Face detection algorithms

Rohit Aich Bhowmick
Suman Polley
Shoraj Tomer

Advisor: Kavita Sutar

February 27, 2021

Contents

List of Figures

List of Tables

1 Introduction

Ability to recognise a face is one of the most important trait of intelligence .With advancement in vision technology (i.e. digital camera) and computing power , our curiosity to teach computer to how to recognition a face is evident . In this direction of advancement some of the earliest steps were eigenface,Fisherface and Viola-Jones face detection algorithm.

We will be studying and implementing these 3 classical face recognition algorithms: eigenfaces, Fisherfaces and Viola-Jones face detection.

2 The Eigenfaces algorithm

The Eigenfaces algorithm was proposed by Turk and Pentland **??** in 1991. This is an SVD-based algorithm. The idea is that If we can extract and identify the features that is common to most of the faces then we can teach a computer how to distinguish between a face and a non-face based on those features. The top k features (principal components) from a set of training images are chosen by applying SVD to the co-variance matrix of the data (i.e. PCA). The images are then projected onto the subspace spanned by the chosen vectors. The Euclidean distance between images is calculated to decide whether a test image is part of the given image set or not.

2.1 Description of the algorithm

2.1.1 Deviation matrix

The data consists if m images $x_1, x_2, \ldots, x_m$, each of size $n \times n$, such that each images belongs to one of c classes $\{X_1, \ldots X_c\}$. The images are flattened to column vectors of size $n^2 \times 1$. The average face of the set is defined by $\Psi = \sum_{n=1}^{m} x_n$(See Figure 1). Each face differs from the average by the vector $\Phi_i = x_i - \Psi$. Let A denote the data matrix i.e.

$$A = [\Phi_1 \Phi_2 \ldots \Phi_n],$$

($\dim A = n^2 \times m$). The idea now is to find the top k left singular vectors of A and the corresponding projection W which will project the given images onto the subspace spanned by the chosen vectors. As a result we will have achieved dimensionality reduction and will have retained the information that best describes the variation in the data.

Figure 1: Average face

2.1.2 Eigenvectors calculation

Theoretically, the way to proceed would be to form the co-variance matrix $C = AA^T$ and find the eigenvalues and eigenvectors of C. But $\dim C = n^2 \times n^2$, so this is expensive. Instead, we find the eigenvalues/eigenvectors of $C^T = A^T X$, which is cheaper as well as useful since it gives us the required non-zero eigenvalues and the corresponding eigenvectors. Notice that if v_i are the eigenvectors of $A^T A$, then $A^T A v_i = \lambda_i v_i$ so that $AA^T(Av_i) = \lambda_i(Av_i)$, thus Av_i are the eigenvectors of AA^T. So it is enough to find the vectors $\{v_i\}$. We will denote $L = A^T A$, where $L_{mn} = \Phi_m^T \Phi_n$. And the eigenfaces(see Figure 2) as

$$u_l = \sum_{i=1}^{k} v_{li} \Phi_i \quad l = 1, \ldots, k$$

Figure 2: Top 9 eigenfaces

2.1.3 Projection onto face-space

The eigenface images calculated from the eigenvectors of L span a basis set with which to describe face images. For Yale data-set B We have taken into account only the first 1007 images for training set and found that about 350 eigenfaces were sufficient for a very good description of the set of face images. With $M' = 350$ eigenfaces. A new face image (x) is transformed into its eigenface components by:

$$\omega_k = u_k^T(x - \Psi)$$

The weights form a vector $\Omega_T = [\omega_1, \omega_2, ...\omega'_M]$ that describes the contribution of each eigenface in representing the input face image, treating the eigenfaces as a basis set for face images. And, For a deviation image $\Phi = x - \Psi$, its projection onto the face space is defined by:

$$\Phi_f = \sum_{n=1}^{M'} \omega_i u_i$$

Define ϵ ,the difference between an image(x) and its projection(Φ_f) onto face-space by:

$$\epsilon^2 = ||\Phi - \Phi_f||^2$$

Let θ_f is the threshold for known face and θ_n is the threshold for non-face. At this stage there are 3 possibilities for an input image(x) and its corresponding ϵ: (1) $\epsilon < \theta_f$ meaning image(x) is a known face i.e x $\in$ training set. (2) $\theta_f < \epsilon < \theta_n$ meaning image(x) is a unknown face i.e x is a face and x $\notin$ training set. (3) $\epsilon > \theta_n$,meaning x is a non-face.

