AdaBoost Face Detection

Hamed Masnadi-Shirazi

Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, California
hmasnadi@ucsd.edu

Abstract

Viola and Jones [1] introduced a new and effective face detection algorithm based on simple features trained by the AdaBoost Algorithm, Integral Images and Cascaded Feature sets. This paper attempts to replicate their results. The Feret Face data set is used as the training set. The AdaBoost Algorithm, simple feature set and Integral Images are briefly explained and implemented in our Matlab based program. A series of ten best features were identified out of a set of close to fifty thousand. These best features were used to produce probability of error plots. Finally our face detection Algorithm is implemented on a series or random Images taken from the internet. More than just ten best features are needed to have a face detector comparable to the two hundred best features of Viola and Jones [1] but the face detector still performs well and anyone can use our program included in the Appendix to implement this effective face detection algorithm and train as many best features as suited for their application..

1. Introduction

Face detection and recognition has become an increasingly researched area. The Viola and Jones [1] method for face detection is an especially successful method as it has a very low false positive rate, can detect faces in real time and yet is very flexible in the sense that it can be trained for different levels of computational complexity, speed and detection rate suitable for specific applications. What makes this algorithm even more attractive is the fact that it can be implemented with slight changes to detect many other objects as well.

Using a set of two hundred best features, Viola and Jones [1] were able to produce a 95% diction rate and a 1 in 14084 false positive rate. They were also able to detect faces within a 380x280 image in less than 0.7 seconds. Such high performance makes this method one of the best face detection algorithms. As mentioned above the same algorithm can be applied to detecting other objects as well and Viola, Jones and Snow [4] have successfully used this method for detecting pedestrians. Considering the above, it is highly desirable to be able to implement this versatile method for anyone who might want to do research in this area.

1.1 Overview

In This paper Section 2 will discuss the simple features and integral images. Section 3 will discuss the AdaBoost training. Section 4 will specifically discuss our program and the methods we used to implement this Algorithm. Section 5 will discuss our results. Appendix A will have a set of example runs of our program on images and Appendix B will have the complete listing of our Matlab program.

2. Features and Integral Images

The first thing to keep in mind is that the Viola Jones [1] method is a feature based detection scheme. So a pool of features must be created and a scheme used to find the "good" features. "Good" features are features that best discriminate between faces and non-faces. Many different forms of feature pools can be created. A desirable feature pool would be one that is exhaustive, has feature forms that can describe the object and has features that can be applied to our images and computed efficiently. The rectangle features used have all the above conditions. An example of such features can be seen in Figure 1.

Figure 1: Example rectangular features used in our program. (From left to right) feature numbers 1, 100, 1000, 15000, 25000, 49000.

It should be mentioned that these features are 24x24 pixels. The image pixels that fall within this space and are under the black pixels are subtracted from the image pixels that are under the white feature pixels (or vice versa). The resulting number is considered the output of this feature applied to our image.

2.1 Integral Images

It might seem computationally expensive to find the output of each feature as described above, but using a clever method called the Integral Images; one can easily and effectively find the output. In this method a new image is created for each test image such that every new pixel is the sum of the pixels above and to the left of it.

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

The same is done for each feature in our feature bank to the extent that each feature is represented only by a series of numbers that define how specific parts of the Integral Image should be added or subtracted to produce the feature output. This will be further explained in Section 4 when we introduce our specific program. Using this method it is no longer necessary to add or subtract individual pixels. The Image is scanned once and the Integral Image is found. After this stage to find the output of a feature applied to our image we just add and subtract a limited number of our Integral Image pixels.

3. Adaboost Training and Feature Selection

Now that we have a set of features and an efficient way to compute the output of each feature, we describe how the best features are found. To realize the importance of this stage it should be mentioned that using 24x24 pixel features we produced an exhaustive set of 49,554 features. A method must be used that can reduce this feature set and give us the best features that discriminate the faces from non-faces and also complement each other.

