Viola and Jones Face Detection Implementation

Iyuehan Yang, Daniyal Usman

*COSC 4P76: Machine Learning, Computer Science Department, Brock University*

*1812 Sir Isaac Brock Way, St-Catharines, ON L2S 3A1, CA*

[Iy12qm@brocku.ca](mailto:Iy12qm@brocku.ca), du11ph@brocku.ca

***Abstract -* The goal of this work is to implement real time face detection using the Viola and Jones algorithm. In order to accomplish this task, this paper describes a machine learning approach for object detection which is capable of processing images extremely fast. The key components which allows fast processing speed is implemented according to the Viola and Jones standards. The first of which is using the sliding window technique to split the full image into smaller sub-windows. The integral image is then calculated from these sub-windows in order to accomplish Haar feature detection in constant time. Finally, the system is trained using AdaBoost which provides us with a linear combination of weighted learners. This is then used to cascade the object detection into several stages in order to eliminate all the non-faces sub-windows as early as possible. The program, implemented in Java 8 with Eclipse IDE will be trained using approximately 2000 positive samples and 20000 negative samples; 5000 set of different negative examples per cascade - all of which will be normalized into a 24 by 24 grayscale image. The final result procudes 4 cascading stages.**

***Keywords -* Machine Learning, Viola and Jones, Haar features, Integral Image, AdaBoost, Cascade classification**

I. Introduction

This document is a report for my term project for the course COSC 4P76: Machine Learning. To perform this experiment, the system will be trained using the data set found at University of Washington, Computer science department for the initial cascade (stage) and random non-face images for the remaining cascades. For the initial cascade, each image is scaled to and normalized to a 24 by 24 grayscale image. For the remaining cascade stages, images are converted to a grayscale and each sub-window is scaled to a 24 by 24. The positive samples used are all full frontal upright images while the negative samples include background images and random objects.

…

The problem definition is to implement Viola and Jones face detection using different scaling and cascading parameters and compare their effects on the system. The goal of this work is to correctly classify each sub-window as either containing a face or not a face.

II. Sliding Window

The sliding window technique is a simple alignment solution widely used in object detection. The size used in our project is a 24 by 24 pixel window which is scanned throughout the image. The key point about sliding window is that each window to overlap from the previous window allowing the set of sub-windows to contain all the possible permutations of the image. Each window is then separately classified or trained on.

III. Haar Features

Haar features are digital image features used in object recognition. Originally, working with image intensities – the RGB pixel values of each pixel – was an extremely computationally expensive task. Therefore, a publication by Papageorgiou et al. discussed working with an alternate feature detection set based on Haar wavelets. Viola and Jones adopted the idea of these Haar wavelets and created the Haar-like features. The general idea of a Haar feature is the difference between the sums of the pixel intensities in 2 separate regions as shown in Fig 3.1.



Fig. 3.1 - Type 1 Haar Feature Example

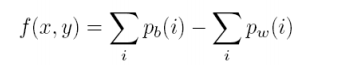
The value calculated from the difference is then used to categorize subsections of an image. The formula used to calculate values of Haar Features is as follows.  


Fig. 3.2 - Haar Feature Value Formula

There are a total of 5 types of Haar features used in this algorithm as shown in Fig 3.3.

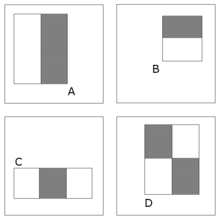


Fig. 3.3 – 4 Types of Haar Features used

In the case of face detection, it is a common observation that the top of the cheeks is whiter in comparison to the region of the eyes. Thus using Haar features, we cover the portion of the eyes with a black rectangle and the cheeks with a white rectangle and calculate the sums of pixel intensities of each region. This value, along with its coordinates, type and size is what is known as a Haar feature. The position of these rectangles is defined relative to the window acting as a bounding box to the target object.

Haar features are considered as weak features and thus require a tremendous amount of them in order to properly classify objects. In this program, each of the 5 different types of features has to be calculated at every location of the sub-window in every possible size. Since our windows are of size 24 by 24, the exhaustive set of features needed to be calculated exceeds 160 000. This is considered extremely computational heavy. However, by using the idea of Integral images, the Haar feature calculations can all be done in constant time which greatly reduces the time needed to calculate all the features for each window.

IV. Integral Images

The idea of Integral images is converting an image into a summed area table and using that table in order to quickly calculate the values rectangular regions of the Haar features. The formula used to calculate this conversion is shown in Fig 4.1.

https://computersciencesource.files.wordpress.com/2011/05/sxyeq.png

Fig. 4.1 – Conversion formula to Integral Image

The integral image is a quick way of calculating the sums of the pixel values in a given window of pixels which is then used to calculating the average intensity in a given window.

