SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348-8379/ https://doi.org/10.14445/23488379/IJEEE-V10I7P113

Volume 10 Issue 7, 140-147, July 2023
© 2023 Seventh Sense Research Group®

Original Article

Implementing Face Detector using Viola-Jones Method

Ali H Alyousef

Information Technology, Saudi Aramco, KSA.

Corresponding Author : aliahy20000@hotmail.com

Received: 09 May 2023 Revised: 03 July 2023 Accepted: 21 July 2023 Published: 31 July 2023

Abstract - *This paper presents implementing a face detection algorithm based on the Viola-Jones method. The Viola-Jones method is a well-known and efficient face detection algorithm that uses Haar-like features, Adaboost, integral images, and the cascade of classifiers. The implementation in this paper was done in MATLAB and was tested using the MIT + MCU database. The results show that the detector achieves a detection rate of 60%, which is lower than the 90% detection rate of the original Viola-Jones method. However, the detector achieves a better false positive rate rejection. The design choices made in this implementation affect the trade-off between the system's accuracy and speed.*

Keywords - *Face detection, Viola-Jones, Haar-like features, Adaboost, Integral images, The cascade of classifiers.*

1. Introduction

The primary objective of this paper is to design a robust face detection system using the Viola-Jones method capable of accurately identifying and localizing human faces within images, even in unconstrained environments. The target detection rate for this system is approximately 90%, with a minimal number of false detections.

Viola and Jones' paper inspires our approach, "Rapid Object Detection using a Boosted Cascade of Simple Features", [1, 2] and incorporates Open CV's pre-trained classifiers [3, 4] for enhanced performance. The entire implementation uses MATLAB (R2012a) [5].

2. The Viola and Jones Framework

The Viola-Jones method is an object detection framework proposed by Paul Viola and Michael Jones in 2001. It comprises four fundamental techniques: Haar-like features, Integral image, AdaBoost, and the cascade of boosted classifiers. [1]

The method relies on Haar-like features, which are rectangle features used to capture image intensity differences. The features are calculated efficiently using an intermediate image representation called the Integral image. The AdaBoost machine learning algorithm is then applied to select the most relevant features and train the classifiers.

To improve the detection speed, the classifiers are organized in a cascade structure where each stage is a robust classifier consisting of a few weak classifiers. The cascade structure allows for the rapid rejection of non-face image regions in the early stages. [7, 8]

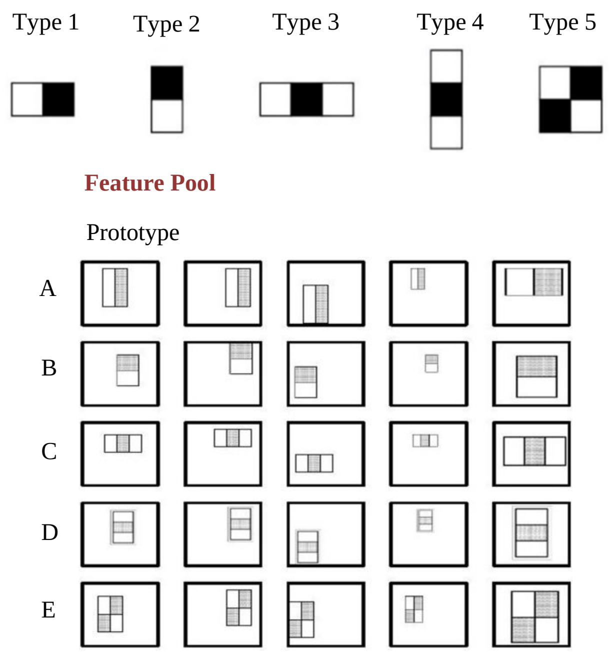

Fig. 1 Five different types of Haar-feature in different sizes [9]

Only simple rectangular features are used, which can be computed very fast. The Integral image allows for fast calculation of the rectangle feature responses. AdaBoost selects the most essential features from a large set to form a small collection of critical features. The cascade structure of increasing complexity classifiers is used to minimize computation time. [1]

The Viola-Jones method achieves competitive detection rates at high speed. It was used to develop a face detection system approximately 15 times faster than any previous approach, with over 90% detection accuracy. The key contributions of this work are introducing new image representation (Integral image), machine learning method (AdaBoost), simple rectangular features, and the cascade structure to optimize speed and performance. [6, 10]

This framework demonstrates how machine learning techniques can be leveraged to solve complex computer vision problems efficiently. [1] The Viola-Jones method is an efficient and scalable object detection framework that builds a robust classifier from a large set of weak classifiers. It has become a foundation for many applications in computer vision and beyond.