Steps of the algorithm in short:

This approach to face recognition involves the following initialization operations:

1. Acquire a set a face images for training.
2. Calculate eigenvectors(eigenfaces) of C that contains the most important features of a face image.
3. Consider only the first K images (eigenfaces). The space spanned by these images is called face space.
4. Project any new image into this face space.Any non-face image must deviate from itself significantly as the basis vectors(eigenfaces) captures features of only face images ,which are supposed to be absent in non-face image.see Figure 3 & Figure 4.

2.2 Experiments and results

2.2.1 Data-sets used

The Table 1 is based on experiments with training model based on Yale data-set B. There are 38 individuals and each of them have 64 images (N.B. There are some bad images that were damaged during the image

acquisition. We have cleaned the dat-set and this refined data-set is available at CroppedYaleB.zip as a zip file). The original data-set is available at yale data-set B site. We have taken into account only the first 16 individuals (consisting of 1007 images)for training set. And tested on 280 unknown face images and 100 non-face images available at nonface.zip and nonface.zip. Of these 287 unknown face images,
140 images(picked at random) are of individuals 17 -38 of yale data-set B site, 40 images are of unknown face images downloaded from various sites (publicly available images) through google search results of frontal face images, 77 frontal-face images of mugshot database, 30 images of yaleData-setA. All the test-face images have been suitably cropped to match the dimensions of the training images. The 100 non-face images were acquired through google search of different non-face images.

Figure 3: a)A known face and b) its projections in face space with $M' = 350$

Figure 4: a)Unknown face image and c)no-face image and their projections b) & d) in face space with M' = 350

2.2.2 Calculating Thresholds

θ_f is the threshold for known face and θ_n is the threshold for non-face. An implementation of eigenface algorithm is available as a jupyter-notebook at this link. We have calculated θ_f by taking 99th percentile of known faces ϵ and 1st percentile of unknown faces and taking their average.Similarly,θ_n by taking 99th percentile of unknown faces ϵ and 1st percentile of non-faces and taking their average.Thus we get θ_f = 4.18057 and θ_n = 14.49589 (after rounding off to 5 significant numbers). These thresholds depends on M'. With increase in M' the projections gets better for all images(see Figure 5 and Figure 6), as a result ϵ gets smaller for all images thus θ_f&θ_f gets smaller.
N.B: Taking the 99th and 1st percentile removes any extreme point's significance in calculating θ_f&θ_n.

Data-set	Results		
	Precision	Recall	F-score
Known Faces	1.0	1.0	1.0
Unknown Faces	0.9860	0.9791	0.9825
Non-Faces	0.9268	0.9500	0.9383

Table 1: Observations - I

2.3 Observations

2.3.1 Variation in M'

It may seem like the more eignefaces we take for training the model the more chance of Under fitting since they would project the unknown face and non-face images better with increase in M'. However this is

Figure 5: a)Unknown face and its projections b),c),d),e) for M'= 100,300,500,700

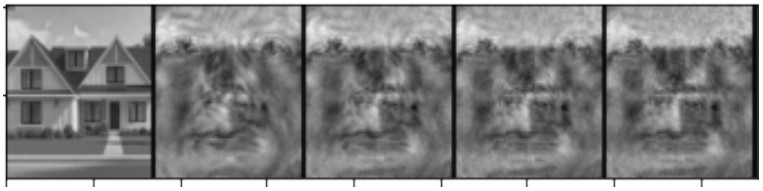


Figure 6: a)Non-face image and its projections b),c),d),e) for M'= 100,300,500,700

partially true . The projection of unknown faces and nonface images images would certainly get better but,so would be for known face images(even more so,since the eigenfaces created from them). In face the classification accuracy would start to fall for nonface images and unknown face images after a certain M' (See Figure 7),where as accuracy of known face images remains 100%.

