fine second stage that checks the spatial distribution of texture descriptors. In this case, $LBP_{(4,1)}$ operator is applied to the whole 16x16 pixel image and a 14x14 result image is obtained.

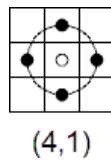


Figure 8. $LBP_{(4,1)}$

Then the resulting image is divided in 3x3 blocks of size 6x6 pixels with 2 pixels overlapping, as shown in Figure 9, where the grey block represents the first 6x6 pixel block and the green lines indicate the rest of overlapped blocks (2 pixels overlap).

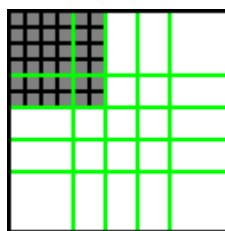


Figure 9. Face detection – 2nd fine stage – 3x3 blocks division with 2 pixels overlapping

Using 3x3 blocks division, each block gives a brief description of the local region by means of its 16-bin labels histogram. As a result, an amount of 144 (3x3 x 16-bin) features vector is obtained from this second fine stage. As it can be later seen, a different weight can be applied to each region in the classification

phase to emphasize most important face regions.

In training stage, these 2 resulting labels histograms are concatenated in an enhanced feature vector, resulting in a face representation histogram of:

$$59\text{-bin} + (3 \times 3 \text{ blocks} \times 16\text{-bin}) = 59\text{-bin} + 144\text{-bin} = 203\text{-bin}$$

Next *Figure 10* images show the result of applying local binary patterns in 1st coarse and 2nd fine stages. In result images (a) and (b), it can be clearly observed why *LBP*s are called a texture descriptor.



(a) Original image



(b) 1st Stg: LBP labels image



(c) 2nd Stg: LBP labels image rescaled to 0...255

(8,1)

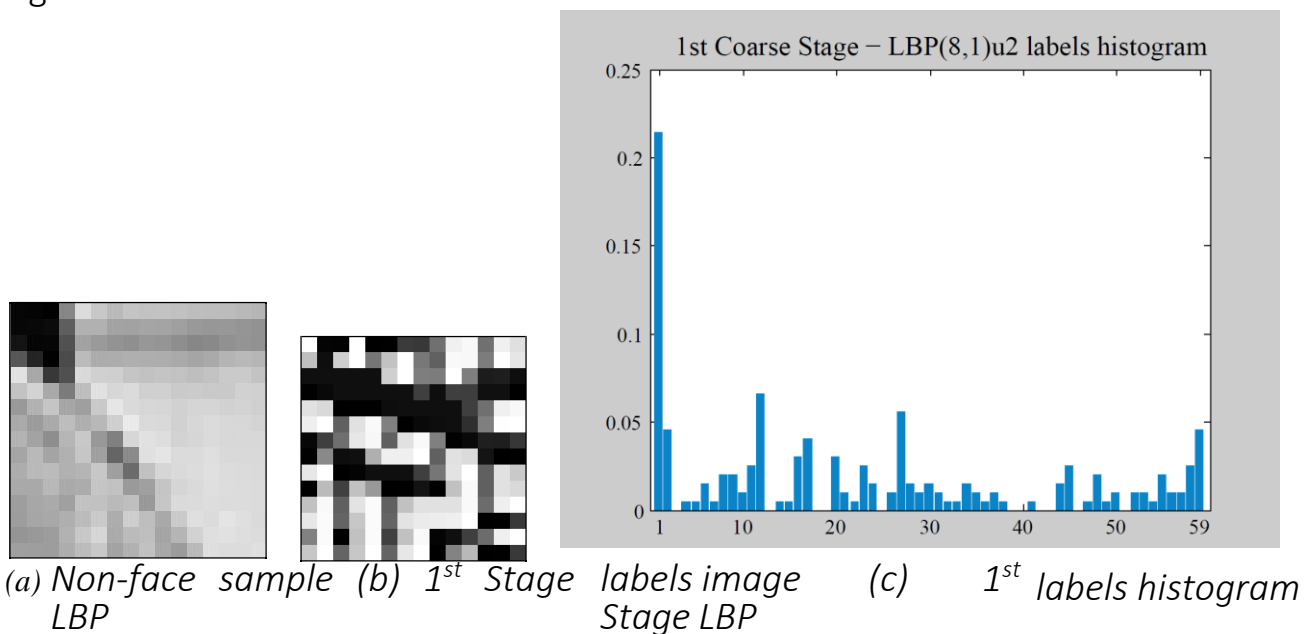
)

Figure 10. Image labeling example¹

As shown in *Figure 10*, the contours of the facial features (eyes, mouth, nose, kin, eyebrows...) are clearly remarked. In 1st stage labels image, contours are strongly highlighted giving a general overview of the image faceness being useful to discriminate non-face images in first fast stage. On the other hand, in

2nd stage labels image, local texture information is more detailed being useful for final stage decision where the image faceness is locally evaluated.

Following figures are intended to compare faces and non-faces labels histograms of both stages.



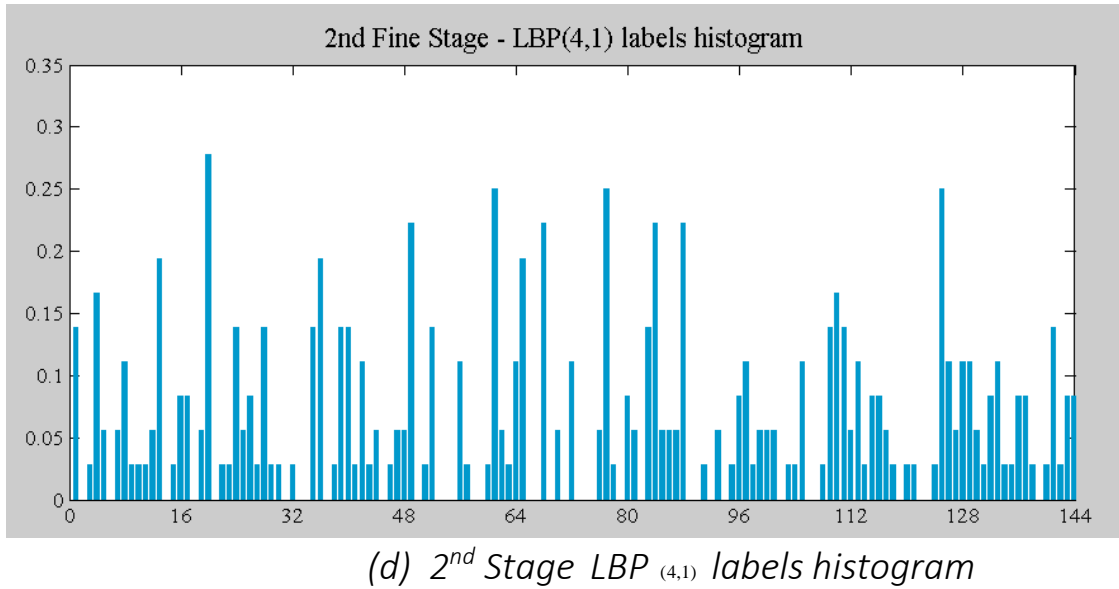


Figure 12. Non-face image - Labels histogram example

Comparing both examples, once again it can be confirmed that most important labels in LBP case corresponds to uniform patterns, as it can be observed in 1st stage LBP^{u2} labels(8,1) histograms where the first bin contains the percentage of non-uniform patterns. In Figure 11 only 18% of the pattern are non-uniform and in Figure 12 only the 22%.

At first glance, it is very difficult to highlight the main differences between both samples histograms that make possible that the resulting enhanced features vectors are useful in the task of faces and non-faces class discrimination. Therefore, somehow it is noticeable that some noise, i.e., information not useful to discriminate between both classes, is mixed with the useful information. Consequently, it can be assumed that a technique to extract the uncorrelated components can be helpful to remove irrelevant information and moreover to reduce features vector dimensionality. For that reason, *Principal Component Analysis* (PCA) will be introduced in next section as a method for this purpose.

4.3. Face Detection Classification

Following *Face Detection Scheme* (Figure 6), once the enhanced feature vectors are obtained for the training samples, some kind of faces and non-faces models are needed in order to complete training stage and to be used in classification stage.

The first one is *Chi Square* dissimilarly metric due to the fact that it is easy to implement and that it is a good technique for histogram comparison, as it will be explained in more detail in *Face Recognition* chapters.

The second one is the *Principal Component Analysis (PCA)* based on the idea of minimizing the features vector size.

4.4. Application to face detection

In this research, LBP enhanced feature vectors for 1st and 2nd stages have dimensions 59 and 144 respectively. Therefore, PCA is an interesting technique that can help to reduce data dimensionality by avoiding irrelevant information.

When comparing faces and non faces mean values, the difference between them is not obvious due to the high dimensionality of the data. Therefore, the goal by using this method is to extract the “*Principal Components*” so as to achieve a better comparison.

In the training stage, faces and non faces samples will generate one unique eigenvectors matrix that will represent the feature space (LBP space). Then for each class (faces and non- faces) one representative model is selected (e.g. the mean of all training samples of each class). These both models are projected to the LBP space.

In the test stage, given a new input test sample, its feature vector will be projected to the orthonormal basis. Then, comparing, this projected version of the input sample with the two projected models, faces and non-faces, the minimum distance will provide the faceness of the input vector. In experimental results *Chapter 7*, it will also be discussed the possibility of performing this comparison in the original space after back-projecting.

Following sections introduces how to use PCA as classifier using both face and non-face models or just by taking into account face class.



Figure 18. Face Detection Block Diagram - PCA Classifier

for more information about PCA algorithms, see *Appendix V. OpenCV Functions Reference*.

In section 7.3.2 in results chapter, the different parameters are adjusted: the number of eigenvectors to eliminate and, in the case of using only faces model, the K values to adjust the detection threshold. After evaluating detection mean ratios (about 85% true positive and 95% true negative), which are very similar to the obtained with *Chi Square*, it can be concluded that the fine-tuning of the parameters in order to maximize detection ratios is a difficult task and no remarkable benefits are achieved. Nevertheless, it is interesting to remark that only by using faces class better results are obtained, about a 84% true positive and 98 true negative.

4.5.SVM applied to face detection classification

Going back to the purpose of this work, let apply this theory to the face detection particular case. Based in *A.Hadid* experiments, [6], a second degree polynomial kernel function is selected as decision surface type. The LBP is computed at each iteration, LBP_x , from the subwindow and fed to the SVM classifier to determine whether it is a face or not.

The cost of constrain violation during the training process, C , has to be empirically calculated and controls the tradeoff between training errors and model complexity. If C is too large, then the system may overfit² and if it is too small, then it may underfit³ (see [41]).