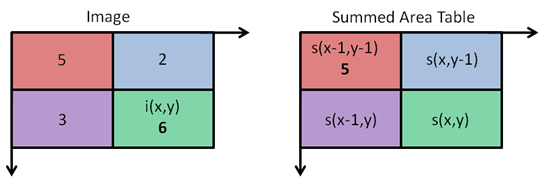


Fig. 4.2 –Integral Image Calculation example

Once the summed area table is filled, calculating the sum of any rectangle region of the Haar feature can be done in 4 array references. As shown in Fig 4.3.

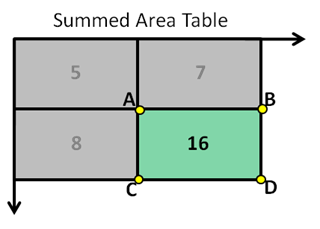


Fig. 4.3 – Calculating are of the green region is done in 4 Array references. The formula is simply D+A-B-C

Since Haar features have regions which involve adjacent rectangular sums and each rectangular region can be calculated in 4 array references, then a 2-rectangle Haar feature can be computed in 6 array references, 3-rectangle Haar features in 8 and 3-rectangle features in 9. Using this technique, computing 160 000 Haar features can be accomplished in an extremely fast time.

V. AdaBoost

Given a feature set and a large number of positive and negative training examples, any machine learning approaches can be used to learn a classification function. In this case, AdaBoost is used in order to train the system using the training data, as well as building a cascading style of classification.

AdaBoost learning is described as boosting relatively weak learners from other learning algorithms and combining these weak learners into a weighted sum that represents the final output of the boosted classifier. The idea is that by combining many weaker learners, as long as they are slightly better than random guessing, then the final model will converge into a strong learner.

In the Viola Jones model, there are over 160 000 weak features associated with each sub-window. If this exhaustive set has to be computed every time during real time detection, the complexity is expensive. The idea behind Viola and Jones theory is that a smaller sub set of these 160 000 features can be combined together into a stronger learner and the main issue is to find these features. The algorithm designed selects the Haar feature which best separates the positive and negative examples.

VI. Cascading Classifiers

The goal of cascading classifiers is to reduce the time it takes to iterate through each sub-window. The idea behind cascading classifiers can be explained through a simple decision tree. See Fig 5.1

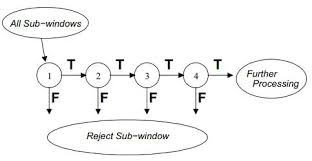


Fig. 5.1 – Cascading classifiers Decision making

For each of the sub-windows in an image, it goes through each stage in the cascading classifier iteratively. Each stage in the cascading classifiers are constructed by training in AdaBoost and then adjusting the threshold to minimize false negatives. When one feature iterates through each stage in the cascade, it has to pass a certain threshold in order to move on to the next stage in the cascade. Generally, the first few stages are designed to have a stronger classifier by reducing the threshold to minimize false negatives. This way, most of the sub-windows passing through the first stage will be immediately rejected.

The structure of the cascade tells us that within a single image, most of the sub-windows are negative samples. Therefore, the cascade attempts to eliminate as many negative samples as early as possible. Only on rare occasions would a positive sample trigger.

The final strong classifier is a linear combination of weak classifiers multiplied by their individual weights as shown in Fig 5.2.

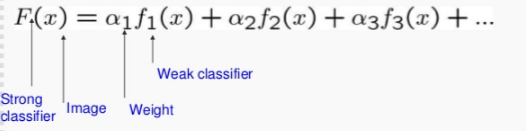


Fig. 5.2 – Strong Learner = Linear Combination of Weak weighted weak learners

In order for the cascade to be optimal, the system would need a high detection rate along with a low false positive rate. To achieve this, a lot of features would be required for each classifier which results in a longer time to compute. In principle, one could trade off the number of classification stages, the number of features in each stage and the threshold of each stage in order to minimize the total number of evaluated features. Finding this optimum is the problem.

The algorithm used to find this optimum goes as follows. Each stage in the cascade will reduce the false positive rates and detection rate. You would set a target for the minimum reduction in false positive rates and the maximum reduction in detection rates. Then for each stage, features are added until the target detection and false positive rates are met.

VII. Implementation

Resources – The results were conducted on an MSI Laptop containing 4GB of RAM and a Intel® Core™ i5-2450M CPU @ 2.50GHz.

Training – the training was the most intensive part of this project. During training, using 2000 positive examples and 5000 negative examples, it takes approximately 15 minutes to find only one feature out of approximately 160000 possible features. An example consists 24 by 24 of an image. The time required for training increases by a significant amount as more training examples are included. Generally, each cascade stage includes even more features than the one before. If we were to compare this to the original Viola Jones implementation, it included approximately 6000 features in their final classifier and was trained on 350 million negatives examples. If we were to find 6000 features with our resources using only our training data, it would take at-least 62 days. Taking this fact, our limited resources and our time constraints into consideration, we used a different approach, hoping that it would be effective.