3.1 Weak Classifiers

Before we move on to explain the training algorithm the concept of a weak classifier should be explained. A training set of labeled faces and non-faces is prepared by scaling each image to a 24x24 image and normalizing it. For each feature a weak classifier is defined by applying the feature to the training set. A weighted histogram is produced and an optimal threshold that best separates the faces from non-faces is found. A parity is also found that describes weather the face outputs are above or below the threshold. So each weak classifier $h_{j(x)}$ has a feature f_j, a threshold θj and a parity pj assigned to it. (Finding the threshold of each weak classifier is an optimization step and any optimization algorithm that minimizes the error of classification can be used.) In our implementation the output of our weak classifier is 1 if the image x is classified as a face and -1 if it is classified as a non-face. Mathematically this is equivalent to:

$$h_j(x)=\begin{cases}1 \leftarrow p_j f_j(x) < p_j \Theta_j \\ -1 \leftarrow Otherwise\end{cases}$$

Figure 2: First three features found using naïve approach. It is readily seen that they are slightly scaled versions of the same basic feature.

Care should be taken to realize that a weak classifier is not a true classifier in our final algorithm. It is nothing more than a weighted histogram of the outputs of one feature applied to all our training data set along with an optimum threshold that separates the face from non-faces outputs and hence provides an error of classification associated with that particular feature.

3.2 Training and AdaBoost Algorithm

An intuitive yet naïve procedure for training our features and finding the set of best features would be to construct the weak classifier for each feature and rank the features in order of smallest error of classification. This simple method unfortunately does not work because doing so will produce the first best feature and a set of features that look very similar to the first but are slightly scale or shifted. These features are all basically the same and fail on the same test images. Figure 2 shows the first few best features found using this naïve approach.

The AdaBoost algorithm fixes this problem by changing the weights used in computing the classification error of our weak classifier. A small error is now weighted more and this ensures that our first best feature and any other feature similar to it will not be chosen as our second best feature. This means that our second best feature is no longer similar to our first best feature and a whole different feature is selected as our next best feature. This second best feature ideally compliments our first best feature in the sense that it is successful at classifying faces that the first best feature failed on. This process is repeated to find as many best features as desired. This is generally a very time consuming processes.

Once a number of best features have been found, they can be used to detect faces in a test image. Each feature votes on weather it thinks the test image is a face or not. Each features vote is weighted in log-inverse proportion to the error of that feature. So a feature with a smaller error gets a heavier weighted vote. See Table 1 for complete algorithm.

- Given example images $(x_1, y_1), \ldots, (x_n, y_n)$ where $y_i = 0, 1$ for negative and positive examples respectively.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, \ldots, T$:
  1. Normalize the weights,
  $$w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$$
  so that w_t is a probability distribution.
  2. For each feature, j, train a classifier h_j which is restricted to using a single feature. The error is evaluated with respect to w_t, $\epsilon_j = \sum_i w_i |h_j(x_i) - y_i|$.
  3. Choose the classifier, h_t, with the lowest error ϵ_t.
  4. Update the weights:
  $$w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$$
  where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
- The final strong classifier is:
$$h(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
where $\alpha_t = \log \frac{1}{\beta_t}$

Table 1: The AdaBoost algorithm. Each for each *(t)* a new best feature is found.

4. Our Implemented Algorithm

In this section we explain the methods and procedures used and give a precise description of our approach to implementing the different aspects of training our face detector.

4.1 The Features

An exhaustive set of 49,554 features were produces using 24x24 pixel features with a minimum rectangle size of 8x8 pixels. Figure 1 shows a few of these features. But this is not how each feature is stored in our program. Storing 49,554 individual 24x24 pixels images would take up a significant amount of memory space. As previously explained it would also be computationally expensive and inefficient to use the raw features in our algorithm. In order to deal with these problems we have used Integral Images. The Integral Image of each feature is found and a simple representation for that feature is produced. This representation uses a few numbers to tell us how to apply each feature to a training image. Each number tells us how the integral image pixel at that location should be added or subtracted to produce the feature output. Figure 3 shows two features and their representation as stored in our Matlab program.

