

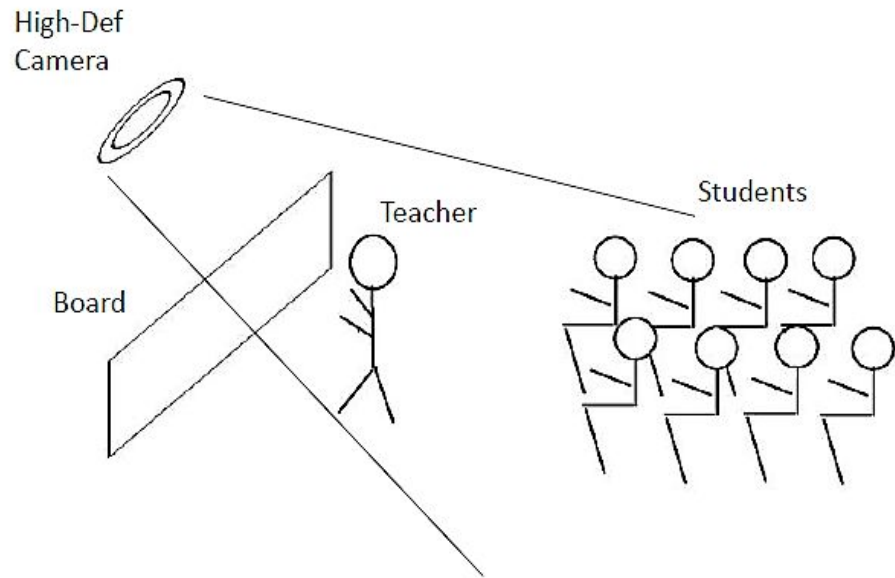


LOG MONITORING SYSTEM

APOORV GAURAV AGARWAL (BE/25034/15)
PRAGEET N GUPTA (BE/25068/15)
SHUBHAM MATHUR (BE/25005/15)

HOW THE PROJECT WORKS

Basic Structure





Phases of the Project

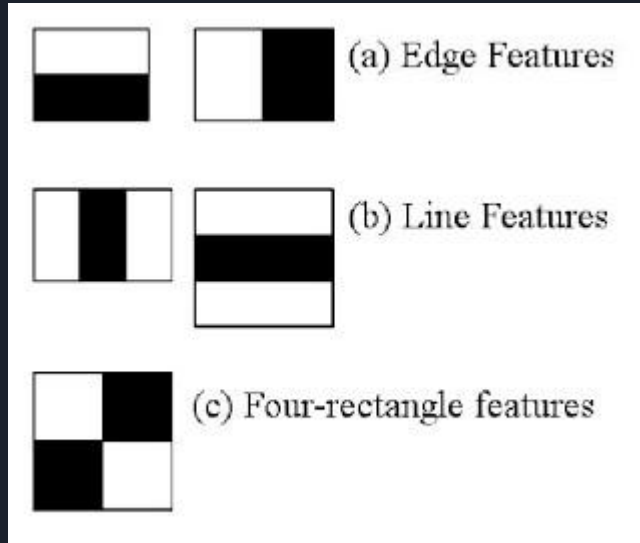
1. FACE DETECTION (USING HAAR CASCADES)
2. FACE RECOGNITION (USING LBPH)
3. DATABASE MANAGEMENT



BASICS

1. Object Detection using Haar feature-based cascade classifiers is an effective object detection method which is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.
2. Initially, the algorithm needs a lot of **Positive images** (images of faces) and **Negative images** (images without faces) to train the classifier.
3. Then these features are extracted from the classifier. Each feature is a single value obtained by subtracting sum of pixels under white zone from sum of pixels under black zone.

WHAT ARE WHITE AND BLACK ZONES



For accuracy in face detection some features are extracted in the classifier. For each feature calculation, we need to find sum of pixels under white and black rectangles. As this computation is really taxing and would make the process slow the concept of integral images is used. It simplifies calculation of sum of pixels to an operation involving just four pixels , however large may be the no of pixels.

WHAT ARE INTEGRAL IMAGES

Original

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

Integral

5	7	10	14	15
6	13	20	26	30
8	17	25	34	42
11	25	39	52	65
15	30	47	62	81

$$5 + 2 + 3 + 1 + 5 + 4 = 20$$

To calculate the integral value of a pixel in the image matrix we need to calculate the sum of all pixel values to the left and on top of that pixel position and add it to the value of the present pixel.

For eg , here :

Sum of all pixels above and to the left of the marked pixel = $(5+2+3+1+5+4)=20$



ADABOOST TRAINING ALGORITHM

The training process uses AdaBoost to select a subset of features and construct the classifier. A large set of images, with size corresponding to the size of the detection window, is prepared. This set must contain positive examples for the desired filter (e.g. only front view of faces), and negative examples (non-faces).

