Study of Viola-Jones Real Time Face Detector

Kaiqi Cen
cenkaiqi@gmail.com

Abstract

Face detection has been one of the most studied topics in computer vision literature. Given an arbitrary image, the goal of face detection is to determine whether or not there are any faces in the image and, if present, return the image location and extent of each face. While this appears to be a trivial task for human beings, it is a very challenging task for computers. The difficulty associated with face detection can be attributed to many variations in scale, location, view point, illumination, occlusions, etc. Although there have been hundreds of reported approaches to face detection, if one were asked to name a single face detection algorithm that has the most impact in recent decades, it will most likely be the Viola and Jones face detection, which is capable of processing images extremely rapidly and achieve high detection rates. This project is going to study and understand the Viola-Jones algorithm by implementing the whole detection framework and based on the implementation, conduct experiment to hopefully further improve the performance.

1. Introduction

This report is going to cover the details of implementing the 3 key components of the Viola-Jones detection algorithm: first is the introduction of a new image representation called "Integral Image" which allows the features used by our detector to be computed very quickly; the second is a learning algorithm, based on AdaBoost, which selects a small number of critical visual features from a larger set and yields extremely efficient classifiers; the last core component is a method for combining increasingly more complex classifiers in a "cascade" which allows background regions of the image to be quickly discarded while spending more computation on promising face-like regions. The report then analyzes the performance limitation of our own implementation and introduce an experimental solution by utilizing HOG features with SVM classifier. The Experiment Setup & Result shows the actual results of running and testing our own classifiers on the standard testing set.

2. Technical Content

The Viola-Jones face detector contains three main ideas that make it possible to build a successful face detector that can run in real time: the image integral, classifier learning with AdaBoost, and the attentional cascade structure.

2.1. Image integral and feature extraction

The first step of the Viola-Jones face detection algorithm is to turn the input image into an integral image. Integral image, also known as a summed area table, is an algorithm for quickly and efficiently computing the sum of values in a rectangle subset of a pixel grid. The integral image at location x, y contains the sum of the pixels above and to the left of x, y, inclusive:

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

where $i(x, y)$ is the pixel value of the original image and

1	1	1
1	1	1
1	1	1

Input image

1	2	3
2	4	6
3	6	9

Integral image

Figure 1. Example of integral image [2]

$ii(x', y')$ is the corresponding image integral value. Using the integral image to compute the sum of any rectangular area is extremely efficient, as shown in Figure 2. The sum of pixels in rectangle ABCD can be calculated with only four values from integral image:

$$\sum_{(x,y) \in ABCD} i(x, y) = ii(D) + ii(A) - ii(B) - ii(C).$$

The base resolution of a sub-window in our implementation is 19 by 19 pixels. The main reason is that the data set we are going to use are all preprocessed to be 19 by 19 pixel face or non-face images so that we don't have to spend extra time on data preparation. Also the 19 by 19 pixels resolution is relatively close to the 24 by 24 pixels sub-window

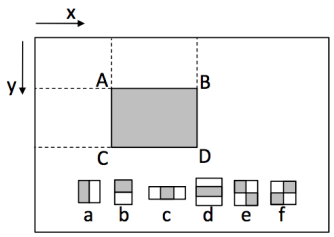

Figure 2. Illustration of the integral image and 6 types of Haar-like rectangle features

used in [5]. In our implementation, for each image sample, we extract two types of Haar-like features: the vertical feature and the horizontal feature (type (a) and (b) in figure 2). Given a image input which is 19 by 19 pixels, we compute all possible Haar-like vertical and horizontal features. Therefore the total number of features we extract from one image input is:

$$(18 + 16 + ... + 2) * (19 + 18 + ... + 1) * 2 = 34200$$

These features can be efficiently computed by first computing the integral image I. Note the integral image I calculated is 20 by 20 pixels as I starts with a row and column of zeros. As a Haar-like feature value is calculated by the pixel sum of the darker rectangle minus the pixel sum of the lighter rectangle, each Haar-like feature can then be computed as the sum of 6 values from I:

$$f = -I(x_1, y_1) + I(x_2, y_2) + 2I(x_3, y_3) - 2I(x_4, y_4) - I(x_5, y_5) + I(x_6, y_6).$$