In section 7.2.3 in results chapter, the SVM parameters will be adjusted for both 1st coarse and 2nd fine classification stages. Then, the face detection classification ratios will be presented showing best results compared to the previous classification proposals: *Chi Square* and *PCA*. In addition, an extra 3rd

classification stage will be proposed, as a contribution of this research project,

to improve final results.

For further information about SVM algorithms used, see *Appendix V. OpenCV Functions Reference*.

4.5.1.PCA before applying SVM

As seen in previous sections, *PCA* is an interesting technique that can help to reduce data dimensionality by avoiding irrelevant information. Therefore, this section proposes the use of *Principal Component Analysis* as post processing of the *Features Extraction* stage and previous to the *SVM* classifier, as it can be seen in *Figure 23*.



The methodology is the same as the already explained in *PCA* section 4.3.2. In the training stage, faces and non faces samples will generate one unique eigenvectors matrix that will represent *LBP* space. Then, each training sample will be projected to the orthonormal basis, then back-projected and directly used to train the *SVM* classifier.

In the test stage, the input vector under study will be projected and back-projected to the *PCA* matrix before entering the *SVM* classification stage.

In section 7.2.3.2 in results chapter, for simplicity, as it is very hard to fine-tuning all the parameters involved in the system, the same values calculated in *PCA* classification (7.2.2) and *SVM* classification (7.2.3) sections will be used. Only the 3rd threshold stage will be adapted to the new system. Final results will be presented showing also satisfactory face detection ratios, although maybe by finding again the optimal number of eigenvectors could improve the system.

Summarizing this section and taking into account experimental results *Chapter 7*, first classification method *Chi Square* serves to have a first perception of the *LBP* descriptor potential (about 85% true positive and 96% true negative). Then, *PCA* classification technique shows that redundant information reduction can

be performed with also similar results (about 85% true positive and 95% true negative). After that, *SVM* classifier together with a 3rd threshold stage contribution will provide the best results of the face detection section (about 95% true positive and 99% true negative). And the last classification proposal, the redundancy reduction using *PCA* before *SVM* classification presents also good results but very similar than using only *SVM*.

4.6. Face Detection System Improvements

Following section is intended to illustrate the improvements applied in the face detection scheme, starting with *Bootstrap Strategy* to reduce false face detection in training stage, and following with classification optimizations such as *Skin Segmentation* and *Overlapped Detections Removal*.

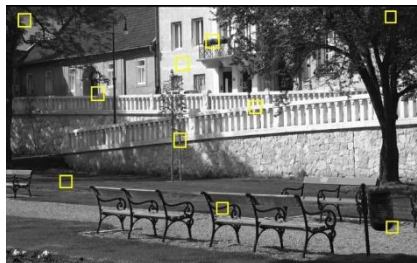
4.6.1. Training Stage

4.6.1.1. Bootstrap strategy to non-face pattern recollection

As proposed by *A.Hadid* in [6] and *K.-K. Sung* in [14], in face detection training stage, the negative examples are collected by means of a bootstrap strategy consisting of reentering false detections into the training set in a iterative way.

Applying such an approach, the difficult task of manually selecting representative non-face training samples is avoided just by weighting useful patterns from among redundant ones. At the end of each bootstrap iteration, the non-faces set is enlarged with new non-faces patterns that the current system has wrongly classified.

Such a strategy is applied in our experiments in order to collect the non-face



patterns. Therefore, firstly, between 6000-7000 non-face patterns are

Figure 24. Natural image from which negative training samples are extracted

randomly extracted from a set of natural images from Internet which do not contain faces.

Then the following scheme is iterated several times until the number of false alarms (non- faces classified as faces) is minimized:

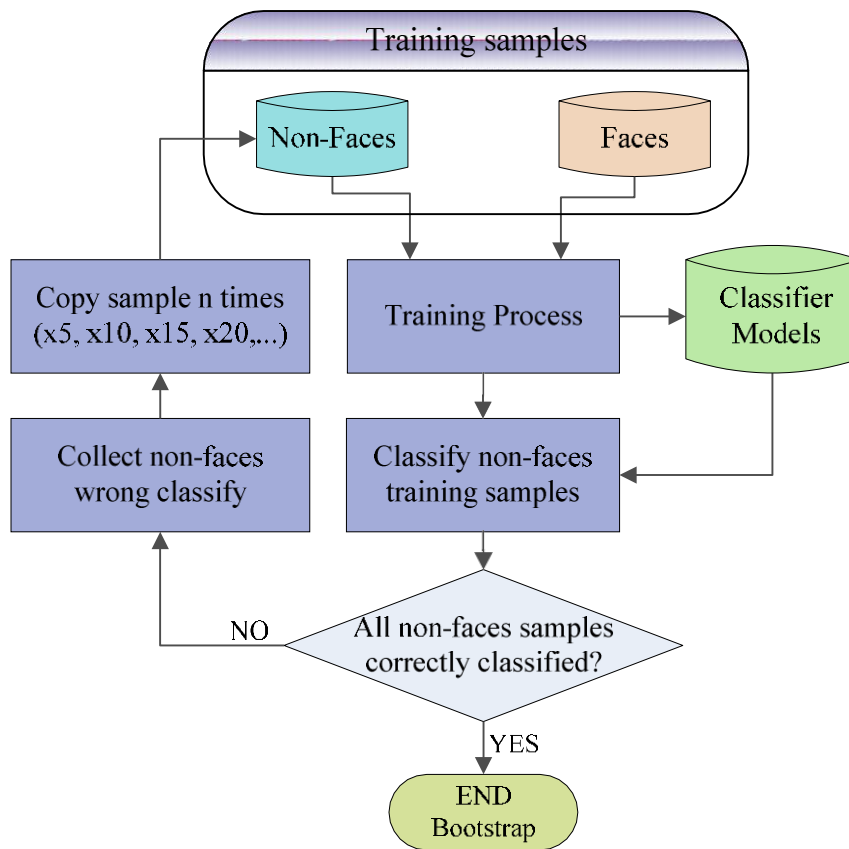


Figure 25. Bootstrap strategy flow chart

As it will be later verified in experimental results chapter, metimes it is not possible to converge to point of no false detections and a commitment between detection ratio drop and false alarm ratio improvement is required.

4.6.2. Image preprocessing (training / test)

Before calculating the LUV components, a preprocessing is applied to the original image to smooth the image and avoid compression noise (JPEG cases) and other artifacts. A 3x3 simple blur filter is used, consisting of the sum over a pixel 3x3 neighborhood with subsequent scaling by $1/(3 \times 3)$.

1/9	1/9	1/9
1/9		1/9
1/9	1/9	1/9

Given an image to be segmented, steps a) and b) are applied. Afterwards, exploring pixel

4.6.2.1. Skin segmentation (test)

l by pixel and taking into account a defined box size (37), the u and v components mean values are calculated as in training stage. This box size is selected in order to avoid image artifacts and to smooth possible region containing an eye or mouth that can create undesirable holes in the final skin mask. If the expected face size is $IMG_MIN_WIDTH_DET \times IMG_MIN_HEIGHT_DET$, then a mouth or an eye is supposed to be approximately a quarter of the face height and a quarter of the face width.

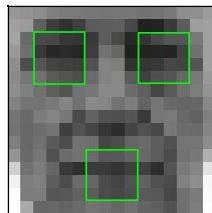


Figure 28. 16x16 image size.

Then, skin model thresholds are applied to determinate if the box region is skin or not.

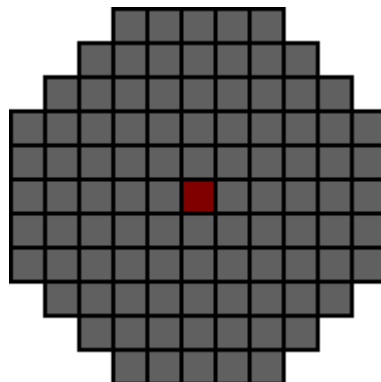


Figure 30. Opening structuring element: 3x5 pixels ellipse

Notice that the structuring element used in both morphological operation must be larger than the object to be removed.

Before applying face detection to each $IMG_MIN_WIDTH_DET \times IMG_MIN_HEIGHT_DET$ block, it is checked if 95% (*Face Skin Threshold*) of the

pixels in the block are classified as skin.

To summarize, next *Figure 31* gives an overview of the skin detection block diagram:



Figure 31. Skin detection block diagram

In the experimental results section 7.1, uv values for skin and non-skin training samples will be calculated in order to defined uv minimum and maximum thresholds. Then detection ratios will be presented ($\sim 98\%$) using a skin database for testing purposes (see section 6.1.3) and after that, a justification of the used morphological operations will be introduced showing some examples of how they are able to improve the efficiency of the proposed face detection scheme, e.g. eliminating skin candidates smaller than the minimum face resolution.

4.7. Overlapping Detections Removal

Following with classification stage improvements, this section exposes a post processing approach to eliminate overlapping detections.

In face detection procedure, the input image is explored (see section 4.1) by means of a scanning method defined by a square or rectangular box which determines the face detector resolution, a

window-sliding moving scan step to get each next block to be analyzed and a

down-sampling rate to allow image exploration in different scales.

As a result of this image scanning procedure, most faces are detected at multiple near positions and scales, leading into false detection that worsens face detector reliability.

In the literature, the most common strategies to resolve overlapping detections around a single face are the following ones:

- Overlapping Detection Removal: the block with higher classification score is considered the correct face detection and all other detection locations which overlap it are likely to be errors, and can therefore be eliminated.
- Overlapping Detection Merge:
 - The centroid of the near detections defines the location of the detection result.
 - Overlapping windows are approximated with one big box that contains all overlapped boxes.

Computationally speaking, the most adequate method is removal one, because

it saves up the centroid or container box calculation. Therefore, in this research removal algorithm was first implemented to evaluate its performance and as the results were very satisfactory, no extension to merge methods was required.

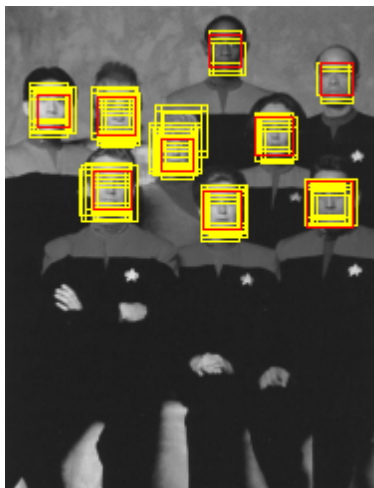
4.7.1. Overlapping Detection Removal Technique

The technique consists of sorting all detection boxes, including different scales, in decreasing order taking into account classification score. Then, the first box with highest faceness score is compared to the rest of the bounding boxes. Each box that overlaps it in more than a given percentage (threshold) is deleted from the boxes array.