Steps of computation :

- Initialise the weights as $w = 1/(2P_-)$ and $w = (1/2P_+)$ for all images where $y_l = 0$ and $y_l = 1$, where y_l is 0 when non-face images are used and $y_l = 1$ when image with face is used.
- Normalize the weights as follows so that $w(i), l$ is a probability distribution:

$$\frac{w_{i,l}}{\sum_{j=1}^n w_{i,j}} \rightarrow w_{i,l}$$

- For each feature j , train a classifier h_j which is restricted to using a single feature. The classifier's error rate is evaluated with respect to $w(i), l$:

$$\varepsilon_j = \sum_{l=0}^{L-1} w_{i,l} |h_j(x_l) - y_l|$$

- Choose classifier $h(i)$ with lowest error values, ε_i .
- Update the weights :

$$w_{i+1,l} = w_{i,l} \beta_i^{1-\varepsilon_i}$$

$$\beta_i = \frac{\varepsilon_i}{1 - \varepsilon_i}$$

- The final classifier is :

$$h(x) = \begin{cases} 1, & \sum_{i=0}^{I-1} \alpha_i h_i(x_i) \geq \frac{1}{2} \sum_{i=0}^{I-1} \alpha_i \\ 0, & \text{otherwise} \end{cases}$$

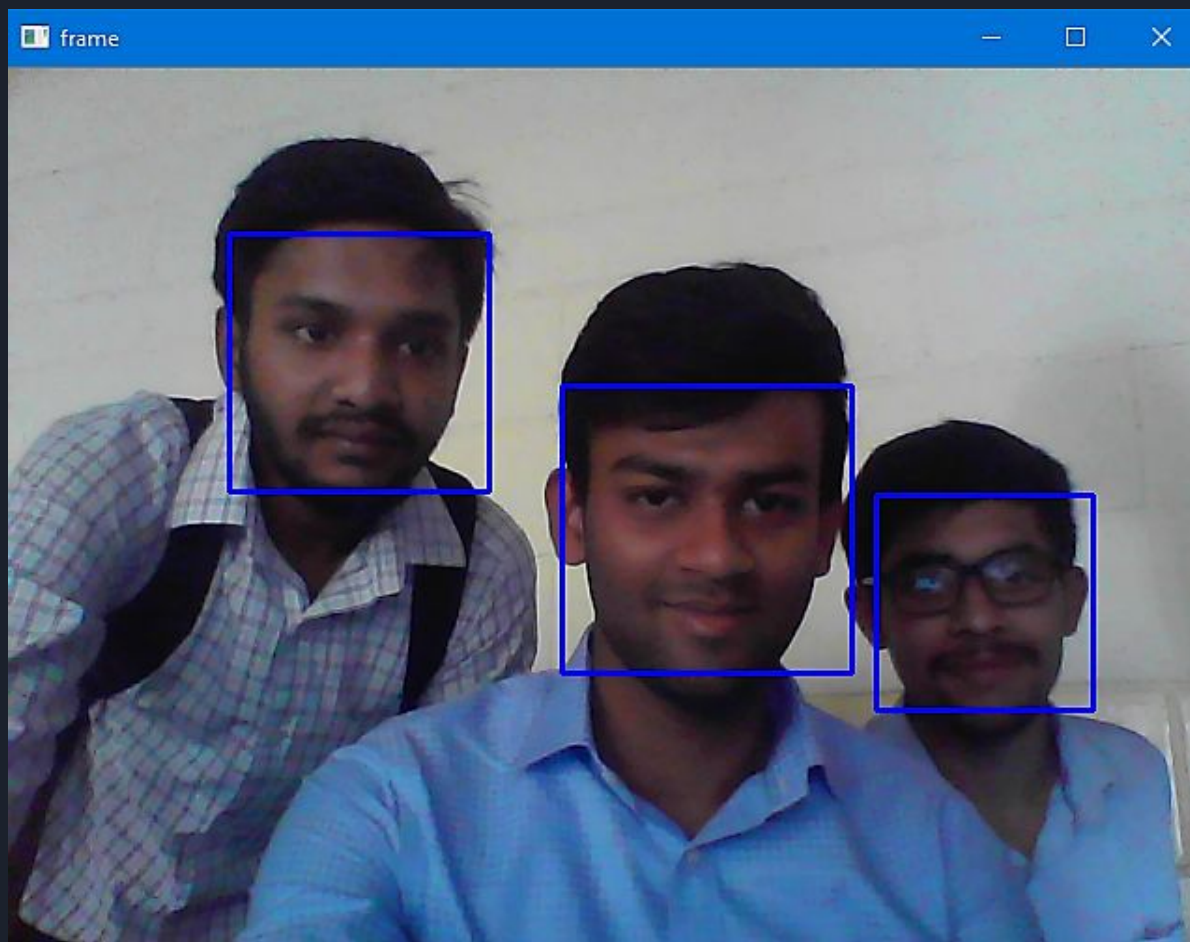
$$\alpha_i = \log \frac{1}{\beta_i}$$



CASCADING EFFECT

A cascade of gradually more complex classifiers achieves detection rates. The evaluation of the strong classifiers generated by the learning process can be done quickly, but it isn't fast enough to run in real-time. For this reason, the strong classifiers are arranged in a cascade in order of complexity, where each successive classifier is trained only on those selected samples which pass through the preceding classifiers. If at any stage in the cascade a classifier rejects the sub-window under inspection, no further processing is performed and continue on searching the next sub-window. The cascade therefore has the form of a degenerate tree. In the case of faces, the first classifier in the cascade – called the attentional operator – uses only two features to achieve a false negative rate of approximately 0% and a false positive rate of 40%. The effect of this single classifier is to reduce by roughly half the number of times the entire cascade is evaluated.

REAL TIME TESTING (FACE DETECTION PHASE)