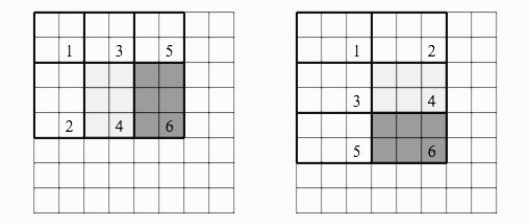

Figure 3. Computing two-rectangle Haar-like features

The `extractFeature` function implements all the feature extraction logic. It takes in `imgNum` number of images, computes all possible vertical and horizontal Haar-like features, which is 34200, and save them to a 34200×imgNum matrix. The `groupFeature` function later divides the matrix into several 5000×imgNum smaller matrices which are used in training phase and make debugging easier.

2.2. AdaBoost Learning

Given a feature set and a training set of positive and negative images, any number of machine learning approaches could be used to learn a classification function. The Viola-Jones uses a variant of AdaBoost to both select a small set of features and train the classifier. A single AdaBoost classifier consists of a weighted sum of many weak classifiers, where each weak classifier is a threshold on a single Haar-like rectangular feature. The weight associated with a given sample is adjusted based on whether or not the weak classifier correctly classifies the sample.A single weak classifier is defined as:

$$h(x, f, p, \theta) = \begin{cases} 1 & pf(x) < p\theta \\ 0 & \text{otherwise} \end{cases},$$

where f denotes the feature value, θ is the threshold and p is the polarity indicating the direction of the inequality.

In our implementation, the AdaBoost learning procedure is as follows:

1. Given training sample images $(x_1, y_1), ..., (x_n, y_n)$, where $y_i = 0, 1$ for negative and positive examples respectively.

2. Initialize the classifier count $t = 0$ and the sample weights $w_i = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0, 1$ respectively, where m and l are the number of negative and positive samples.

3. While the number of negative samples rejected is less 50%:

   (a) Increment $t = t + 1$.

   (b) Normalize the weights $w_i = \frac{w_i}{\sum_j w_j}$.

   (c) Select the best weak classifier with respect to the weighted error

   $$\epsilon_t = \min_{f,p,\theta} \sum_i w_i |h(x_i, f, p, \theta) - y_i|.$$

   (d) Define $h_t(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t and θ_t are the minimizers of ϵ_t.

   (e) Update the weights as

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

   where $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$ and $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise.

   (f) Compute the strong classifier

   $$H(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \gamma_t \\ 0 & \text{otherwise} \end{cases},$$

   where $\alpha_t = \log \frac{1}{\beta_t}$ and γ_t is chosen such that all positive training samples are correctly classified.