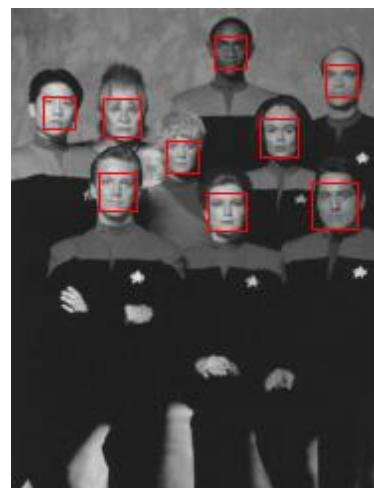
Empirically, 30% is decided as overlapping threshold achieving best results. Only

in case of real face overlapping, one face partially occluding another one, can lead to undesired results.

Next figure shows the result of applying overlapping detection removal algorithm. In yellow all face candidates and in red the result of applying overlapping discrimination:



(a) In yellow all face detection and in red result face with higher SVM faceness score.



(b) Result after overlapping boxes removal.

Figure 32. Overlapping detection removal example

CHAPTER - 5

LOCAL BINARY PATTERNS APPLIED TO FACE RECOGNITION

Local Binary Patterns descriptor is not only an efficient discriminative feature for face detection scenarios, but also for face recognition task.

This work pretends not only explore face detection approach but also face recognition method. Once *LBP* extraction is implemented, face recognition implementation based in the *University of Oulu* proposed methods (see [1][2][4]) is a straightforward task.

The *LBP* based face recognition approach consists of extracting the *LBP* features histograms from the whole face (59-bin histogram), divide the face image into 7x7 blocks and concatenate the blocks histograms into a unique vector (7x7 blocks x 59-bin/block = 2891 features). These features are calculated for each individual sample and a mean model is computed for each ID subset.

In face identification step, given a new input face image, the 2891 features vector is extracted and compared to all available models by means of a dissimilarity metric. The input face will be identified with the minimum ID model distance.

- Training stage (top box): input face samples are introduced in the pre-processing block together with an face-coordinates file and an ID label to crop faces from input image and resize them, if necessary.
- Test stage (bottom box): for each new test image, face detection block will output the cropped and resized face that will go through pre-processing stage for image quality improvement.

As, it will be seen in more detail, this chapter not only pretends to explain and test *University of Oulu* method, but also to improve it by means of using *Principal Component Analysis* to extract data redundancy simplifying features vector dimensionality.

5.1. Image preprocessing

Before the classification, input images should be adapted and enhanced to the face recognition system, either in training or test stage. The following steps describe image preprocessing stages:

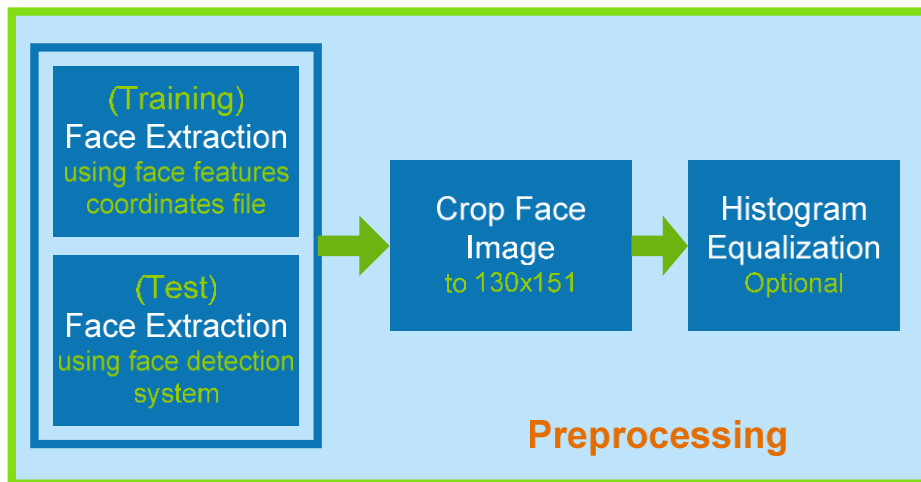


Figure 34. Face Recognition – Preprocessing Block Diagram

- Face Extraction

Faces are extracted from images by means of:

- Training case: a face features coordinates files provided for each image in the training dataset (see face recognition database information in section 6.3).
- Test case: a face detection pre-stage.

- Crop extracted faces images

In the literature [1], 7x7 blocks of size 18x21 pixels are recommended as a good trade-off between recognition performance and feature vector length. Therefore, and taking into account that 2 pixels margin is needed to apply need to be cropped to the following size:

$$\text{Width} = (7 \text{ blocks} \times 18 \text{ pixels}) + (2 \text{ margins} \times 2 \text{ pixels/margin}) = 130 \text{ pixels}$$

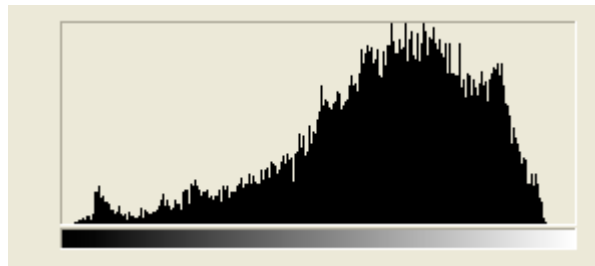
$$\text{Height} = (7 \text{ blocks} \times 21 \text{ pixels}) + (2 \text{ margins} \times 2 \text{ pixels/margin}) = 151 \text{ pixels}$$

- Histogram Equalization

Histogram equalization is a method of contrast adjustment using image's histogram.



(a) Original image



(b) Original image histogram



(c) Image after histogram equalization



(d) Equalized histogram

Figure 35. Histogram Equalization example

As it can be observed in (d), sometimes it is impossible to achieve a completely equalize histogram due to the fact that in this equalization process:

- A bin can be moved to a new grey level value bin.
- Bins with different values can result in a same grey value, then various bins can be accumulated into one.
- But a bin can never be divided in different grey level values.

Face images are very sensitive to illumination variations and small changes in light conditions can produce large changes in face appearance making face recognition task more difficult to handle. Therefore, histogram equalization is a useful preprocessing stage to face recognition task in order to not only enhance contrast but also edges and face details.

5.2.Face descriptor extraction

Just like in face detection, in face recognition a signature or a description of the face is required. This description will represent each individual in our database. A good face descriptor should retain all the specific features of each individual while compacting all this discriminant information in a few values or coefficients.

In this work, the approach of *A.Hadid* and *T.Ahonen* ([3][4]) has been selected. The authors proposed to use a LBP face descriptor. First, the original face descriptor will be explained. Second, the improvements proposed in this work will be presented in more detail.

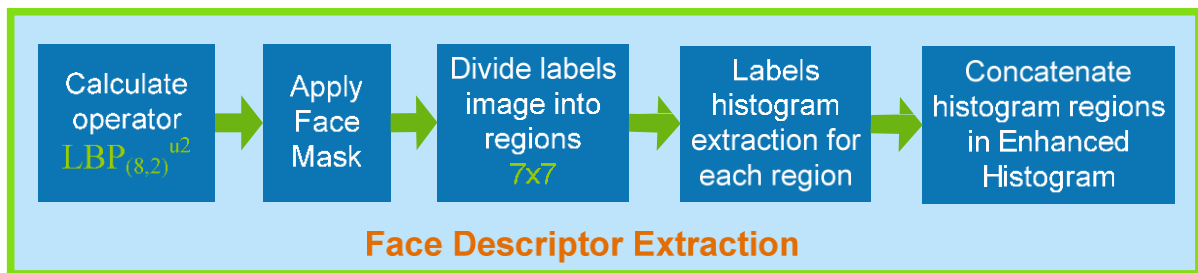


Figure 36. Face Recognition – Face Description Extraction Block Diagram

- Calculate operator from whole face image
- Apply face mask

The above mentioned researchers propose the application of a mask (see Figure 37) in order to avoid non face parts in the image. As it will be explained in later experimental results chapter 7.3, if the image is later divided in blocks and an appropriate weight value w_i is applied to each one, then the same effect can be achieved, at the same time that this extra step can be saved. Nevertheless, in face recognition experiments this mask will be used to verify the effect of using it.

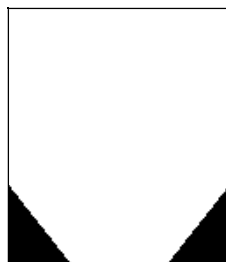


Figure 37. Image mask example

- Divide labels image into several regions (7x7 block of 18x21 pixels)

As a good compromise between face image description and resulting feature vector length (7x7 blocks x 59-bin/block = 2891), a 7x7 window division is proposed as a best choice for image recognition in *T.Ahonen* [1].



Figure 38. LBP image example of size 126x147: 7x7 blocks of 18x21 pixel

5.3.Face identification

Once the database is prepared and the spatially enhanced histograms are extracted from each identity face, *T.Ahonen* and *A.Hadid*, in [3] and [4], proposed *Chi Square* algorithm as a basic dissimilarity metric in order to compare each incoming face with the available database identities.

As it will be later exposed in experimental results chapter, this method provides satisfactory results, but as a contribution to this research work, it can be improved at the same time that face descriptor dimensionality is reduced by

eliminating redundant information using *Principal Component Analysis* approach.

Before exposing these dissimilarity metrics, next section gives an overview of face regions analysis to take advantage of human face recognition system to improve artificial recognition systems.

5.3.1.Face Regions Analysis

As already exposed in face detection chapter, section 4.3.1.2, dividing the face image in different regions can help to emphasize face features with essential information for face description and to remove irrelevant information by using an adequate weight value.

Obviously, nose, mouth and eyes regions are likely to provide more important information in face description task, than bottom left and right blocks where normally background areas, hair and even cloth can be present. Moreover, neurophysiologic studies based on face recognition human system evidence

that most important face features are eyes and eyebrows, followed by mouth and nose (see [28][29]).

Therefore, and taking into account human system knowledge, *Figure 39* shows a possible good choice for weighting face areas in face recognition procedure.

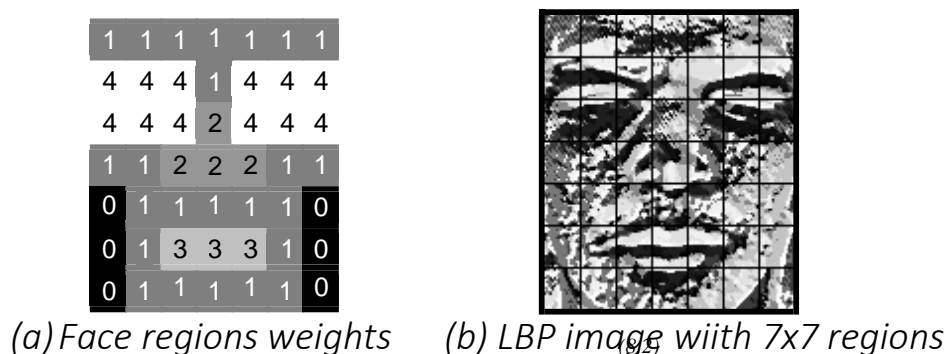


Figure 39. Face regions weights sample taking into account human recognition system

In experimental results chapter, different sets of block weights are defined taking into account above assumptions in order to later decide, in a empirically way, the combination to produce best classification results.