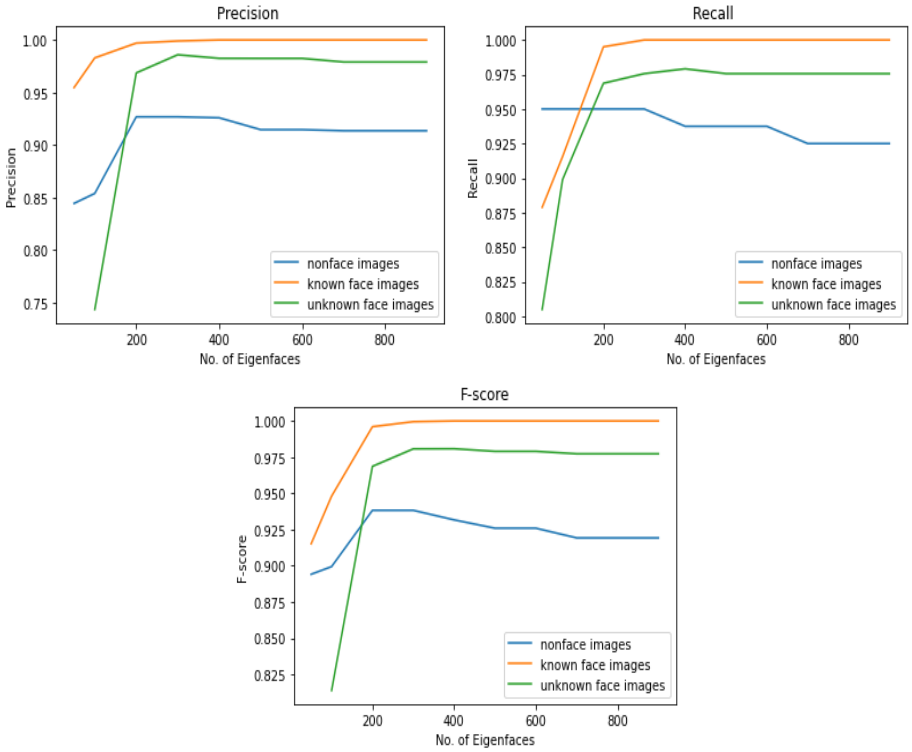


Figure 7: Variation of scores w.r.t M'

Notice that the projection quality does not get significantly better after a certain M' . The reason is, in PCA(eigneface algorithm) the eigenfaces are ranked according to their singular values so,the top eigenfaces contains most of the distinct features ,where as the bottom eigenfaces(see Figure 8 contains no distinct features. Basically they are noise for us.

And with increase in noise in training model the key features that separate face from a non-face have lesser significance.

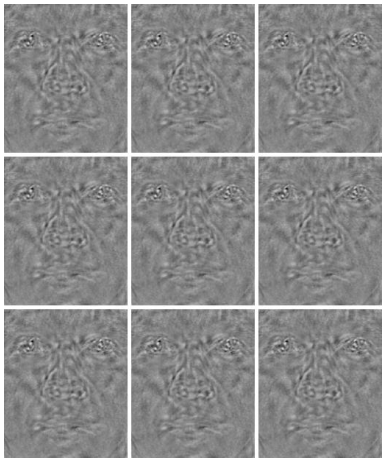


Figure 8: Bottom 9 eigenfaces for $M' = 350$

2.3.2 Recognition

The Eigenfaces approach maximizes the total scatter, which can lead to problems if the variance is generated by an external source(For example: Light Source), because components with a maximum variance over all classes aren't necessarily useful for classification . For example : Eigenface algorithm would classify darker images of all individual as same. So to preserve some discriminative information we applied a Linear Discriminant Analysis and optimized as described in the Fisherfaces method(See Section 3). The Fisherfaces method considers in between class scatter which is great for classification.