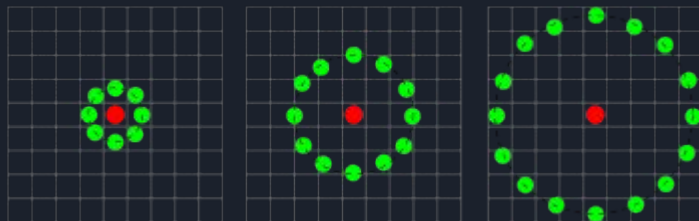
LBPH Face Recognizer

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 consider the results as a binary number. It has since been found to be a powerful feature for texture classification; it has further been determined that when LBP is combined with the Histogram of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.

LBPH Face Recognition Algorithm

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

- Divide the examined window into cells (e.g. 16x16 pixels for each cell).
- For each pixel in a cell, compare the pixel to each of its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise or counter-clockwise.
- Where the center pixel's value is greater than the neighbor's value, write "0". Otherwise, write "1". This gives an 8-digit binary number (which is usually converted to decimal for convenience).






- Compute the histogram, over the cell, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center). This histogram can be seen as a 256-dimensional feature vector.
- Optionally normalize the histogram.
- Concatenate (normalized) histograms of all cells. This gives a feature vector for the entire window.



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.



opencv_lbphfaces:

threshold: 1.7976931348623157e+308

radius: 1

neighbors: 8

grid_x: 8

grid_y: 8

histograms:

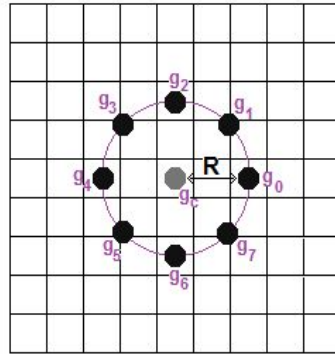
- !!opencv-matrix

rows: 1

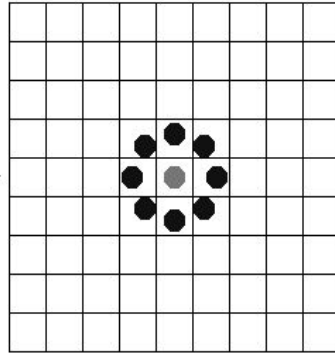
cols: 16384

dt: f

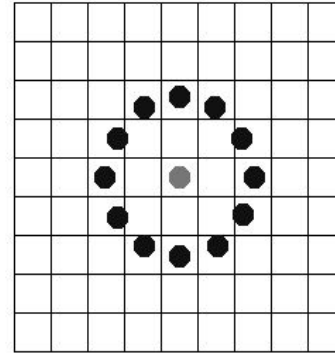
data: [3.40264663e-02, 9.45179630e-03, 0., 5.67107741e-03,
1.89035926e-02, 0., 1.89035921e-03, 5.67107741e-03, 0., 0.,
0., 0., 9.45179630e-03, 3.78071843e-03, 0., 2.26843096e-02,
3.59168239e-02, 1.70132332e-02, 1.89035921e-03,
3.78071843e-03, 1.89035921e-03, 3.78071843e-03, 0., 0.,
1.32325143e-02, 5.67107741e-03, 0., 0., 4.34782617e-02,
5.67107741e-03, 2.45746691e-02, 4.34782617e-02, 0., 0., 0.,
0., 1.89035921e-03, 0., 1.89035921e-03, 3.78071843e-03, 0.,
0., 0., 0., 0., 0., 0., 1.51228737e-02, 5.67107741e-03,
0., 0., 0., 0., 0., 0., 1.32325143e-02, 0., 0., 0.,