5.4.Face Recognition Classification

Human beings perform face recognition automatically every day and practically with no effort.

Although it sounds like a very simple task for us, it has proven to be a complex task for a computer, as it has many variables that can impair the accuracy of the methods, for example: illumination variation, low resolution, occlusion, amongst other In computer science, face recognition is basically the task of recognizing a person based on its facial image. It has become very popular in the last two decades, mainly because of the new methods developed and the high quality of the current videos/cameras.

Note that **face recognition** is different of **face detection**:

- **Face Detection:** it has the objective of finding the faces (location and size) in an image and probably extract them to be used by the face recognition algorithm.
- **Face Recognition:** with the facial images already extracted, cropped, resized and usually converted to grayscale, the face recognition algorithm is responsible for finding characteristics which best describe the image.

The face recognition systems can operate basically in two modes:

- **Verification or authentication of a facial image:** it basically compares the input facial image with the facial image related to the user which is requiring the authentication. It is basically a 1x1 comparison.
- **Identification or facial recognition:** it basically compares the input facial image with all facial images from a dataset with the aim to find the user that matches that face. It is basically a 1xN comparison.

There are different types of face recognition algorithms, for example:

- Eigenfaces (1991)
- Local Binary Patterns Histograms (LBPH) (1996)
- Fisherfaces (1997)
- Scale Invariant Feature Transform (SIFT) (1999)
- Speed Up Robust Features (SURF) (2006)

Each method has a different approach to extract the image information and perform the matching with the input image. However, the methods Eigenfaces and Fisherfaces have a similar approach as well as the SIFT and SURF methods.

Today we gonna talk about one of the oldest (not the oldest one) and more popular face recognition algorithms: **Local Binary Patterns Histograms (LBPH)**.

5.5. LBPH(Local Binary Patterns Histograms)

5.5.1. Objective

The objective of this post is to explain the **LBPH** as simple as possible, showing the method step-by-step.

As it is one of the easier face recognition algorithms I think everyone can understand it without major difficulties.

5.5.2. Introduction

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number.

It was first described in 1994 (LBP) and has since been found to be a powerful feature for texture classification. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.

Using the LBP combined with histograms we can represent the face images with a simple data vector.

As LBP is a visual descriptor it can also be used for face recognition tasks, as can be seen in the following step-by-step explanation.

Note: you can read more about the LBPH here: http://www.scholarpedia.org/article/Local_Binary_Patterns

5.5.3. Step-by-Step

Now that we know a little more about face recognition and the LBPH, let's go further and see the steps of the algorithm:

5.5.3.1. Parameters

the LBPH uses 4 parameters:

- **Radius:** the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1.
- **Neighbors:** the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.
- **Grid X:** the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
- **Grid Y:** the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

Don't worry about the parameters right now, you will understand them after reading the next steps.

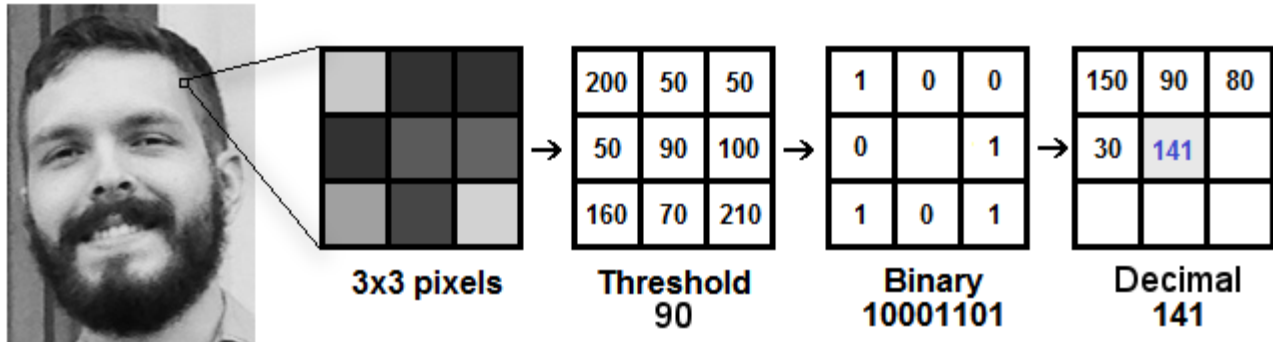
5.5.3.2. Training the Algorithm

First, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID. With the training set already constructed, let's see the LBPH computational steps.

5.5.3.3. Applying the LBP operation

The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters **radius** and **neighbors**.

The image below shows this procedure:

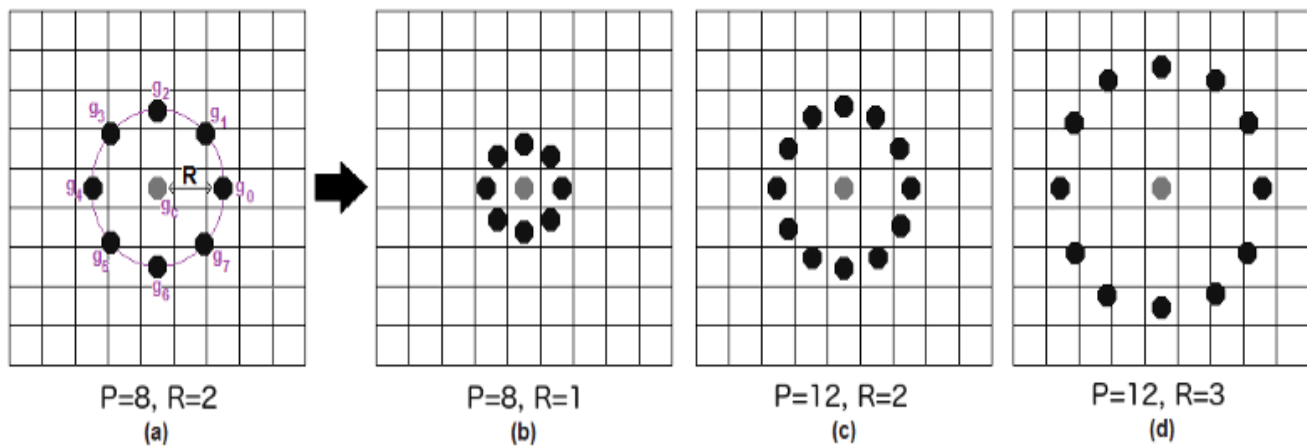


Based on the image above, let's break it into several small steps so we can understand it easily:

- Suppose we have a facial image in grayscale.
- We can get part of this image as a window of 3x3 pixels.
- It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).
- Then, we need to take the central value of the matrix to be used as the threshold.
- This value will be used to define the new values from the 8 neighbors.
- For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
- Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g. 10001101). Note: some authors use other

approaches to concatenate the binary values (e.g. clockwise direction), but the final result will be the same.

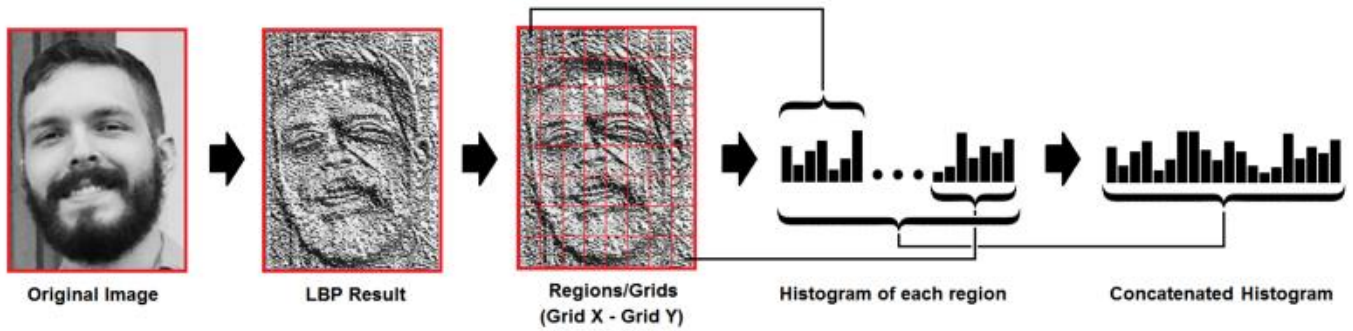
- Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is actually a pixel from the original image.
- At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.
- **Note:** The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.



It can be done by using **bilinear interpolation**. If some data point is between the pixels, it uses the values from the 4 nearest pixels (2x2) to estimate the value of the new data point.

5.5.3.4. Extracting the Histograms

Now, using the image generated in the last step, we can use the **Grid X** and **Grid Y** parameters to divide the image into multiple grids, as can be seen in the following image:



Based on the image above, we can extract the histogram of each region as follows:

- As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.
- Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have $8 \times 8 \times 256 = 16,384$ positions in the final histogram. The final histogram represents the characteristics of the image original image.

The LBPH algorithm is pretty much it.

5.5.3.5. Performing the face recognition

In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

- So to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.
- We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: **euclidean distance**, **chi-square**, **absolute value**, etc. In this example, we can use the Euclidean distance (which is quite known) based on the following formula:

$$D = \sqrt{\sum_{i=1}^n (hist1_i - hist2_i)^2}$$

- So the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a '**confidence**' measurement.

Note: don't be fooled about the 'confidence' name, as lower confidences are better because it means the distance between the two histograms is closer.

- We can then use a threshold and the 'confidence' to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.

