# Computer Vision – Project 5

## Adaboost: Face Detection

Devin Villegas

Matthew Farmer

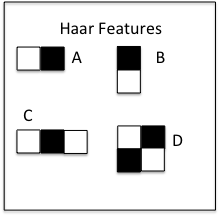
# Introduction

In 2001, facial detection software took a huge leap forward with the publication of Viola and Jones object detection framework [1], representing the first real time facial detection software. They combined Haar Basis Functions, the Adaboost machine learning algorithm from Freund and Schapire [2], and the idea of an Integral Image to achieve the real-time classification. In this report we show our implementation of this framework as well as some independent analysis of the Haar Features themselves.

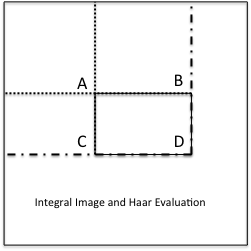
# Theory

The V&J framework begins with object detection within an image. The idea is that given a test image can we tell if, and where, the image contains an object we’re interested in. In our case, that represents a face. We need the framework to be fast, so we’ll want to operate on a more abstracted construct than raw greyscale pixel values. V&J tackled this with Haar Features and the Integral Image.

Haar Features are an extension to the Haar Basis Functions, representing an evaluation of the image. For any given feature being evaluated on an image we sum the values of pixels in the area of the black rectangle, sum the value of the pixels in the white area, and then subtract the sum of pixels in the black area from the sum of pixels in the white area.



Viola and Jones used the idea of Integral Images to evaluate these features in constant time. An integral image is one in which the value at a given pixel location is the sum total of all pixel values above and to the left of that location.



The value of the Integral Image at position A will represent the total sum of all pixels contained in the rectangle A. Using this, if we wanted to calculate the sum of pixels in the rectangle ABDC, then it boils down to simple algebra. Treating values at A, B, C, and D as their Integral Image value, then rectangle ABDC value = D + A – B – C.

This was a great technique in 2001 when the paper was released, but some informal testing in MATLAB showed this was slower than fully vectorized evaluations of the images and features, so we didn’t use this technique. Additionally, V&J had to skip the first row of pixels due to the evaluation method of the Haar Features, but we did not since we calculate sums directly. This produced a larger set of features for us to work with.

The Adaboost machine learning algorithm is pretty straight forward as well. By training a series of weak binary classifiers, the algorithm can produce a strong classifier. V&J used the Haar Features as a large set of weak classifiers to feed into the Adaboost algorithm to create a cascade detector.

# Our Process

## Reading Images

Training and test images were pulled from the CBCL MIT dataset [4]. This data set uses 19x19 images, instead of the 24x24 that V&J used. However, our images have cropped faces, so we don’t need to crop any of the face or non-face training data. We vectorize each of the 2,429 faces and 4,548 non-faces, concatenating them into a large matrix of size 6977 x 361. In addition, we created another of these matrices by whitening all of the training images.

## Creating Haar Features

One observation we made when creating each of the Haar Features was that the ‘reverse’ of the feature produces a different value. Namely, switching the black and white areas of a Haar Feature will change the sign of the valuation of it (ex. 20 would become -20). Because of this, we processed all of our Haar Features twice so we could compare the different errors. To achieve an exauhstive set of Haar Features for each type, we initially calculate the features in a 20x20 matrix and then delete the last row and column. The algorithm we use to create the features doesn’t populate the last row and column anyway, but there’s a function in our source we used to confirm the last rows and cols were all empty before proceeding.

Using the entire 19x19 space to create each of our A-D type of features produced the following amounts. After creation, we vectorized and concatenated each of these into one large 361x53,130 matrix.

|  |  |  |  |
| --- | --- | --- | --- |
| Type A | Type B | Type C | Type D |
| 1-17,100 (17,100) | 17,101-34,200 (34,200) | 34,201-45,030  (10,830) | 45,031-53,130 (8,100) |

## Evaluation of Features and Error Calculation

By multiplying the 6977x361 with the 361x53,130 matrices, we are left with a 6,977x53,130 matrix where the i,j entry represents the jth Haar Feature evaluated on the ith image, which we now call the Haar value.