3. The Code Implementation

This section focuses on the algorithm implementation in designing the face detector. The programming language used was MATLAB (R2012a). The first section covers the initial pre-processing, including reading the image and changing its format to a standard one.

The second section includes the calculation of the integral image and standard deviation. The third section relates to scaling the image to detect different sizes of faces. The fourth section focuses on feature extraction. Moreover, the final section relates to overlapping issues and the algorithm was written to solve them.

3.1. Pre-Processing

The first step is to read the image. As there are different image formats, the second step is to convert the image into grayscale. Gray scale representation converts the image into a matrix, where each pixel corresponds to one element in the matrix. The element's value depends on how bright or dark the pixel is at that point in the image. There are two classes of gray scale: double class and unit8 class.

Many functions operate by only using double class, in which case unit8 class must be converted to double. The double class assigns a floating value between 0 and 1 to each pixel, while the unit8 assigns a value between 0 and 255 to each pixel. The value 0 represents the black pixel, and values 1 and 255 represent the white pixel. These processes are implemented in MATLAB. Note that Image Processing Toolbox is needed to use these commands.

```
%Read the input image
   Input_Image = imread('1.jpg');
%Convert the colour image to grayscale
   Input_Image =
rgb2gray(Input_Image);
%Convert the unit8 class to double
   Input_Image = double(Input_Image);
```

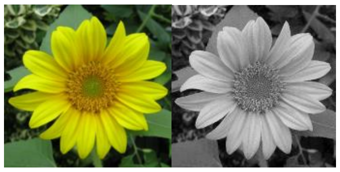
Fig. 2 Converting a colour image to grayscale

3.2. Integral Image

The Integral Image technique enhances the speed of finding the presence or absence of hundreds of Haar features at every image location. The Integral Image is calculated for every pixel in the image. The integral value is the sum of all the pixel values above it and to its left, including the original pixel value. This process starts at the top left and finishes at the bottom right. The 2012 MATLAB version has added the Integral Image command to the Computer Vision System Toolbox. However, before calculating the integral image, it is better to resize the images so that the maximum dimension equals 512 pixels. This step ensures that large images are resized before further processing. The next piece of code shows how the resize is implemented.

```
%Read the columns and rows size
   [rows,cols] = size(Input);

% the cols size will be checked first
    if (cols>512)
     cdifference=cols-512;
    else
     cdifference=0;
    end

% then the rows size will be checked
    if (rows > 512)

          rdifference=rows-512;
    else
          rdifference=0;
    end

%then we determine which ratio to take
    if (cdifference>rdifference)
            Ratio=cols/512;
    elseif (rdifference>cdifference)
              Ratio=rows/512;
    else
              Ratio=1;
    end
%we resize the image here depend on the Ratio
Input = imresize(Input, [rows cols ]/ Ratio);
```

Then the sum of the Integral Image will be calculated using this line of code. Note that the 'integralImage' command is only introduced on MATLAB R2012a.

```
RegionSumIntegral = integralImage(Input);
```

However, if the image is compressed, all the pixels will be darker than the original image. So if the original sub window contains 3 pixels, [1 0 1] and the Haar-like feature has three rectangles, each of size 1x1 pixel, and arranged as [1 -1 1], with a classifier's threshold of 1.8 when the Haar-feature activation is computed, the result will be (1*1) + (-1*0) + (1*1) = 2 > 1.8. This means the feature was detected successfully. Now consider the same sub-window, but this time with the image exposed. The feature is still there, but all the pixels are darker. The sub-window now contains a different pixel value [0.5 0 0.5]. The result of the computation is now different (5*1) + (-1*0) + (0.5*1) = 1 < 1.8, which means the feature has not been detected. The feature was there but too dim to be seen.