6. CODE

```
from tkinter import *
from tkinter import ttk
import cv2
import os
import numpy as np
from PIL import Image
from threading import Thread
import mysql.connector
from mysql.connector import errorcode
from datetime import datetime
import time
from apscheduler.schedulers.background import BackgroundScheduler
from dateutil.parser import parse
tableoftab=[]
scheduler=BackgroundScheduler()
arra=[]

#create database
def create_database(cursor):
    try:
        cursor.execute(
            "CREATE DATABASE {} DEFAULT CHARACTER SET 'utf8'".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Failed creating database: {}".format(err))
        exit(1)

    try:
        cursor.execute("USE {}".format(DB_NAME))
    except mysql.connector.Error as err:
        print("Database {} does not exists.".format(DB_NAME))
        if err.errno == errorcode.ER_BAD_DB_ERROR:
            create_database(cursor)
            print("Database {} created successfully.".format(DB_NAME))
            cnx.database = DB_NAME
```

```
        else:
            print(err)
            exit(1)

#detect faculty in database
def startDetectorfaculty(strpid):
    if(strpid!=None):
        if(scheduler!=None):
            scheduler.remove_job(strpid)

    cnx =
mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'

    try:
        cnx.database = DB_NAME
    except mysql.connector.Error as err:
        if(err.errno == errorcode.ER_BAD_DB_ERROR):
            create_database(cursor)
            cnx.database = DB_NAME
        else:print(err);

    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read('trainer/trainer.yml')
    cascadePath = "Classifiers/face.xml"
    faceCascade = cv2.CascadeClassifier(cascadePath);
    path = 'dataSet'
    cam = cv2.VideoCapture(0)
    b=False
    nooframes=100
    #font = cv2.FONT_HERSHEY_SIMPLEX
    #cv2.putText(im,"detector",1, 1, 0, 1, 1)
    #font = cv2.InitFont(cv2.CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0, 1, 1)
#Creates a font
    while(True):
        ret, im =cam.read()
        gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
        faces=faceCascade.detectMultiScale(gray, scaleFactor=1.2,
```

```

minNeighbors=5, minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
    nooframes-=1
    name=""
    for (x,y,w,h) in faces:
        nbr_predicted, conf =
recognizer.predict(gray[y:y+h,x:x+w])
        cv2.rectangle(im,(x-50,y-
50),(x+w+20,y+h+20),(255,255,255),1)
        nbr_predicted=str(nbr_predicted)
        a=[]
        for i in range(0,len(nbr_predicted),2):
            a.append(int(nbr_predicted[i:i+2]))
        number=''.join(chr(i) for i in a)
        #print(number)
        cursor.execute("SELECT * FROM `map` WHERE
Short=\""+number+"\"")
        listofout=[]
        for (a,b) in cursor:
            listofout.append(a)
        try:
            name=str(listofout.pop())
        except:
            name=""
        cv2.putText(im, "Faculty : "+ name, (x-50,y+h+50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2,cv2.LINE_8,False)
        cv2.namedWindow("im", cv2.WND_PROP_FULLSCREEN)

        cv2.setWindowProperty("im",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)

        cv2.imshow('im',im)
        if(cv2.waitKey(1)==ord('y')):
            fnG.set(name)
            print(name)
            cv2.destroyAllWindows()
            cam.release()
            startDetectors()
            b=True
            break
        if(cv2.waitKey(1)==ord('q')):
            cv2.destroyAllWindows()

```

```
        b=True
        break
    if(nooframes==0):
        cv2.destroyAllWindows()
        cam.release()
        cursor.close()
        cnx.commit()
        cnx.close()
        b=True
        #if(name!=""):
        fnG.set(name)
        print(name)
        startDetectors()
    if(b==True):
        break

def destartDe(strpid):
    if(scheduler!=None):
        scheduler.remove_job(strpid)
    stoperofdet.set("stop")
    print(stoperofdet)

#Unique ID for Each
def printd(event=None):
    global tableoftab
    global arra
    print("nhvn")
    for i in range(1,4):
        arra.append([str(i*10+1),str((i+1)*10+i+1)])

    try:
        hourint,minint=map(int,tableoftab[i][1].get().split(":"))
    except:
        break
    date = parse(time.ctime())
    print(date)
    diff=(minint*60+hourint*3600)-
```

```

((date.minute*60+date.hour*3600+date.second))
        scheduler.add_job(startDetectorfaculty,                                'interval',
seconds=diff,id=arra[i-1][0],args=(arra[i-1][0],))
        hourout,minout=map(int,tableoftab[i][2].get().split(":"))
        diff=(minout*60+hourout*3600)-
((date.minute*60+date.hour*3600+date.second))
        scheduler.add_job(destartDe,                                        'interval',
seconds=diff,id=arra[i-1][1],args=(arra[i-1][1],))
        scheduler.start()
def Shortner(FullName):
    cnx                                                                    =
mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'
    try:
        cnx.database = DB_NAME
    except mysql.connector.Error as err:
        if(err.errno == errorcode.ER_BAD_DB_ERROR):
            create_database(cursor)
            cnx.database = DB_NAME
        else:print(err);
    select_q="SELECT * FROM `map`"
    cursor.execute(select_q)
    number=0
    for (a,b) in cursor:
        number=b
    number=str(hex(int(number,16)+1))[2:]
    number='0'*(4-len(number))+number
    add_stuednt = "INSERT INTO map (Name, Short) VALUES (%s, %s)"
    data_student = (FullName,number)
    cursor.execute(add_stuednt, data_student)
    cnx.commit()
    cursor.close()
    cnx.close()
    return number
def startGen(event=None):
    cnx                                                                    =
mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'

```



```

try:
    cnx.database = DB_NAME
except mysql.connector.Error as err:
    if(err.errno == errorcode.ER_BAD_DB_ERROR):
        create_database(cursor)
        cnx.database = DB_NAME
    else:print(err);
cam = cv2.VideoCapture(0)
detector=cv2.CascadeClassifier('Classifiers/face.xml')
i=0
offset=50
name=nameVar.get()
sId=studentoId.get()
listofclass=classNames.get().split(",")
print(name,classNames)
for k1 in listofclass:
    insert_q="INSERT INTO facclass VALUES (%s, %s)"
    cursor.execute(insert_q,(name,k1))
    create="CREATE TABLE `attend`.`"+name+"_"+k1+"` ( `ID`
VARCHAR(20) NOT NULL ) ENGINE = InnoDB;"
    cursor.execute(create)
cnx.commit()
cursor.close()
cnx.close()
while(True):
    ret,im =cam.read()
    if(not ret):
        continue
    gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    faces=detector.detectMultiScale(gray, scaleFactor=1.2,
minNeighbors=5, minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
    for(x,y,w,h) in faces:
        i=i+1
        cv2.imwrite("dataSet/face-"+str(name) + "#" + str(sId)
+'.'+ str(i) + ".jpg", gray[y-offset:y+h+offset,x-offset:x+w+offset])
        cv2.rectangle(im,(x-50,y-50),(x+w+50,y+h+50),(225,0,0),2)
        cv2.imshow('im',im[y-offset:y+h+offset,x-
offset:x+w+offset])
        cv2.waitKey(100)
    if(i>33):

```

```

        cam.release()
        cv2.destroyAllWindows()
        break
def startGens(event=None):
    cam = cv2.VideoCapture(0)
    detector=cv2.CascadeClassifier('Classifiers/face.xml')
    i=0
    offset=50
    name=nameVar.get()
    clasa=classNames.get()
    sId=studentoId.get()
    print(sId)
    cnx
mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'
    try:
        cnx.database = DB_NAME
    except mysql.connector.Error as err:
        if(err.errno == errorcode.ER_BAD_DB_ERROR):
            create_database(cursor)
            cnx.database = DB_NAME
        else:print(err);
    cursor.execute("INSERT INTO stuclass (Name,Class) VALUES
('"+name+"','"+clasa+"')")
    cnx.commit()
    cursor.close()
    cnx.close()
    #print(name)
    while(True):
        ret, im =cam.read()
        gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
        faces=detector.detectMultiScale(gray, scaleFactor=1.2,
minNeighbors=5, minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
        for(x,y,w,h) in faces:
            i=i+1
            xyzim=gray[y-offset:y+h+offset,x-offset:x+w+offset]
            cv2.equalizeHist(xyzim)
            cv2.imwrite("dataSetS/face-"+str(name) + "#" + str(sId)
+'.'+ str(i) + ".jpg", xyzim)

```

```

        cv2.rectangle(im,(x-50,y-50),(x+w+50,y+h+50),(225,0,0),2)
        cv2.imshow('im',im[y-offset:y+h+offset,x-
offset:x+w+offset])
        cv2.waitKey(100)
    if(i>50):
        cam.release()
        cv2.destroyAllWindows()
        break

```

#Takes image from ggenerated dataset

```
def get_images_and_labels(path):
```

```
    cnx
```

```
mysql.connector.connect(user='root',password="",host="localhost")
```

```
    cursor = cnx.cursor()
```

```
    DB_NAME = 'attend'
```

```
    try:
```

```
        cnx.database = DB_NAME
```

```
    except mysql.connector.Error as err:
```

```
        if(err.errno == errorcode.ER_BAD_DB_ERROR):
```

```
            create_database(cursor)
```

```
            cnx.database = DB_NAME
```

```
        else:print(err);
```

```
    image_paths = [os.path.join(path, f) for f in os.listdir(path)]
```

```
    # images will contains face images
```

```
    images = []
```

```
    # labels will contains the label that is assigned to the image
```

```
    labels = []
```

```
    getN=None
```

```
    firstTime=True
```

```
    inoofima=0
```

```
    for image_path in image_paths:
```

```
        # Read the image and convert to grayscale
```

```
        image_pil = Image.open(image_path).convert('L')
```

```
        # Convert the image format into numpy array
```

```
        image = np.array(image_pil, 'uint8')
```

```
        # Get the label of the image
```

```
        FullName
```

```
,
```

```
getN
```

```
os.path.split(image_path)[1].split(".")[0].replace("face-",
"" ).upper().split("#")
```

```

    # Detect the face in the image
    if(firstTime):
        firstTime=False
        cursor.execute("INSERT INTO `map` VALUES
('"+FullName+"', '"+getN+"'")
        cnx.commit()
        cursor.close()
        cnx.close()
    print("GetN",getN)
    if(inoofima==51):
        inoofima=0
    number=int(''.join(str(ord(c)) for c in getN.upper()),10)
    cascadePath = "Classifiers/face.xml"
    faceCascade = cv2.CascadeClassifier(cascadePath);
    faces = faceCascade.detectMultiScale(image)
    inoofima+=1
    # If face is detected, append the face to images and the label
    to labels
    for (x, y, w, h) in faces:
        images.append(image[y: y + h, x: x + w])
        labels.append(number)
        #cv2.imshow("Adding faces to traning set...", image[y: y
+ h, x: x + w])
        cv2.waitKey(10)
    # return the images list and labels list
    return images, labels

```

```

def startTrain(event=None):
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    path = 'dataSet'
    images, labels = get_images_and_labels(path)
    cv2.imshow('test',images[0])
    cv2.waitKey(1)
    recognizer.train(images, np.array(labels))
    recognizer.write('trainer/trainer.yml')
    cv2.destroyAllWindows()

```

```

def startTrains(event=None):

```

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
path = 'dataSets'
images, labels = get_images_and_labels(path)
cv2.imshow('test',images[0])
cv2.waitKey(1)
recognizer.train(images, np.array(labels))
recognizer.write('trainer/trainers.yml')
cv2.destroyAllWindows()
def startDetector(event=None):
    cnx
mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'

    try:
        cnx.database = DB_NAME
    except mysql.connector.Error as err:
        if(err.errno == errorcode.ER_BAD_DB_ERROR):
            create_database(cursor)
            cnx.database = DB_NAME
        else:print(err);