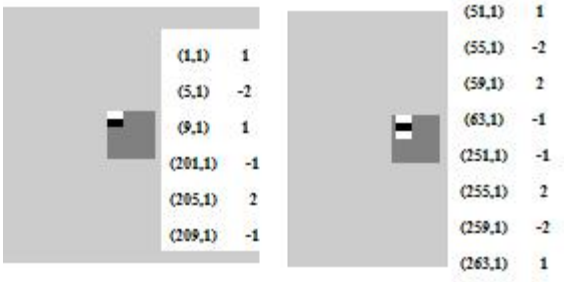

Figure 3: Each feature is reshaped as a sparse column vector with elements at the corners of the black and white rectangles. The location of the corners (left columns) and the number multiplied by the integral image at that location (right columns) is stored. The left image is for feature#1 and the right image is for feature#15000.

4.2 The Training Data Set

The Feret face set was used as our face training set. 739 frontal faces were used. The faces were originally 256x384 pixels. The location of the eyes, nose and mouth for each face was known and used to cut out the portion of the image that only contained the face. Each face was then resized to 24x24 pixels and normalized to be used by our algorithm. Figure 4 shows a few of these training faces. 739 non-face training images were used as well. 400 of these non-face images were artificially randomly generated using a uniform distribution and 300 were produced from random images. Each image was resized and normalized. Figure 4 shows a few of the training non-face images. The integral image for each training image was also computed and stored.

4.3 The Training Algorithm

The same general procedure for training using the AdaBoost algorithm was used as explained in Table 1. In order to make our algorithm more efficient we represented each integral image as a row vector. In this case the output of each feature applied to any image is simply the dot product of the feature vector and the integral image vector. The threshold for each weak classifier was found using the optimization function “fminsearch.m” in Matlab. We have tried to keep our program as clear as possible. Appendix B has a complete listing of our program.

Finding each best feature takes about 5 hours of computing time using a Pentium 4 with 2.4Ghz processor. This time could be reduced by using other programming languages.

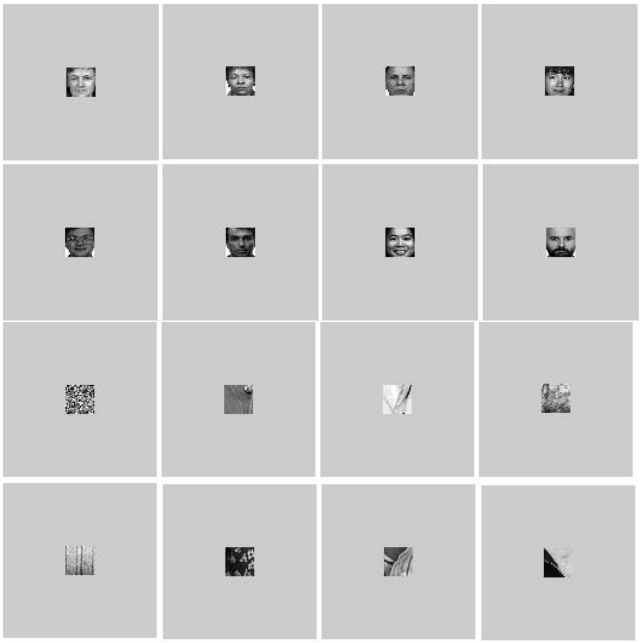

Figure 4: A few face and non-face images used in our training set.

5. Results

We allowed our program to run for approximately 50 hours and find the fist 10 best features. These features can be seen in Figure 5. It can be readily seen that these 10 best features correspond to the best features found by Viola and Jones [1]. These features also make intuitive sense as they represent the lighter and darker areas of the eyes, nose, forehead and cheeks on a typical face.

These 10 best features were then used to find probability of error plots. Figure 6 shows these plots. It is seen that the number of face images misclassified decreases as the number of features used increases. The same is true for non-face images. The Images used to produce these plots were the same training set images. This is why the probability of error is so low. Although using the same training set as a test set is not the best course of action, we are still able to verify the accuracy and correctness of our 10 best features by seeing that they are very successful at actually doing what they were trained to do and that is to classify faces over the training set. The fact that the probability of error is very low over the training set and that the probability of error decreases with increasing number of features used, shows the validity of our 10 best features.