Now, consider the standard deviation of the pixel values to remedy this problem. Still considering the [0.5 0 0.5] sub-window, its standard deviation is 0.2887. So when you compare the activation to the threshold, the threshold must be multiplied by the standard deviation of the pixels in the sub-window to adjust to the picture's lighting. The threshold becomes, in this example, 1.8*0.2887=0.5196. So, this solves the problem because now the result is 1 > 0.5196. The feature will be detected. The line of calculating the standard deviation Integral Image is straightforward.

```
RegionSTDIntegral = integralImage(Input.^2);
```

3.3. Scaling

Faces in images can vary in size. So, the detector is implemented to scan across the image at multiple scales and locations. There are two ways in which scaling can be achieved, either by scaling the image itself or by scaling the Haar-like feature. Both methods will give the same results, but scaling the Haar feature was found to take less computational time. Each scale will be divided into sub-windows to extract the Haar feature. However, how many different scales are needed to evaluate the classifier? Let the scale update equal R. We need to shrink the initial scale to get down to scale =1. So then the equation will be:

In scale * (Rsn) = 1, solve for sn
Where inscale: is the initial scale
R = scale update
Sn = scale iteration

$$.:\mathrm{sn} = \log\frac{1}{\mathrm{inscale}} \div \log \mathrm{R}$$

Then the result needs to be rounded to the nearest integer toward infinity. So in MATLAB, it appears like this:

```
sn=ceil(log(1/inscale)/log(R));
```

The scale update (R) is set to be equal to 0.8. Therefore, each scale is larger than the previous one by a factor of 0.8. On the other hand, the initial scale is not a fixed number. It depends on the image dimension. The highest dimension of the image is divided by 20. So, for a 512x512 image, the initial scale is 25.6, giving 18 scales. The choice of values will be discussed further in the discussion.

After knowing how many scales there are, the next step is to create a loop that goes through each scale and extracts its features. As mentioned, each scale will consist of sub-windows. So to divide the scaled image into sub-windows, we will generate the set of coordinates (x(k), y(k)).

```
[x,y]=ndgrid(0:step:(IntegralImages.width-w
1),0:step: (IntegralImages.height-h-1));
```

Later, these sub-windows will be fed into the cascade classifier to perform face detection.

3.4. Extracting

In order to extract and calculate the Haar feature, we need to understand the structure of the cascaded open cv classifier. The cascade classifier has 22 stages. Each stage has many Haar-feature (trees). Each tree is a vector consisting of one row and 21 columns. Every value in the vector corresponds to something related to the feature. Each Haar feature may contain up to 3 rectangles, and five essential elements must be known for each feature. The five elements are the 'x' coordinate, 'y' coordinate, the width of the feature, height, and weight. The values from 6 to 10, 11 to 15, and 16 to 20 in the vector correspond to the first, second, and third rectangles. As you may notice, each rectangle has five values representing its shape. The following lines of code show that these values are extracted. Note that the notation 'Leaf' corresponds to Haar-feature. Also, note that the feature's elements are multiplied by the scale to scale the feature instead of the image itself. [11].

```
%- Each haar-feature may contain up to 3
rectangles
    for i_Rectangle = 1:3
%- point to the correspond rectangular values
    Rectangle = Leaf(:,(1:5)+i_Rectangle*5);
%- extract the rectangular x coordinate
    RectX = floor(Rectangle(:,1)*Scale+x);
%-  The  extract the rectangular y coordinate
    RectY = floor(Rectangle(:,2)*Scale+y);
%-  The  extract the rectangular width

    RectWidth = floor(Rectangle(:,3)*Scale);
%-  The  extract the rectangular hight
    RectHeight = floor(Rectangle(:,4)*Scale);
%-  The  extract the rectangular weight either 1
or -1.
    RectWeight = Rectangle(:,5);

end
```

The integral image is used for each rectangle in the feature to find the sum of the pixels' values inside that rectangle. However, what are the coordinates for that rectangle? The following figure 3 illustrates how the values of the four corners are obtained. Then the integral image equation mentioned in the theory section is applied to find the sum of pixel intensities for that particular rectangle in the image.

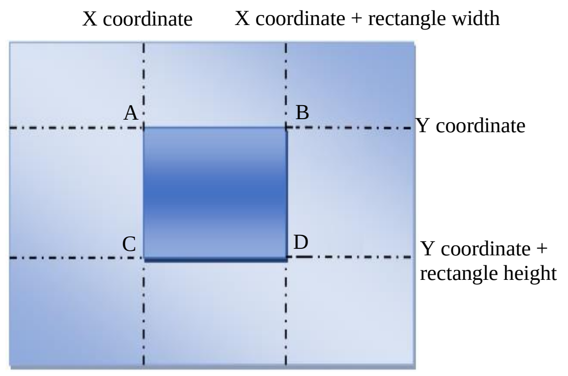
X coordinate
X coordinate + rectangle width
A
B
Y coordinate
C
D
Y coordinate + rectangle height

Fig. 3 Integral image calculation

The following lines of code show how this is implemented in MATLAB.

```
rect_sum =

IntegralImage((xcord+RectWidth)*IIwidth +ycord + RectHeight+1)

+ IntegralImage(xcord*IIwidth+ycord+1)

- IntegralImage((xcord+RectWidth)*IIwidth+ycord+1)

- IntegralImage(xcord*IIwidth+ycord+RectHeight+1)
```

After calculating the sum of pixel intensities for the rectangle, the algorithm calculates the weighted sum of pixel values by multiplying the sum of pixel values with the rectangle weight within the current rectangle. The weighted sum of pixel values for the three rectangles is added to form the feature value.