    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read('trainer/trainer.yml')
    cascadePath = "Classifiers/face.xml"
    faceCascade = cv2.CascadeClassifier(cascadePath);
    path = 'dataSet'
    cam = cv2.VideoCapture(0)
    b=False
    nooframes=100
    #font = cv2.FONT_HERSHEY_SIMPLEX
    #cv2.putText(im,"detector",1, 1, 0, 1, 1)
    #font = cv2.InitFont(cv2.CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0, 1, 1)
#Creates a font
    while(True):
        ret, im =cam.read()
        gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
        faces=faceCascade.detectMultiScale(gray, scaleFactor=1.2,
```

```

minNeighbors=5, minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
    nooframes-=1
    print(nooframes)
    name=""
    for (x,y,w,h) in faces:
        nbr_predicted, conf =
recognizer.predict(gray[y:y+h,x:x+w])
        cv2.rectangle(im,(x-50,y-
50),(x+w+20,y+h+20),(255,255,255),1)
        nbr_predicted=str(nbr_predicted)
        a=[]
        for i in range(0,len(nbr_predicted),2):
            a.append(int(nbr_predicted[i:i+2]))
        number=''.join(chr(i) for i in a)
        #print(number)
        cursor.execute("SELECT * FROM `map` WHERE
Short=\""+number+"\"")
        listofout=[]
        for (a,b) in cursor:
            listofout.append(a)
        try:
            name=str(listofout.pop())
        except:
            name=""
        cv2.putText(im, "Faculty : "+ name, (x-50,y+h+50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2,cv2.LINE_8,False)
        cv2.namedWindow("im", cv2.WND_PROP_FULLSCREEN)

        cv2.setWindowProperty("im",cv2.WND_PROP_FULLSCREEN,cv2.WINDOW_FULLSCREEN)

        cv2.imshow('im',im)
        if(cv2.waitKey(1)==ord('y')):
            fnG.set(name)
            cv2.destroyAllWindows()
            cam.release()
            startDetectors()
            b=True
            break
        if(cv2.waitKey(1)==ord('q')):
            cv2.destroyAllWindows()

```

```

        b=True
        break
    if(nooframes==0):
        cv2.destroyAllWindows()
        cam.release()
        cursor.close()
        cnx.commit()
        cnx.close()
        b=True
        if(name!=""):
            fnG.set(name)
            startDetectors()

    if(b==True):
        break

```

#generate Excel sheet of attendance

def CSV(event=None):

```

    cnx
    mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'

```

try:

```

    cnx.database = DB_NAME
except mysql.connector.Error as err:
    if(err.errno == errorcode.ER_BAD_DB_ERROR):
        create_database(cursor)
        cnx.database = DB_NAME
    else:print(err);

```

```

tabname=nameVar.get()+"_"+classNames.get()
selectcol="show columns from attend."+tabname
cursor.execute(selectcol)
soham=""
for x in cursor:
    soham+=x[0]+", "
soham+='\n'
select_csv="SELECT * FROM `"+tabname+"`"

```

```

    cursor.execute(select_csv)
    for x in cursor:
        for okokok in x:
            if(okokok!=None):soham+=okokok+', '
            else:soham+=", "
        soham+='\n'
    cursor.close()
    cnx.commit()
    cnx.close()
    fileName=nameVar.get()+' '+datetime.now().ctime()+'.csv'
    fileName=fileName.replace(":", "")
    fileName=fileName.replace(" ", "")
    #print(fileName)
    myFile = open(""+fileName+"", 'w')
    myFile.write(soham)
    myFile.close()
    os.system('start '+''+fileName+'')
def startDetectors(event=None):
    cnx
mysql.connector.connect(user='root',password="",host="localhost")
    cursor = cnx.cursor()
    DB_NAME = 'attend'
    date = parse(time.ctime())
    try:
        cnx.database = DB_NAME
    except mysql.connector.Error as err:
        if(err.errno == errorcode.ER_BAD_DB_ERROR):
            create_database(cursor)
            cnx.database = DB_NAME
        else:print(err);
    cursor.execute("SELECT * FROM `facclass` WHERE
Name=\""+fnG.get()+"\"")
    clasthin=''
    for (ab,ck) in cursor:
        clasthin=ck
    thistime=time.ctime()
    cursor.execute("ALTER TABLE `"+fnG.get()+"_"+clasthin+"` ADD
`"+thistime+"` VARCHAR(20) ")
    recognizer = cv2.face.LBPHFaceRecognizer_create()

```



```

recognizer.read('trainer/trainers.yml')
cascadePath = "Classifiers/face.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
path = 'dataSets'
cam = cv2.VideoCapture(0)
fn='attendance'
b=False
arrayofar={}
#font = cv2.FONT_HERSHEY_SIMPLEX
#cv2.putText(im,"detector",1, 1, 0, 1, 1)
#font = cv2.InitFont(cv2.CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0, 1, 1)
#Creates a font
while(True):
    ret, im =cam.read()
    if(not ret):
        #print("c")
        continue
    gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    faces=faceCascade.detectMultiScale(gray,          scaleFactor=1.2,
minNeighbors=5, minSize=(100, 100), flags=cv2.CASCADE_SCALE_IMAGE)
    for (x,y,w,h) in faces:
        nbr_predicted, conf = recognizer.predict(gray[y:y+h,x:x+w])
        if conf > 50 :
            name=""
            cv2.rectangle(im,(x-50,y-
50),(x+w+20,y+h+20),(255,255,255),1)
            nbr_predicted=str(nbr_predicted)
            a=[]
            for i in range(0,len(nbr_predicted),2):
                a.append(int(nbr_predicted[i:i+2]))
            number=''.join(chr(i) for i in a)
            #print(number)
            cursor.execute("SELECT * FROM `map` WHERE
Short=\""+number+"\"")
            listofout=[]
            for (a,b) in cursor:
                listofout.append(a)
            try:
                name=str(listofout.pop())

```

```

except:
    name=""
    select_q="SELECT * FROM `stuclass` WHERE
Name=\""+name+"\"
    cursor.execute(select_q)
    if( name not in arrayofar.keys()):
        arrayofar.update({name:0})
    classno=None
    for (a,b) in cursor:
        classno=b
    if(classno!=None):
        try:
            update_q="UPDATE
"+fnG.get()+"_"+str(classno)+"` SET `"+thistime+"`='P' WHERE
ID=\""+number+"\"
            select_q="SELECT * FROM
"+fnG.get()+"_"+str(classno)+"` WHERE ID=\""+number+"\"
            cursor.execute(select_q)
            cfsel=0
            for spofkopsd in cursor:
                cfsel+=1
            if(cfsel==0):
                print(fnG.get()+"_"+str(classno))
                add_stuednt = "INSERT INTO
"+fnG.get()+"_"+str(classno)+" (ID, `"+thistime+"`) VALUES (%s, %s)"
                data_student = (number, 'P')
                cursor.execute(add_stuednt,
data_student)

                print("excuted")
            else:
                cursor.execute(update_q)
                arrayofar.update({name:1})
        except mysql.connector.errors.ProgrammingError:
            arrayofar.update({name:0})

    else:
        cv2.rectangle(im,(x-50,y-
50),(x+w+20,y+h+20),(255,255,255),1)
        name="unknown"
        #Draw the text
        cv2.putText(im, "Student : "+name, (x-50,y+h+50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2,cv2.LINE_8,False)

```

```
        cv2.imshow('im',im)
        print(name)
        if(cv2.waitKey(1)==ord('q')):
            cv2.destroyAllWindows()
            cam.release()
            cursor.close()
            cnx.commit()
            cnx.close()
            print("cnx closed")
            b=True
            break
    if(stoperofdet.get()=="stop"):
        cv2.destroyAllWindows()
        cam.release()
        cursor.close()
        cnx.commit()
        cnx.close()
        print("cnx closed")
        stoperofdet.set("")
        b=True
        break
    if(b==True):
        break

noone=0
nozero=0
print(arrayofar)
for b in arrayofar.values():
    if(b==0):
        nozero+=1
    else:
        noone+=1

filename="log "+datetime.now().strftime("%d %b %Y")+".txt"
filename=filename.replace(":","")
filename=filename.replace(" ","")

logfile=open(filename,"w")
logfile.write(str(noone)+"                               Accpted\n"+str(nozero)+"
Rejected\n"+"total students detected "+str(noone+nozero))
logfile.close()
```

```

    print(filename)
    os.system('start '+''+filename+'')
def getData1(event=None):
    print("getData1")
def getData2(event=None):
    print("getData2")
    global nameVar1,nameVar2,optFrame,root1,root

    if(nameVar1.get()=="jay" and nameVar2.get()=="jay123"):
        print("x")
        root1.destroy()
        root=Tk()

        btn = StringVar()
        globalId=StringVar()

    def full(event=None):
        state = True
        root.attributes("-fullscreen", state)

    optFrame=LabelFrame(root,text="Options",font=("times new
roman",23),fg="black",padx=40,pady=20)
    def close_window(event=None):
        global root

        root.destroy()
    optFrame.pack(side=LEFT,fill=BOTH, expand=YES)

#-----making optFrame-----

    #changes
    info=LabelFrame(optFrame,
details",fg="black",font=("times new roman", 18))
    info.grid(row=1, column=0, sticky=W)
    def getData(event=None):
        text="Personal

```

```

name=nameVa.get()
clas=className.get()
id=studentId.get()
global nameVar,classNames,studentoId
nameVar.set(name)
classNames.set(clas)
studentoId.set(id)

```

```

Label(info,text="Name:",font=("times new roman",
17),fg="black").grid(row=0,column=0,padx=0)
nameVa=StringVar()
className=StringVar()
studentId=StringVar()
nameEntry = Entry(info, textvariable=nameVa,font=("times new
roman",17))
nameEntry.grid(row=0,column=1,padx=20,ipady=2)
getDataButton = Button(info,
text="Submit",bg="grey",font=("times new roman", 17),fg="white")
getDataButton.grid(row=1,padx=20,pady=10,column=1)
getDataButton.bind("<Button-1>", getData)
Label(info,text="Class Names:",font=("times new roman",
17),fg="black").grid(row=0,column=2,padx=0)
classEntry = Entry(info, textvariable=className,font=("times
new roman",17))
classEntry.grid(row=0,column=3,padx=10,ipady=2)
Label(info,text="Id:",font=("times new roman",
17),fg="black").grid(row=0,column=4,padx=0)
idEntry = Entry(info, textvariable=studentId,font=("times new
roman",17))
idEntry.grid(row=0,column=5,padx=10,ipady=2)
#changelnd
#One Button for all things

recg=LabelFrame(optFrame,
text="Recognize",fg="black",padx=245,pady=5,font=("times new roman",
18))
recg.grid(row=3,column=0,sticky=W)

```