3 The Fisherfaces algorithm

The Fisherfaces algorithm is an improvement over the eigenfaces algorithm. It is based on Fisher's linear discriminant (fld) **??**. This method takes into consideration the between-class and within-class scatter and tries to maximize the ratio of the between-class scatter to the within-class scatter. The data is re-arranged (projected onto a suitable subspace) so that the focus is more on linearly separating the classes rather than images themselves. Thus, while PCA (the method used for eigenfaces) retains the variations that come about due to changes in lighting and facial expressions, the Fisherfaces method overcomes these to some extent by focusing on the between-class scatter rather than the total scatter.

3.1 Description of the algorithm

Let X denote the image (data) matrix i.e.

$$X = [x_1|x_2|\dots|x_n],$$

($\dim X = n^2 \times m$) as in the eigenfaces algorithm. Let S_T denote the total scatter matrix of the given data,

$$S_T = \sum_{i=1}^{m}(x_i - \mu)(x_i - \mu)^T \quad \text{where} \quad \mu = \frac{1}{m}\sum_{i=1}^{m} x_i.$$

μ is the mean or the average face of the data. Then $S_T = XX^T$ (i.e. the matrix C of the eigenfaces algorithm).

Let S_B and S_W denote the between-class and within-class scatter matrices, respectively. They are defined as:

$$S_B = \sum_{i=1}^{c} n_i(\mu_i - \mu)(\mu_i - \mu)^T$$

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

where n_i is the number of images in class X_i and $\mu_i = \frac{1}{n_i}\sum_{i=1}^{n_i} x_i$ is the mean image of class X_i.
This method require to find a projection matrix W such that it maximizes the ratio of between class scatter to within class scatter. Therefore, the optimal projection W_{opt} is chosen as the matrix with orthonormal columns which maximizes the ratio of determinant of between class scatter matrix and within class scatter matrix of projected samples, i.e.,

$$W_{opt} = \arg\max_W \left| \frac{W^T S_B W}{W^T S_W W} \right|$$

$$= [\mathbf{w_1}\ \mathbf{w_2}\ \cdots\ \mathbf{w_m}]$$

Where $\{w_i | i = 1, 2, ...m\}$ is the set of generalized eigenvectors of S_B and S_W corresponding to m largest eigenvalues, i.e.,

$$S_B W_i = \lambda_i S_W W_i, i = 1, 2, ...m$$
$$S_W^{-1} S_B W_i = \lambda_i W_i, i = 1, 2, ...m$$

If S_W is nonsingular matrix then required projection matrix W_{opt} obtained by taking first m eigen-vectors of $S_W^{-1} S_B$ as column vectors works very well. Note that here m have upper bound $c - 1$, where c is the number of classes we have.

Note that the within-class scatter matrix $S_W \in \mathbb{R}^{n*n}$ is always singular as number of images for the learning set is far less than number of pixels in each image. To overcome this problem we project the image set to a lower dimensional space so that resulting within class scatter matrix S_W is non-singular. For dimensionality reduction we use PCA, it reduce the dimension of feature space to $N - c$,i.e., total number of images subtract the number of classes. The Matrix W_{pca} is same and obtained in same manner as in eigenface algorithm. Then we find the between scatter matrix and within scatter matrix into this reduced dimensional spcae defined as

$$S_{BB} = W_{pca}^T S_B W_{pca} \qquad and$$
$$S_{WW} = W_{pca}^T S_W W_{pca}$$

So, the matrix W_{fld} (fisherfaces) is the optimal matrix which maximize the ratio of determinant of the between class matrix to the within class matrix in reduced dimensional space of projected samples, i.e.,

$$W_{opt} = \arg\max_W \left| \frac{W^T W_{pca}^T S_B W_{pca} W}{W^T W_{pca} S_W W_{pca} W} \right|$$