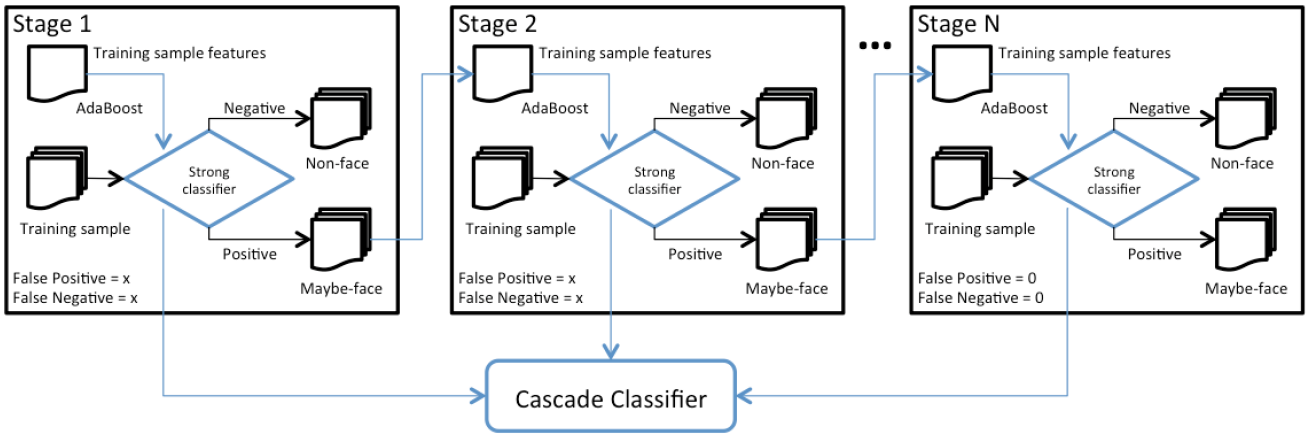

Figure 4. Cascade work flow

(g) Evaluate negative samples by the newly computed strong classifier H and update the number of rejected negative samples

To calculate the minimum error in step (c), we have to search over every possible feature for every single training sample. However, for a given feature, we can pass through a sorted list of the training images once to find the optimal θ. This can be accomplished by maintaining four sums: the total sum of positive sample weights T^+, the total sum of negative sample weights T^-, the sum of positive sample weights below the current sample w^+, and the sum of negative weights below the current sample w^-. The error for a threshold which splits the range between the current and previous sample in the sorted list can be computed as

$$e = \min(w^+ + (T^- - w^-), w^- + (T^+ - w^+)).$$

Note that the first error in the min function is the error associated with labeling all samples below the current sample negative and labeling the samples above positive. In this case, the polarity of the weak classifier should be $p = -1$.

2.3. Cascade Classifier

The cascaded classifier is composed of stages each containing a strong classifier from AdaBoost. The job of each stage is to determine whether a given sub-window is definitely not a face or maybe a face. When a sub-window is classified to be a non-face by a given stage it is immediately discarded. Conversely a sub-window classified as a maybe-face is passed on to the next stage in the cascade. It follows that the more stages a given sub-window passes, the higher the chance the sub-window contains a face.

The `AdaCascade` script contains the implementation of AdaBoost learning under Cascade structure. In our implementation the first stage trains a strong classifier by the AdaBoost procedure explained in section 2.2. with features from all training samples. Then we use the strong classifier to classify training samples and calculate false positive and false negative counts for this stage. The second stage will train a strong classifier using only the samples classified as positive by the first stage. Use the strong classifier to classify the remaining samples and calculate the false positive and false negative count for this stage. Repeat the stages until one stage achieves zero false positive and false negative. All strong classifiers threshold in each stage are saved to form the final Cascade classifier. Figure 4 has the illustration of the Cascade process.

2.4. HOG with SVM Alternative

Our implementation of the Viola-Jones face detector yields good performance with frontal faces from the MIT face database. However, when testing with the subset of CMU test set, the detection rate drops significantly. One of the major possible reason is that unlike the training set which mainly are frontal faces, the CMT test set contains more multiview variations and illumination changes. According to [5] and [6], the original Haar-like feature set has limitation for multi-view face dection and lack robustness in handling faces under extreme lighting conditions, despite that the Haar features are usually normalized by the test window's intensity covariance.

In this case, Histogram of Orientated Gradient(HOG) becomes a great alternative feature option as it is largely invariant to global illumination changes and is capable of capturing geometric properties of faces that are difficult to capture with linear edge filters such as Haar-like features. Unlike the Haar-like features, the HOG feature space is relatively small for a 19 by 19 images. Therefore, the idea to improve detection rate is to quickly extract HOG features and train a very simple SVM classifier and combine it with the Cascade classifier. Function `extractHOGFeature` computes the HOG features in a similar manner to what