```

Label(recg,text="Registration:",font=("times new roman",
17),fg="black").grid(row=1,column=0,sticky=W)
#Label(recg,text="Registration :",padx=5,bg="grey",font=("times new
roman", 17),fg="white").grid(row=1,column=0,sticky=W)
genButtonf = Button(recg,
text="Faculty",bg="grey",font=("times new roman", 17),fg="white")
genButtonf.grid(row=1,column=1,sticky=W, padx=20)
genButtonf.bind("<Button-1>", startGen)
genButtons = Button(recg,
text="Student",bg="grey",font=("times new roman", 17),fg="white")
genButtons.grid(row=1,column=2,sticky=W)
genButtons.bind("<Button-1>", startGens)

Label(recg,text="Training:",font=("times new roman",
17),fg="black").grid(row=2,column=0,sticky=W)
trainButtonf = Button(recg, text="Faculty
Trainer",bg="grey",font=("times new roman", 17),fg="white")
trainButtonf.grid(row=2,column=1,sticky=W,padx=20)
trainButtonf.bind("<Button-1>", startTrain)
trainButtons = Button(recg, text="Student
Trainer",bg="grey",font=("times new roman", 17),fg="white")
trainButtons.grid(row=2,column=2,sticky=W,pady=10)
trainButtons.bind("<Button-1>", startTrains)

Label(recg,text="Recognizer:",font=("times new roman",
17),fg="black").grid(row=3,column=0,sticky=W)
recgButton = Button(recg, text="Start",bg="grey",font=("times
new roman", 17),fg="white")
recgButton.grid(row=3,column=1,sticky=W,padx=20)
recgButton.bind("<Button-1>", lambda e :
startDetectorfaculty(None))
def close_window(event=None):
    global root
    root.destroy()

##To Get Out Of The Program
Kill=LabelFrame(optFrame, text="Close and Excel file
Generation",fg="black",padx=365, pady=10,font=("times new roman", 18))
Kill.grid(row=4, column=0, sticky=W)
button = Button(Kill, text=" EXIT ",bg="grey",font=("times
new roman", 17),fg="white")
button.grid(row=4,column=1,sticky=W)
button.bind("<Button-1>", close_window)

```

```

        button = Button(Kill, text="    CSV    ",bg="grey",font=("times
new roman", 17),fg="white")
        button.grid(row=4,column=2,sticky=W,padx=20)
        button.bind("<Button-1>", CSV)

root1=Tk()
root1.title("ADMIN PANEL")
def close_windowdown(event=None):
    global root1

    root1.destroy()
fnG=StringVar()
classNames=StringVar()
nameVar = StringVar()
studentoId = StringVar()
stoperofdet=StringVar()
stoperofdet.set("")
adminFrame=LabelFrame(root1,text="Admin    Login",font=("times    new
roman",23),fg="black")
adminFrame.pack(side=BOTTOM,fill=BOTH,expand=YES)
Label(adminFrame,text="USERNAME:",font=("times    new    roman",
17),fg="black").grid(row=0,column=0,sticky=W,padx=15)
nameVar1 = StringVar()
nameEntry1 = Entry(adminFrame, textvariable=nameVar1,font=("times new
roman",17))
nameEntry1.grid(row=0,column=1,padx=20,ipady=2)
#getDataButton = Button(info, text="Submit",bg="grey",font=("times new
roman", 17),fg="grey")
#getDataButton.grid(row=1,padx=20,pady=20,column=1,sticky=W)
#getDataButton.bind("<Button-1>", getData1)
Label(adminFrame,text="PASSWORD:",font=("times    new    roman",
17),fg="black").grid(row=3,column=0,sticky=W,padx=15,pady=10)
nameVar2 = StringVar()
nameEntry2 = Entry(adminFrame, textvariable=nameVar2,font=("times new
roman",17))
nameEntry2.grid(row=3,column=1,padx=35,ipady=2,pady=10)
getDataButton1
                =                Button(adminFrame,
text="Submit",bg="grey",font=("times new roman", 17),fg="white")
getDataButton1.grid(row=4,padx=20,pady=20,column=1,sticky=W)
getDataButton1.bind("<Button-1>", getData2)
Label(adminFrame,text="Recognizer:",font=("times    new    roman",

```

```
17),fg="black").grid(row=0,column=2,sticky=W)
recgButton = Button(adminFrame, text="Start",bg="grey",font=("times new
roman", 17),fg="white")
recgButton.grid(row=0,column=3,sticky=E,padx=10,pady=9)
recgButton.bind("<Button-1>", lambda e : startDetectorfaculty(None))
button2 = Button(adminFrame, text="    EXIT    ",bg="grey",font=("times
new roman", 17),fg="white")
button2.grid(row=4,column=2,sticky=W)
button2.bind("<Button-1>", close_windowdown)

mainloop()
```

How to run Code

ALL COMMENTS IN CODE

1)Create (Here already given) 3 folders that are:

- dataSet(Faculty dataset images)
- dataSets(Student dataset images)
- trainer

2)install libraries like opencv-contrib-python, numpy,Pillow, mysql-connector-python, python-dateutil, apscheduler, datetime.

3)install Xampp for backend database of attendance.

4)install or import sql database(attend and attendance_management files) of data folder in phpmyadmin in local host.

5) Run Apache and mysql server on xampp.

6)run final.py

7)It shows admin panel entered below password.

admin username : jay

admin password : jay123

8)Then enter name class and unique Id and submit.

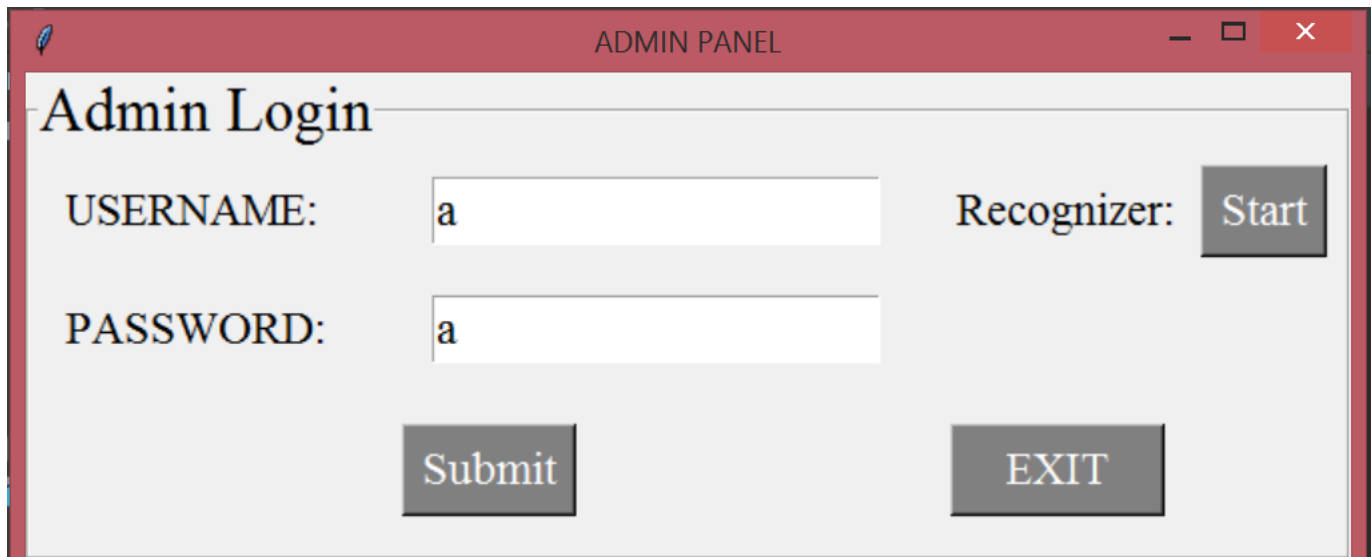
9) Then register it as either as student or faculty.

10)Then train model.

11) Then start recognizer (First window recognizes faculty and class and then student).

12) Then, It will create log file that contains the information about number of students and faculties.

7. EXPERIMENTAL RESULTS



The screenshot shows a window titled "ADMIN PANEL" with a red header bar. Inside, the "Admin Login" section has two input fields: "USERNAME:" with the value "a" and "PASSWORD:" with the value "a". To the right of the password field is a "Recognizer:" label and a "Start" button. At the bottom, there are two buttons: "Submit" and "EXIT".

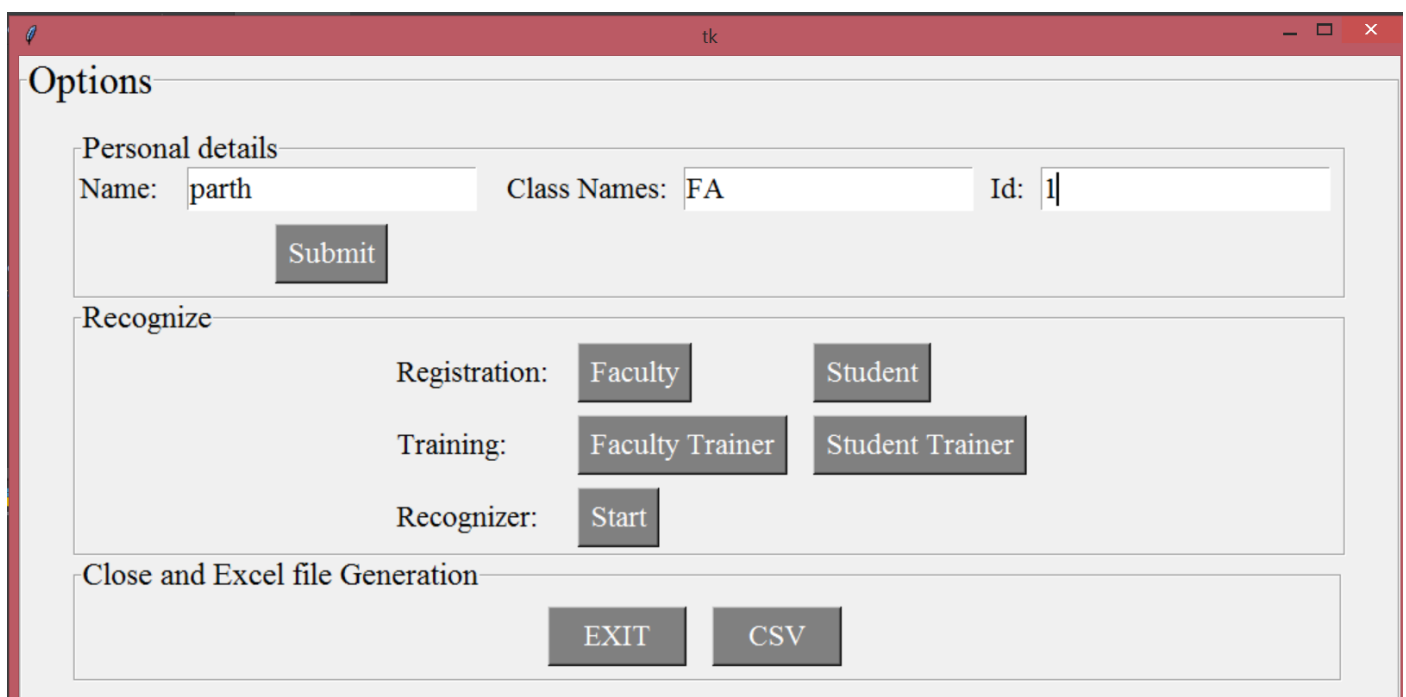
ADMIN PANEL

Admin Login

USERNAME: a Recognizer: Start

PASSWORD: a

Submit EXIT



The screenshot shows a window titled "Options" with a red header bar. It contains three sections: "Personal details" with fields for "Name:" (value "parth"), "Class Names:" (value "FA"), and "Id:" (value "1"), followed by a "Submit" button; "Recognize" with buttons for "Registration:" (Faculty, Student), "Training:" (Faculty Trainer, Student Trainer), and "Recognizer:" (Start); and "Close and Excel file Generation" with buttons for "EXIT" and "CSV".

Options

Personal details

Name: parth Class Names: FA Id: 1

Submit

Recognize

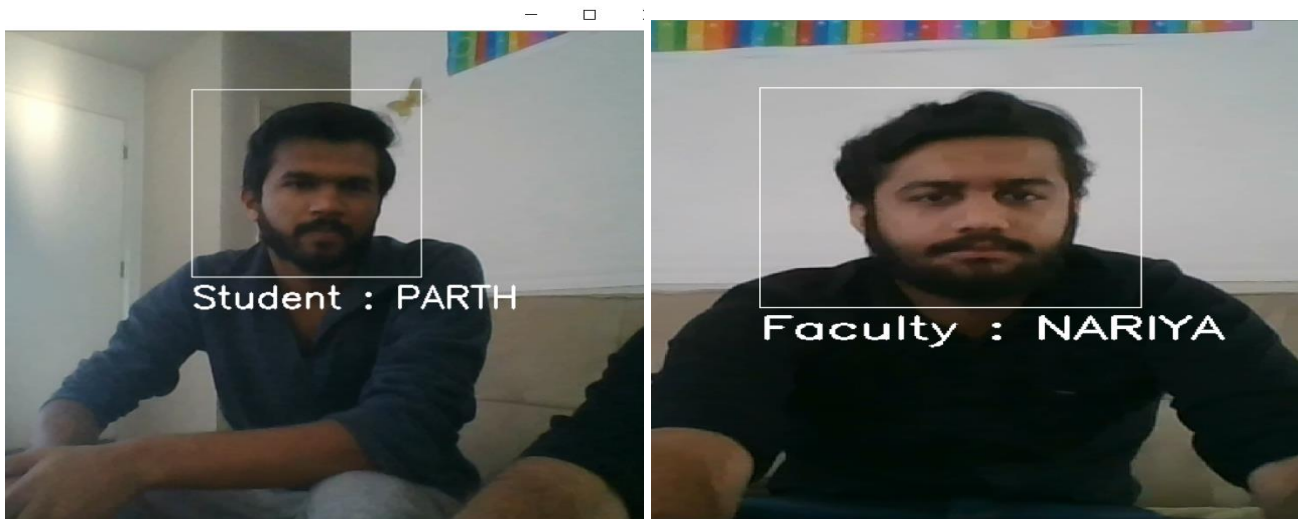
Registration: Faculty Student

Training: Faculty Trainer Student Trainer

Recognizer: Start

Close and Excel file Generation

EXIT CSV