We also gathered a small set of random images off the internet to validate our features on non-training set images. In order to do this our program scans the image at

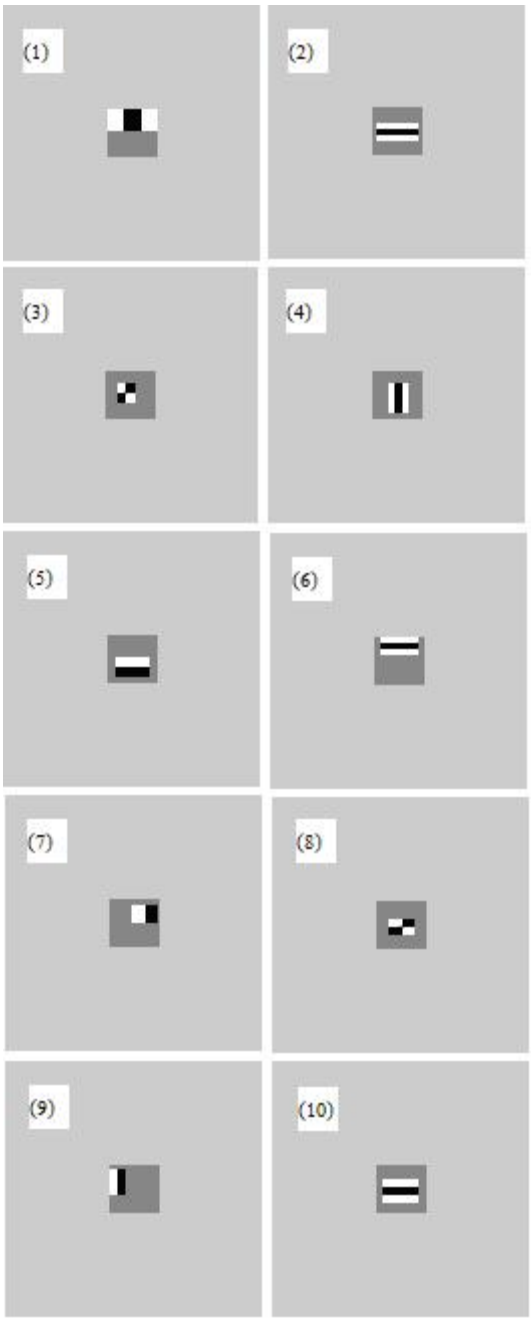

Figure 5: The 10 best features.

multiple levels and crops out sections of the image, resizes the image to 24x24 pixels, normalizes and attempts to classify the image. If it is decided that the image is a face, then a rectangular box is drawn around the original cropped out section of the image. It should be mentioned that a single face within an image will usually have multiple rectangular boxes around it, since it is classified as a face at multiple levels and by different scans. This is a phenomenon mentioned in the Viola Jones [1] paper as well.