$$W_{opt} = \arg\max_W \left| \frac{W^T S_{BB} W}{W^T S_{WW} W} \right|$$

The matrix W_{fld} is the matrix of generalized eigen-vectors of matrices S_{BB} and S_{WW}, i.e., columns of W_{fld} is the eigen-vectors of the matrix $S_{WW}^{-1} S_{BB}$. Note that W_{fld} reduce the dimension of projected space to $c - 1$, where c is the number of classes. Therefore the required optimal matrix W_{opt} is given by

$$W_{opt}^T = W_{fld}^T W_{pca}^T$$

The order of matrices W_{pca} is $n \times (N - c)$ and W_{fld} is $(N - c) \times m$, where m have upper bound $c - 1$.
Therefore the optimal projection matrix is of size $m \times n$, so it reduced a vector of dimension n (number of pixels in our image) to a vector of dimension m (Where m is very small compared to n). So, our next step we find a list of projections corresponding to each training image we have.

3.1.1 projection and classification

For our training data set, we prepare a list, called projections, which contain the projection vector of training images i.e., $W_{fld}x$, where x is the image vector. Also we prepare a corresponding list have true class corresponding to each image vector.
Now, for classification we use Nearest neighbour classification. Suppose we have a image vector y and we want to find the class of y, Then

1. Find the projection vector $W_{opt}y$

2. Find its distance with all training images

3. Select the projection vector say z which have minimum distance with projection vector corresponding to y Then we classify y have the same class as class of image corresponding to projection vector z

Steps of the algorithm in short:
This approach to face classification involves the following steps:

1. Acquire a set of face images distributed among classes. Further, spilt the data set into training and testing data sets.

2. Find the between-class scatter matrix S_B, within-class scatter matrix S_W and total scatter matrix S_T using training data set.

3. Using the total scatter matrix S_T, we find the matrix W_{pca} i.e., matrix of principle components, where columns of W_{pca} are the eigenvectors of $S_W^{-1}S_B$ corresponding to largest eigenvalues. W_{pca} is used for dimensionality reduction.

4. Using W_{pca}, we find the between scatter matrix S_{BB} and within scatter matrix S_{WW} in new reduced dimension space.

5. Find the fisher face matrix W_{fld} , which maximizes the ratio of between class scatter and within class scatter. columns of W_{fld} are the eigenvectors of the matrix of $S_{WW}^{-1}S_{BB}$.

6. Get the optimal matrix W_{opt}, which is product of matrices W_{pca}^T and W_{fld}^T.

7. For a test image, we find the projection using $W_{opt}^T = W_{fld}^T W_{pca}^T$ matrix and then find its distance with all the images i.e., projection of training images and then classify our test image have the same class as the image having minimum distance.

3.2 Experiments and results

For our experiment we have consider Yale data-set A and Yale data-set B. In Yale data-set A there are 15 individuals having 11 images each. For training purpose we have used 9 images per individual and for testing purpose we have used 2 images per individual selected randomly. In Yale data-set B, there are 38 individuals having 64 images each (There are some bad images that were damaged during the acquistion). We have cleaned the data set and refined data set is available at FaceData.zip as a zip file here. The original data set is same as in the eigenface algorithim. For testing purpose we have used 12 images per individual and rest of the images are used for training purpose. Testing images are selected randomly and varying under different light conditions and facial expression. For convenience of calculation, we have resized Images (both training and testing) to size $100 * 100$.

An implementation of fisherface algorithm is available as a jupyter notebook at this link.

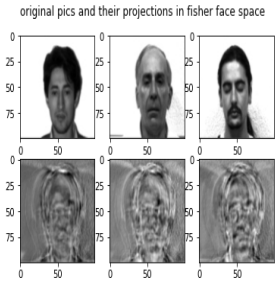


Figure 9: Yale Data Set - A

For yale Data set-A fisher face algorithm gives us accuracy score of 0.90, while for yale Data Set-B, it gives us accuracy score of 0.98. In both cases we have used The Nearest Neighbor classification in projected space.