Once we have this massive matrix of Haar values then we need to go about setting an error minimizing threshold with which to classify whether the image evaluated was a face or not. Initially, we did this with an iterative algorithm – first to get a baseline of correctness because it was very easy to confirm that the calculation was correct, but also to see just how much slower the iterative algorithm was than other methods. One complete pass of the iterative mechanism took about 5 hours on a high end Mac Server (16gb ram, 8 cores).

We also calculated the minimum error threshold in the style of the original V&J paper, where the threshold chosen has both a value and a polarity, which represents whether we consider everything less than the threshold to be a face or everything greater than the threshold is a face. This evaluation was completely vectorized – able to produce one complete pass of minimum error thresholds in 94 seconds on the same machine. Using the hint emailed to the class on Saturday, we also calculated the error in a third way, which dropped the polarity. This mechanism runs on all 8 cores and was able to produce complete results from one pass on our data set in 159 seconds.

## Adaboost

In the V&J Adaboosted classifier some of the later, and more complicated, stages would use a combination of features evaluated through a voting consensus. Specifically, you could associate a normalized weight with each feature in the classifier, and if 50% of the weight voted in favor of the test image being a face then it would pass that stage. For simplicity sake, we did one feature per stage. The process was as follows:

* Extract the minimum total error classifier
* Add that to the current stage
* Re-weight all of the images based on whether they were classified correctly or not
* Repeat

We attempted to create a 200-stage classifier. The problem with this process

came up in the re-weighting step. We assign the weight, alpha, with the formula (1/2)\*Ln((1-et)/et) where et represents the error of this extracted classifier. This error is calculated from the weights assigned to images, using the formula in the project description. What ended up happening is after the 78th stage, the error was effectively zero, causing alpha to blow up. This broke our algorithm, so the 79th stage onward was useless. We did not test the Adaboosted classifier.

## Analysis of Haar Features

We analyzed both polarity features using the V&J method and DrA method (from the hint) by sorting out the best overall error, recognition of faces, as well as rejection of non-faces. Here are our results.



Overall, we can clearly see that some features are better than others at certain types of classification. For example, the best scoring non-face classifiers for V&J were often the four-rectangle features where as not a single one of these features made it into the top 200 overall for either method of error calculation.

## Effect of Scale

When we rescaled the features and some testing images, we found that the detection of true faces wasn’t affected greatly. However, the rejection of non-faces suffered as well as detecting faces in non-face test images over 50% of the time.

## Conclusion

Overall, this project has a lot more room for systemic analysis and work. In the analysis of Haar Features, we often saw similar features scoring highly. If we were to evaluate these features in different stages of a classifier, they would not help reduce overall error. One intelligent way to do this is to perform a maximal area suppression on features that operate in the same neighborhood. In pursuit of a quality face detector, the majority of images analyzed in the wild are non-faces. Building a ‘custom’ classifier where we choose the first few stages as different types of features that are very good at rejecting non-faces, then build complexity in the later stages by selecting a set of features that are optimized for accepting faces would be ideal.

Another consideration is to expand our Haar Feature set. There have been quite a few interesting papers published recently where the mechanism for evaluating features has been revamped to perform quicker, and the number of features chosen has been increased by a large amount. This represents features pertaining to pupils or other relevant nuances of faces. There are also some interesting works with features that pertain to tilted faces.

The source code has been emailed along with data analysis tables. The scope of the project left us with over 30 Matlab scripts and functions, and printing all of that out would be hard to navigate.

References:

[1] Viola, Paul, and Michael Jones. "Rapid object detection using a boosted cascade of simple features." *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE, 2001.

[2] Freund, Yoav, and Robert E. Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting." *Journal of computer and system sciences* 55.1 (1997): 119-139.

[3] Crow, Franklin C. "Summed-area tables for texture mapping." *ACM SIGGRAPH Computer Graphics*. Vol. 18. No. 3. ACM, 1984.

[4] CBCL MIT dataset: http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html