we did in problem set 3: it calls `computeHOGFeature` function to compute the gradient and magnitude and create histogram of gradient orientations; All block features are bi-linearly interpolated and binned to N orientations and finally being concatenated together. After tuning the parameters, cell size of 6 pixels, block size of 2 cells with 9 bins yield the best performance For a 19 by 19 pixels window. Finally, the simple SVM classifier is trained by using `vl_svmtrain()` of the `vl_feat` library. The `testSVM` script contains the code for training and testing simple SVM classifier on the testing data set. The Cascade classifier and SVM classifier are combined as

$$H(x) = \begin{cases} 1 & C(x) = 1 \text{ or } S(x) = 1 \\ 0 & \text{otherwise} \end{cases},$$

where C is the Cascade classifier, S is the SVM classifier and H is the final classifier.

3. Experiment Setup and Results

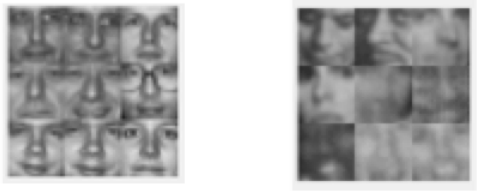

Figure 5. training images (left) v.s. testing images (right)

The data set used for training and testing comes from MIT CBCL Face Database.Click here for accessing the data set. The training set face were generated for [4]. The training set non-faces were generated for [1]. The test set is a subset of the CMU Test Set 1 [3], information about how the subset was chosen can be found in [1].All samples are 19 by 19 pixels Grayscale PGM format images. The training set has 2,429 faces and 4,528 non-faces. The test has 472 faces and 23,573 non-faces. All the source code is written in Matlab and the only external library we used is `vl_feat` library.

The final Cascade classifier is trained on 2000 faces and 2000 non-faces from the training set and the final Cascade classifier has 9 stages. When testing on 429 remaining faces and other 1000 non-faces from the training set, the detector successfully detects **425** faces and mis-classifies **0** non-faces. The false negative rate is below **1%** and the false negative rate is **zero**. The result is shown at Figure 6.

However, when testing on the CMU Teset Set with 472 faces and 1000 non-faces, **17** non-faces are classified as faces, which yields a pretty low false positive rate of **1.7%**,

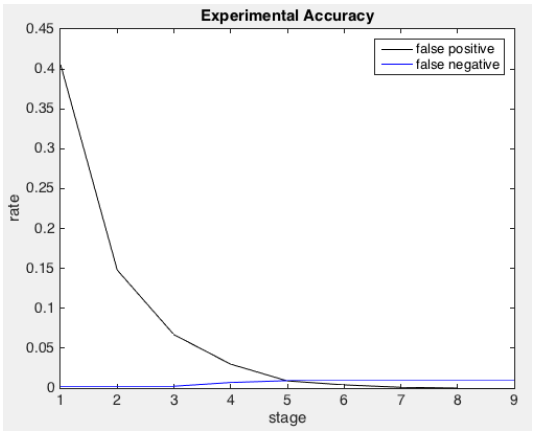

Figure 6. Result of testing on training data

but the detector fails to detect **275** faces which indicates a false negative rate of **58.4%** and **41.6%** detection rate. The result is shown at Figure 7.

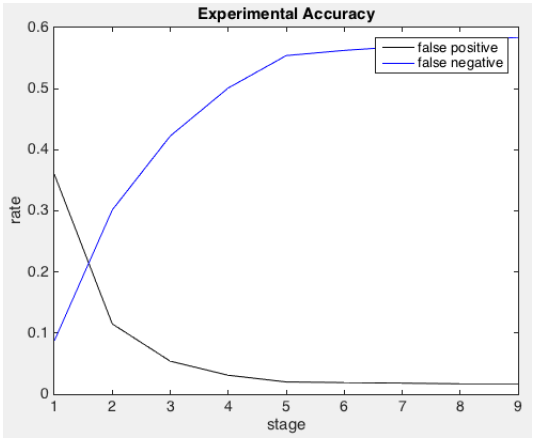

Figure 7. Result of testing on subset of CMU Test set

For the HOG + SVM classifier, after tuning the HOG hyper parameters, the best performance achieved on the same 471 faces and 1000 non-faces data sets is **37%-42%** detection rate and **2.9%-3.5%** false positive rate, with $cellsize = 6, blocksize = 2$ (12 by 12 pixels) and $binnumber = 9$.