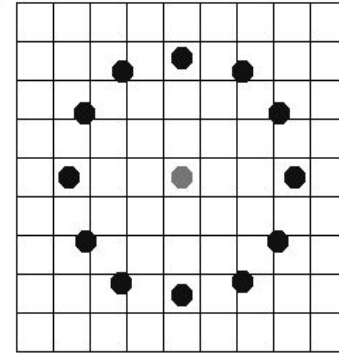
P=8, R=2
(a)



P=8, R=1
(b)




P=12, R=2
(c)




P=12, R=3
(d)

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.




After these computation 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.



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.

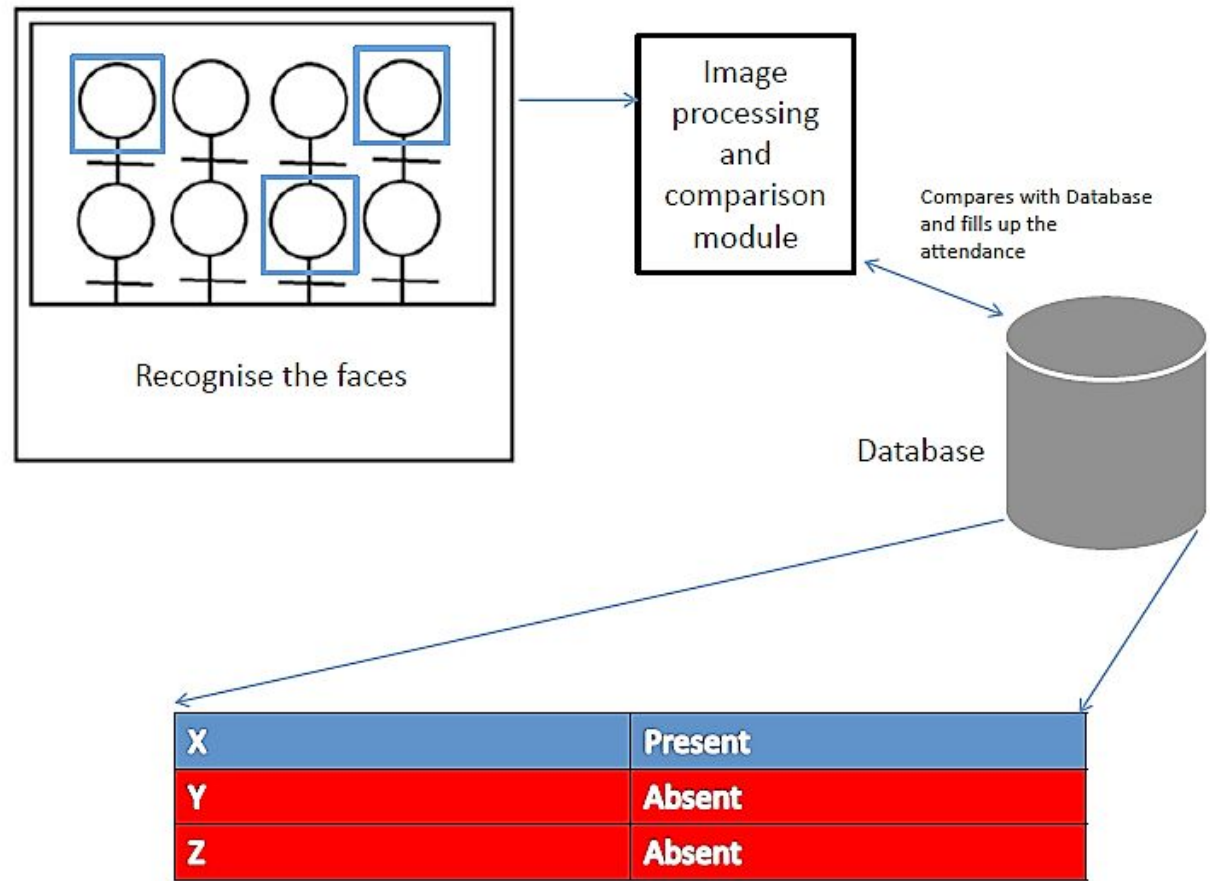
- So to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.
- We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: **euclidean distance, chi-square, absolute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



DATABASE MANAGEMENT PHASE





DATABASE STRUCTURE

DATE	TIME	NAME	OCCURRENCES
2018-04-20	09:00:00	APOORV	6000

- Attendance will be marked in 1 hour time frame.
- If the no of occurrences is greater than 5000 , attendance will be marked.