```
C:\Users\Parth Valani\Desktop\CV\log04Dec2019.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
log04Dec2019.txt
1 0 Accpted
2 1 Rejected
3 total students detected 1
length : 48 lines : 3 Ln : 3 Col : 26 Sel : 0 | 0 Windows (CR LF) UTF-8 IN
```

8. COMPARISON

Comparing Local binary pattern with other descriptors

To gain better understanding on whether the obtained recognition results are due to general idea of computing texture features from local facial regions or due to the discriminatory power of the local binary pattern operator, we compared LBP to three other texture descriptors, namely, the gray-level difference histogram, homogeneous texture descriptor, and an improved version of the text on histogram. The recognition rates obtained with different descriptors. It should be noted that no weighting for local regions was used in this experiment. The results show that the tested methods work well with the easiest fb probe set, which means that they are robust with respect to variations of facial expressions, whereas the results with the fc probe set show that other methods than LBP do not survive changes in illumination. The LBP and text on give the best results in the dup I and dup II test sets. We believe that the main explanation for the better performance of the local binary pattern operator over other texture descriptors is its tolerance to monotonic gray-scale changes. Additional advantages are the computational efficiency of the LBP operator and that no gray-scale normalization is needed prior to applying the LBP operator to the face image.

9. DISCUSSION AND CONCLUSION

As the project is a simple project, all the students that are already recognized will be marked their attendance that will be presented by Figure 3. After testing the system, the project finds out that under a controlled environment, the project shows a promising result. Some of the controlled environment includes: a. Background. In order for the face recognition to work at it best, the background need to be static. It is harder to recognize face in a dynamic background. b. Lighting. Lighting also impose an important aspect in recognizing face. An environment with adequate light shows more accurate results rather than a dim environment. c. Facial changes. Some changes to the facial area may cause inconsistency in recognizing the student's faces. For example, students with veil and spectacle are harder to be recognize if they did not wear their veil or spectacle. Hence, it is important to address these issues in the future in order to enhance the accuracy of the face recognition process. In future, we could use CNN to crop the faces to build our dataset and train

our model. Then during real-time inference use a cascade to crop the faces before feeding it into another model. Also, we will use adaptive histogram equalization instead of histogram equalization.

For the conclusion, this project presents the automated system that will aid attendance taking process to be faster, more reliable and accurate. The project has already achieved all the traits that will make this project becomes reliable and useful to be used in order to take student's attendance. Using this system, the attendance taking process does not take a longer time compared to the traditional method. This system also assists the administration team by automatically save student's attendance. However, there are some issues that need to be address in the future. Some of them might include the background dynamics where the system can recognize face even though the background moves. Other issues will be the changes to the student's face and also the illumination of the environment where it should have enough light for the recognition to work. Hopefully, this research will satisfy the growing needs of technological advancement in attendance taking process since the traditional process of attendance taking have many flaws that need to be covered.

10. REFERENCES

Publications:

- [1] T.Ahonen, A.Hadid & M.Pietikäinen, *Face Description with Local Binary Patterns: Application to Face Recognition*. Draft, 5th June 2006.
- [2] T.Ahonen, A.Hadid & M.Pietikäinen. *Face Recognition with Local Binary Patterns*. Machine Vision Group, InfoTech. University of Oulu, Finland. T.Pajdla and J.Matas (Eds): ECCV 2004, LNCS 3021, pp.469-481, 2004. Springer-Verlag Berlin Heidelberg 2004.
- [3] A.Hadid & M.Pietikäinen. *A Hybrid Approach to Face Detection under Unconstrained Environments*. Machine Vision Group, Dept. of Electrical and Information Engineering. University of Oulu, Finland.
- [4] A.Hadid, M.Pietikäinen & T.Ahonen. *A Discriminative Feature Space for Detecting and Recognizing Faces*. Machine Vision Group, InfoTech Oulu and Dept. of Electrical and Information Engineering. University of Oulu, Finland.
- [5] T.Ojala, M.Pietikäinen & T.Mäenpää. *Multiresolution gray-scale and rotation invariant texture classification with Local Binary Patterns*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 971-987, July 2002.
- [6] G.Heusch, Y.Rodriguez & S.Marcel. *Local Binary Patterns as an Image Preprocessing for Face Authentication*. IDIAP Research Institute, Martigny, Switzerland. Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland.
- [7] Y.Rodriguez & S.Marcel. *Face Authentication using Adapted Local Binary Pattern Histograms*. IDIAP Research Institute, Martigny, Switzerland.
- [8] S.Marcel, Y.Rodriguez & G.Heusch. *On the Recent Use of Local Binary Patterns for Face Authentication*. IDIAP Research Institute, Martigny, Switzerland. International Journal of Image and Video Processing, Special Issue on Facial Image Processing. May 2007.
- [9] E.Osuna, R.Freund & F.Girosi. *Training Support Vector Machine: an Application to Face Detection*. In Proc. Computer Vision and Pattern Recognition, pages 130-136, June 1997.
- [10] Josep R. Casas, F. Marqués & P.Salembier. *Apunts de l'assignatura: Processament d'imatge*. Image Processing Group, Signal Theory & Comm.

- [11] Dept, UPC. Barcelona, Fall 2004.
- [12] Lindsay I Smith. *A tutorial on Principal Components Analysis*. February 26, 2002.
- [13] *Elias, Shamsul J., et al. "Face recognition attendance system using Local Binary Pattern (LBP)." Bulletin of Electrical Engineering and Informatics 8.1. March, 2019*

Books:

- [14] Richard O. Duda, Peter E.Hart, David Stork. *Pattern Classification*, 2nd Edition Wiley.
- [15] Joan Cabestany, Sergio Bermejo. *Sistemas conexionistas para el tratamiento de la información*, Edicions UPC, Barcelona 2003.

Reference Manuals:

- [16] *Intel® Open Source Computer Vision Library. CXCORE, CV, Machine Learning and HighGUI Reference Manuals*, Intel Corporation. PDF Files (2006).

Web Pages:

- [17] <http://opencvlibrary.sourceforge.net>: OpenCV Wiki-pages
- [18] <http://tech.groups.yahoo.com/group/OpenCV/>: OpenCV Open Source Computer Vision Library Community
- [19] <http://citeseer.ist.psu.edu/burges98tutorial.html>: A Tutorial on Support Vector Machines for Pattern Recognition - Burges (ResearchIndex)
- [20] <http://www.ee.oulu.fi/mvg/page/publications>: University of Oulu - Machine Vision Group
- [21] <http://www.ee.oulu.fi/research/imag/texture/lbp/about/LBP%20Methodology.pdf>: University of Oulu - LBP methodology
- [22] <http://www.humanscan.de/support/downloads/facedb.php> : The BioID Face Database
- [23] http://www.ysbl.york.ac.uk/~garib/mres_course/2005/Lecture5.ppt : Resampling techniques.