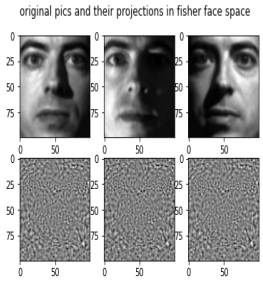


Figure 10: Yale Data Set - B

4 The Viola-Jones algorithm

The Viola-Jones algorithm was proposed by Paul Viola and Michael Jones in 2001. This is one of the first algorithm which provided very good object detection rates in real-time. The main idea lies in the irregular colour composition on a human face due to shadow; which we characterize using the Haar features, which are some rectangle-like features described in the algorithm. Also, we learn about the summed area table for the calculation of integral images which helps in the computation of the Haar features in a much efficient manner. We select all the important Haar features in the image and then we check for the value of the Haar feature, if it is above a certain threshold or not, to determine whether it is a face or a non-face image.

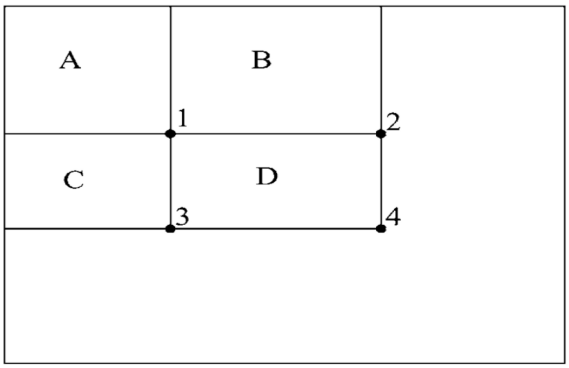
4.1 Description of the algorithm

The algorithm can be briefly described in the following stages:

- Haar Features and Integral Image
  We divide the image frame into a grid of rectangles. Now we select a subset of this rectangle grid to form a Haar feature, here we are using 3 types of Haar Features; two-rectangle features, three-rectangle features and finally a four-rectangle feature. The regions have same shape and size, and are horizontally or vertically adjacent. We compute the sum of the pixels in the light region and subtract it from the sum of the pixels in the dark region to compute the Haar feature values.
  Now to make our algorithm more efficient, we use the concept of Integral images or Summed area table to compute the Haar feature values. So, the integral image at the location x, y contains the sum of the pixels above and left of x, y inclusive.

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

  where $ii(x, y)$ is the integral image and $i(x, y)$ is the pixel value in the original image.
  Here, we observe that using the integral image, we can compute the rectangular sum using the four vertices of the considered rectangle.



  The sum of the pixels within the rectangle D can be computed using the integral image value at points 1, 2, 3 and 4, it is given by 4+1-(2+3).

- AdaBoost Algorithm and Classifiers

  This is a machine learning algorithm for selecting the best subset of features among all available features. We first initialize the weights of each training example as $w_i = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negative and positive samples.
  Firstly, we normalize the weights as $w_i = \frac{w_i}{\sum_j w_j}$. Now, training the weak classifiers is the most computationally expensive part of this algorithm as because, each time a new weak classifier is selected as the best, all of them have to be retrained since the training examples are weighted differently. For each feature, we sort the training examples based on feature value to find the optimal threshold and polarity. For each element in the sorted list, we have four sums; the total sum of positive example weights T^+, the total sum of negative example weights T^-, the sum of positive weights below the current example being considered S^+ and the sum of negative weights below the current example being considered S^-. The error is given by:

$$e = min(S^+ + T^- - S^-, S^- + T^+ - S^+)$$

  Threshold is set to the value of the feature at which the error is minimum. So, this way we train all the weak classifiers and they will be returned in an array. Now we select the best weak classifier amongst these by iterating through each classifier and calculating the average weighted error of each one. We update the weights as,