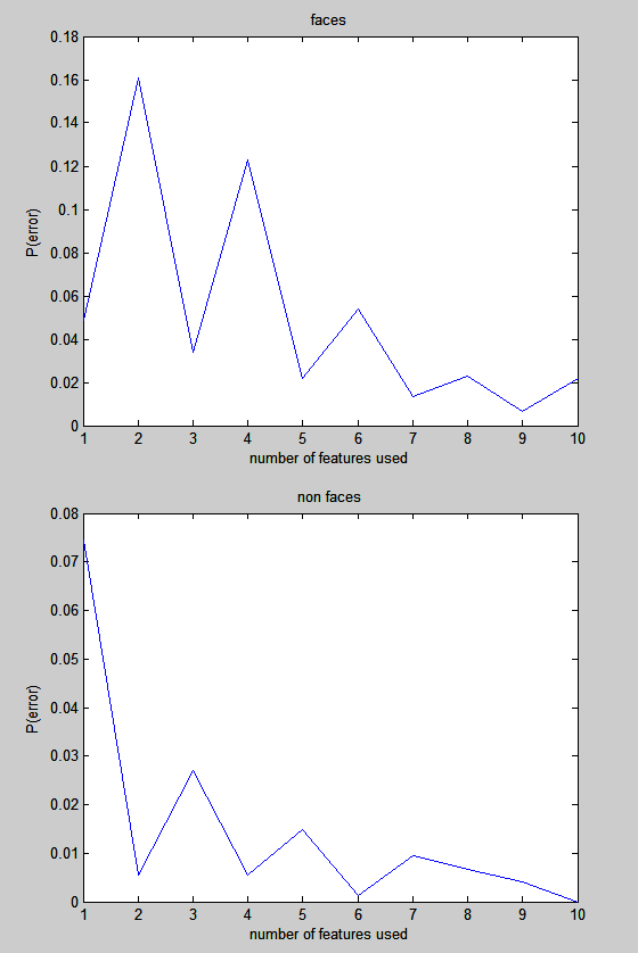

Figure 6: Shows the probability of error decreasing for increasing number of features used. Horizontal axis is the "Number of Features Used" the vertical axis is P(error) for Face and Non-Face Images.

A few of these faces can be seen in Figure 7. Appendix A shows more of these results. In order to limit the number of false positives within these images a lower bound on the size of the cropped out images was imposed on our detection program. This was done to compensate for the small number of 10 features used compared to 200 features used by Viola and Jones [1]. If this had not been done then the program would draw multiple small rectangles around portions of the image that did not include faces. Assuming that the faces within the image are relatively larger than 24x24 pixels and that their size is approximately known, our 10 best features can be successfully used.

6. Conclusion

We were able to successfully replicate the results found by Viola and Jones [1] and even arrive at the same feature set

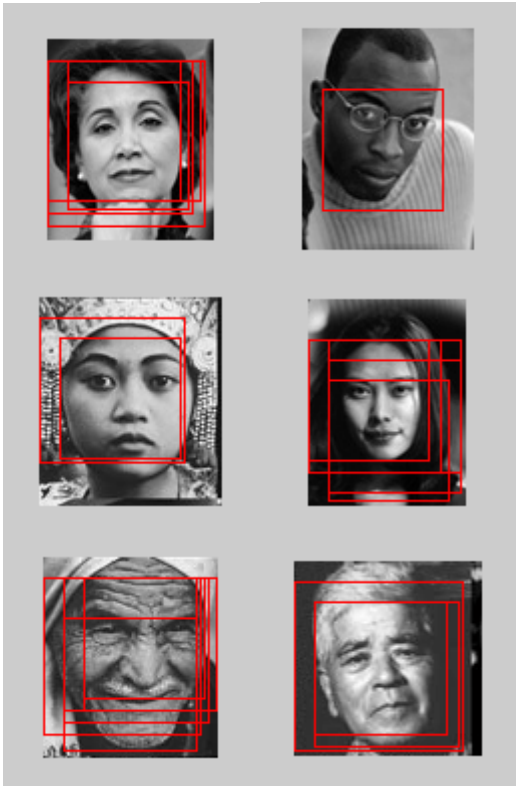

Figure 7: successfully detected faces.

even though our training set was different. This further shows the robustness of the algorithm. Although training takes much time, the detection algorithm is fast and can be used to scan large images quickly. Using 10 best features is not sufficient for detecting faces with low false positive rates in the general case, but on the other hand a small set of 10 features can be used successfully if the faces within the image are relatively large and an approximate bound can be placed on their size.

7. Acknowledgements

We would like to thank Ian Fasel for providing the training data set and the rectangular feature bank function.

References

[1] Viola and Jones,2001. *Robust Real Time Object Detection.* Proceedings, 2nd International Workshop on Statistical and Computational Theories of Vision.

[2] Freund and Schapire,1999. *A Short Introduction to Boosting.* Journal of Japanese Society for Artificial Intelligence.

[3] Ian Fasel and Schapire and J.R Movellan. 2004. *A Generative Framework for Boosting with Applications to Real Time Eye Coding.* Computer Vision and Image understanding,2004.

[4] Viola and Jones and Snow. 2001. *Detecting Pedestrians Using a Boosted Cascade of Simple Features.*In Proceedings IEEE Conf. on Computer Vision and Pattern Recognition.

Appendices

A. Examples