When testing the final classifier by combining the cascade classifier and SVM classifier on the same 471 faces and 1000 non-faces data sets, the detection rate increases from **41.6%** to **62.8%** (**296** faces) while the false positive rate also increases from **1.7%** to **5.9%** (**59** non-faces).As shown in Figure 8, the final results, the combination of AdaBoost + Cascade and HOG + SVM successfully improves the performance by a **20%** increase in detection rate but also with a trade off of a **5%** increase in false positive rate, which is relatively acceptable. The `testClassifier` script contains the code for testing all the classifiers.

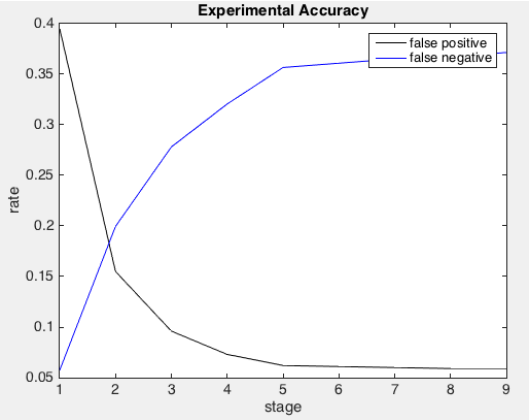

Figure 8. Result of testing combined classifier on CMU Test set

4. Conclusion

The purpose of this project was to implement the Viola-Jones face detection algorithm and obtain reasonable performance. The results we tested meet our expectations and the experimental solution proposed effectively improves the performance of the final implementation. In terms of the processing time of our final detector, surprisingly the time bottle neck was not at the AdaBoost learning phase but the Haar-like rectangular feature extraction phase. It took me over 2 hours to extract 34200 features for 2000 training samples while the total learning time on 2000 positive samples and 2000 negative samples is within 1 hour. This fact indicates that firstly the attentional cascade applied in Viola-Jones algorithm indeed effectively speeds up the training process. Secondly Haar-like feature space is actually pretty huge and time consuming to compute. It is also likely that our code for feature extraction can be re-factored in a more time-efficient manner.

Besides, there are many other aspects that can be improved. First of all, data preparation seems to be a major source of performance limitation. Given more image data sets with more ideal image size (around 24 by 24 pixels), we are able to train a more solid classifier. With subwindow larger than 19 by 19 pixels a lot more feature values can be extracted for both Cascade and SVM classifier and thus more details of the face can be well captured. Secondly, in terms of feature extraction, our implementation only extracts 2 types of Haar-like features. To improve performance, many more types of Haar-like features actually can be applied for training. Especially, the type c feature in figure 2 seems to be a representative feature as nose occupies significant part of a 19 by 19 pixel face. The SVM classifier can also be incorporated into the cascade structure to speed up if learning time scales up with increasing data samples.

The source code of the project can be found at `https://github.com/JackCen/CS231A`

References

[1] B. Heisele, T. Poggio, and M. Pontil. Face detection in still gray images. A.I. memo 1687, Center for Biological and Computational Learning, MIT, Cambridge, MA, 2000.

[2] O. H. Jensen. *Implementing the Viola-Jones face detection algorithm*. PhD thesis, Technical University of Denmark, DTU, DK-2800 Kgs. Lyngby, Denmark, 2008.

[3] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[4] K.-K. Sung. *Learning and Example Selection for Object and Pattern Recognition*. PhD thesis, MIT, Artificial Intelligence Laboratory and Center for Biological and Computational Learning, Cambridge, MA, 1996.

[5] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[6] C. Zhang and Z. Zhang. A survey of recent advances in face detection, 2010.