Instead of training our algorithm to convergence; minimize false positive rate. We used a set amount of cascade stages with set amounts of features. We concluded with using 4 cascade stages with 2,4,8,16 features for each stage respectively.

Cascade 0 – This is the initial stage, approximately 2000 examples for the positive training and approximately 10000 examples for the positive evaluation set. The reason why the examples in the positive training set is so low is because we compared the lowest error from training with 2000 examples and the lowest error from training with 10000 examples. The difference between the two was insignificant. Therefore, we reduced the number of positive examples to reduce the time it took to train. We then focused more of our efforts towards negatives examples to reduce the number of false positives. The negative training and testing set included approximately 5000 examples each. We would’ve preferred to train it with an even greater negative set but this would have significantly increased the training time. The training concludes with a strong classifier for the initial cascade. The classifier is used on the training set and evaluation set and uses the examples that are classified as false positives for the next training set.

Cascade n+1 – Used a random negative image for training. Negatives examples are generated by false positive examples retrieved through applying the Cascade 0 strong classifier to a 24 by 24 sub-window along the entire image by a distance of one pixel. The sub-window then increases by a factor of 1.25 and repeats the previous procedure to generate even more examples. This negative set is combined with the surviving examples from the previous stage. We limited the total number of examples to be 5000. The initial set of positive examples is used for all cascade stages. The resulting new classifier repeats the same step for extracting all false positives from the current sets and the training set for the next cascade stage.



Figure 5.3 – One of the random images for cascade training after the initial step

VIII. Results and Conclusion

C# - Cascade #

PET – Positive Examples for Training

NET – Negative Examples for Training

PEE – Positive Examples for Evaluation

NEE – Negative Examples for Evaluation

FP – False Positive on Evaluation Set

NP – False Negatives on Evaluation Set

NEMT – Negative Examples Misclassified from Training

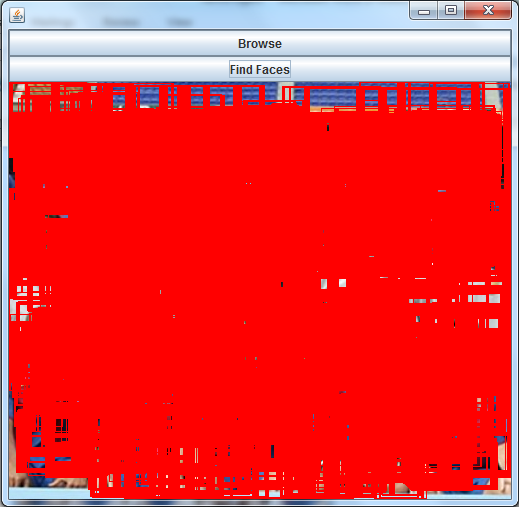
NEME – Negative Examples Misclassified from Evaluation

S – Surviving Negative Set for Training for next cascade stage

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C#** | **PET** | **NET** | **PEE** | **NEE** | **FP %** | **FN%** | **NEMT** | **NEME** | **S** |
| **0** | 2647 | 5009 | 10586 | 5011 | 0.04 | 0.27 | 208 | 212 | 420 |
| **1** | 2647 | 5001 | 10586 | 4581 | 0.12 | 0.24 | 1177 | 562 | 1739 |
| **2** | 2647 | 5001 | 10586 | 3262 | 0.1 | 0.38 | 604 | 335 | 939 |
| **3** | 2647 | 5001 | 10586 | 4062 | 0.14 | 0.17 | 771 | 598 | 771 |

The results from training didn’t go as planned. We failed to consistently reduce the false positives rate. I believe the algorithm is under-fitting because of the lack of examples. It doesn’t seem to further generalize as it progresses down the other cascades. However, when implemented as a final product, you will notice that in Fig 5.3 it does generalize to a certain degree but fall off as it progresses, Fig 5.5.

Fig 5.4 – initial sub-windows of 24 by 24 detection

Fig 5.5 – significantly larger window detection

IX. References

[2]Y. Wang, 2016. [Online]. Available: http://www.ipol.im/pub/art/2014/104/article.pdf. [Accessed: 29- Apr- 2016].

[4]O. Jensen, 2016. [Online]. Available: http://etd.dtu.dk/thesis/223656/ep08\_93.pdf. [Accessed: 29- Apr- 2016].

[3]P. Viola and M. Jones, 2016. [Online]. Available: http://www.vision.caltech.edu/html-files/EE148-2005-Spring/pprs/viola04ijcv.pdf. [Accessed: 29- Apr- 2016].