$$w_i = w_i \beta_t^{1-e_i}$$

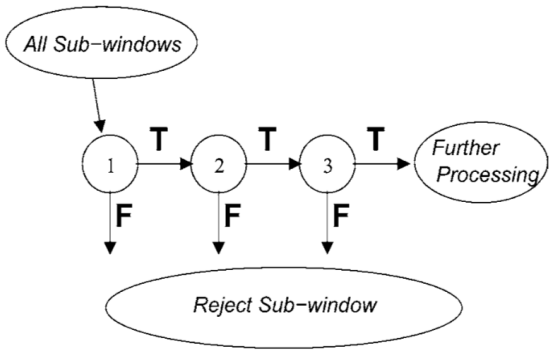
  where $\beta_t = \frac{e_t}{1-e_t}$ and $e_i = 0$ if the considered example is classified correctly and 1 otherwise. Here, e_t denotes the error of the best classifier for the t-th training example. Now, we store the value of $\alpha_t = log(\frac{1}{\beta_t})$ and the best weak classifiers in two different arrays.
  Finally, we construct the strong classifier using our weak classifiers.

$$C(x) = \begin{cases} 1 & if \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}$$

  where h_t is the t-th best weak classifier.

- Cascade Classifier

  A Cascade Classifier is a multi-stage classifier that can perform detection quickly and accurately. Each stage consists of a strong classifier produced by the AdaBoost Algorithm. From one stage to another, the number of weak classifiers in a strong classifier increases. An input is evaluated on a sequential basis. If a classifier for a specific stage outputs a negative result, the input is discarded immediately. In case the output is positive, the input is forwarded onto the next stage.



  The training of each classifier in the Cascade classifier is similar to the training of the regular Viola-Jones algorithm. The only difference here is that after the first classifier that trains on all the training examples, each subsequent classifier is trained on all the positive examples and the wrongly classified negative examples by the previous classifier, i.e, the false positives. In our algorithm, we allow the user to put in the number of classifiers they wish to use for every layer, which can be input as an array of integers, for example, [5,10,15,20] will create four layers of classifiers with number of features 5, 10, 15 and 20 in each layer respectively. The main idea in the cascade is to reduce the false positive rate.

4.2 Experiments and results

4.2.1 Observations for standalone classifiers

- UFI = unconstrained facial images data-set ufi-cropped
- the testing data-set is from UFI
- Only frontal face images from the UFI data-set have been included. The images have been cropped to 24×24 pixels size.

1. Number of classifiers = 10

Data-set	Size of data-set	Training			Testing		
	(Faces,non-faces)	False positives	False negatives	Accuracy	False positives	False negatives	Accuracy
NIST	(10,30)	0/30	0/10	100%	1/50	6/50	93%
NIST	(100,300)						
NIST	(300,900)	53/900	24/300	93.58%	5/50	5/50	90%
UFI	(100,300)	86/300	55/100	64.75%	13/50	23/50	64%

Table 2: Observations - I

2. Number of classifiers = 20

Data-set	Size of data-set	Training			Testing		
	(Faces,non-faces)	False positives	False negatives	Accuracy	False positives	False negatives	Accuracy
NIST	(10,30)	0/30	0/10	100%	3/50	8/50	89%
NIST	(100,300)	1	1	1%	1	1	1%
NIST	(300,900)	53/900	12/300	94.58%	6/50	5/50	89%
UFI	(100,300)	61/300	5/100	83.5%	11/50	27/50	62%

Table 3: Observations - I

3. Number of classifiers = 30

Dataset	Size of dataset	Training			Testing		
	(Faces,non-faces)	False positives	False negatives	Accuracy	False positives	False negatives	Accuracy
NIST	(10,30)	0/30	0/10	100%	2/50	10/50	88%
NIST	(100,300)	1	1	1	1	1	1
NIST	(300,900)	7/900	7/300	98.83%	1/50	6/50	93%
UFI	(100,300)	0/300	0/100	100%	3/50	41/50	56%

Table 4: Observations - I

4.2.2 Observations for cascaded classifiers

1. Cascade specification: 10 layers each consisting of 10 classifiers:

2. Cascade specification: 10 layers each consisting of 20 classifiers:

3. Cascade specification: 10 layers each consisting of 30 classifiers:

Data-set	Size of data-set	Training			Testing		
	(Faces, non-faces)	False positives	False negatives	Accuracy	False positives	False negatives	Accuracy
NIST	(10,30)	0/30	0/10	100%	1/50	6/50	93%
NIST	(100,300)	1	1	1%	1	1	1%
NIST	(300,900)	0/900	21/300	98.25%	2/50	6/50	92%
UFI	(100,300)	0/300	0/100	100%	2/50	32/50	66%