Looking back at the tree vector, the first three elements are essential. The first value in the vector is the feature (tree) threshold. The second and third values are 'right value' and 'left value'. The feature value calculated previously is set to be compared with the feature threshold. The algorithm assigns the 'right value' to the detection result if the feature value exceeds the threshold. Otherwise, the 'left value' is assigned. Finally, the detection results will be added to form the stage value at the end of each stage. This stage value will be compared with the stage threshold. If the result passes the stage threshold, the sub-windows will get through the next cascade stage. Otherwise, it continues to the following scale. On the next page is a summary diagram (Figure 4) of the structure of the open cv classifier.

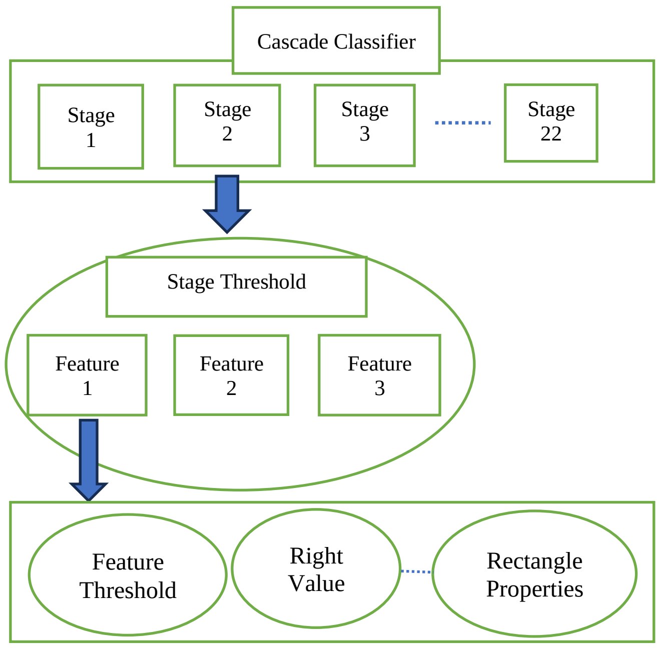

Fig. 4 This diagram shows the structure of the open CV classifier

3.5. Overlapping

Since the final detector is insensitive to small changes in translation and scale, multiple detections will usually occur around each face in a scanned image. The same is often true of some types of false positives. In practice, returning one final detection per face often makes sense. I introduced a new threshold called the 'overlapping ratio' to achieve this. So, any rectangles overlapped by more than the overlapping ratio were removed. The choice of which rectangle will replace the multi-detections depends on the areas of these rectangles. The rectangle with the most significant area replaces all other rectangles. This process decreases the number of false positives since an overlapping subset of false positives is reduced to a single detection. The overlapping ratio is a tuneable parameter; 0.8 gives acceptable results. The complete code was written under MATLAB's function 'pruneRectangles'. The following figure 5 shows a detection result with and without overlapping.

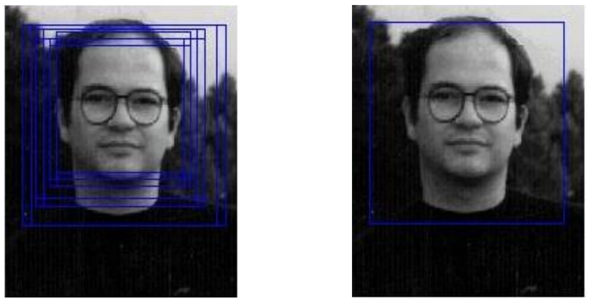

Fig. 5 Image before and after deleting the overlapping

After removing the overlapped windows, the final step is to display the detection results. The following figure 6 shows the entire algorithm for this face detection project.