Table 5: Observations - II

Data-set	Size of data-set	Training			Testing		
	(Faces,non-faces)	False positives	False negatives	Accuracy	False positives	False negatives	Accuracy
NIST	(10,30)	0/30	0/10	100%	3/50	8/50	89%
NIST	(100,300)	1	1	1%	1	1	1%
NIST	(300,900)	0/900	2/300	99.83%	0/50	5/50	95%
UFI	(100,300)	0/300	0/100	100%	1/50	39/50	60%

Table 6: Observations - II

In the following table we list our observations for standalone and cascaded classifiers trained on 3 datasets: NIST with 100 faces and 300 non-faces, NIST with 300 faces and 900 non-faces and UFI with 100 faces and 300 non-faces. The trained classifiers are then tested in 4 datasets: NIST, UFI, AT& T and the Yale dataset. Each testing dataset contains 50 faces and 50 non-faces.

A *false positive* is declared when a non-face is detected as a face; a *false negative* is declared when a face is detected as a non-face.

For brevity, we have color-coded the observations with the number of classifiers used as follows: red denotes 10 classifiers were used for that observation, blue denotes 20 classifiers were used and green denotes the result corresponding to 30 classifiers.

Data-set	Size of data-set	Training			Testing		
	(Faces,non-faces)	False positives	False negatives	Accuracy	False positives	False negatives	Accuracy
NIST	(10,30)	0/30	0/10	100%	2/50	9/50	89%
NIST	(100,300)	1	1	1	1	1	1
NIST	(300,900)	0/900	0/300	100%	1/50	6/50	93%
UFI	(100,300)	0/300	0/100	100%	1/50	37/50	62%

Table 7: Observations - II

Testing datasets ↓		Training datasets					
		NIST (100 faces, 300 non-faces)		NIST (300 faces, 900 non-faces)		UFI (100 faces, 300 non-faces)	
		Standalone classifiers	Cascaded classifiers	Standalone classifiers	Cascaded classifiers	Standalone classifiers	Cascaded classifiers
NIST	False positives	1, 1, 1	0, 1, 1	6, 7, 3	2, 2, 3	18, 16, 1	3, 2, 2
	False negatives	11, 5, 5	11, 5, 7	39, 39, 44	40, 40, 38	34, 13, 9	3, 2, 2
	Accuracy	88, 94, 94	89, 94, 92	55, 54, 53	58, 58, 59	48, 71, 90	92, 97, 95
UFI	False positives	3, 1, 0	3, 1, 0	5, 6, 1	2, 0, 1	13, 11, 3	2, 1, 1
	False negatives	31, 31, 38	32, 31, 41	5, 5, 6	6, 5, 6	23, 27, 41	32, 39, 37
	Accuracy	66, 68, 62	65, 68, 59	90, 89, 93	92 95, 93	64, 62, 56	66, 60, 62
AT & T	False positives	3, 1, 0	3, 1, 0	5, 6, 1	2, 0, 1	13, 11, 3	2, 1, 1
	False negatives	41, 47, 47	41, 47, 47	49, 49, 50	48, 46, 47	27, 34, 39	33, 35, 36
	Accuracy	56, 52, 53	56, 52, 53	46, 45, 49	50, 54, 52	60, 55, 58	65, 64, 63
Yale	False positives	1, 1, 1	0, 1, 1	6, 7, 3	2, 2, 3	18, 16, 1	3, 2, 2
	False negatives	14, 7, 5	14, 7, 7	46, 46, 47	48, 47, 48	43, 12, 9	6, 5, 3
	Accuracy	85, 92, 94	86, 92, 92	48, 47, 50	50, 51, 59	39, 72, 90	91, 93, 95

Table 8: Red : 10 classifiers, Blue: 20 classifiers, Green: 30 classifiers