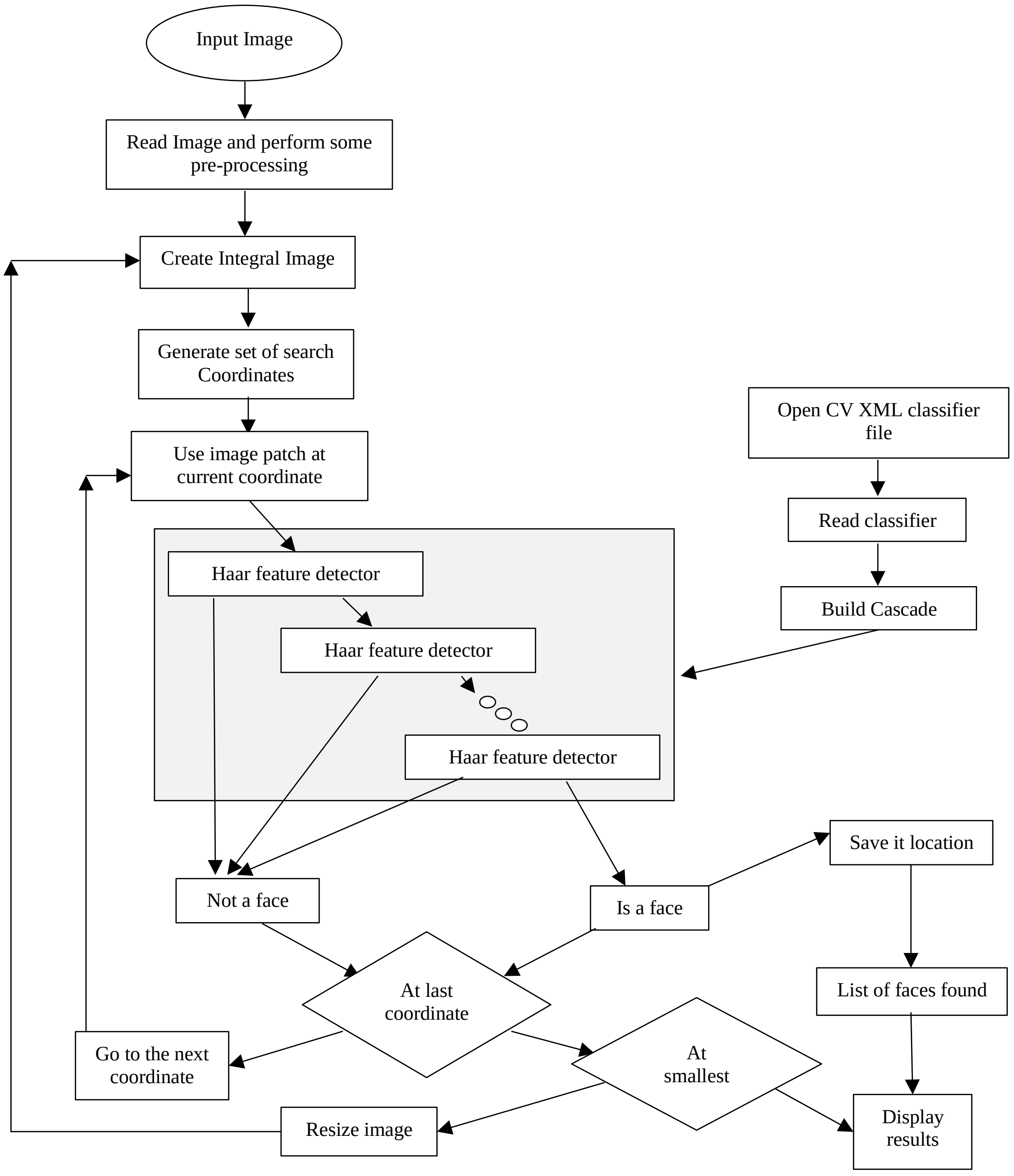

Fig. 6 Algorithm summary flow chart

4. Performance

The final detector is tested on the MIT+CMU face database. The training set contains 130 images and 507 faces. MIT+CMU are one of the most typical representative test sets for evaluation. It includes many variations of conditions such as sizes, scales, illumination, and camera variation. Past experiments show that systems that perform well on this database are not brittle or limited to a single set of conditions. This is due to the complexity of this database. [14]

The 130 images were tested individually; however, 12 gave an error during the test. The remaining 118 images were tested successfully. These remaining images contained 430 faces. Two hundred fifty-four of these faces were detected successfully, while 176 were not detected.

The detection rate for this detector was, therefore, 60%. The total false positive was 74 sub-windows. These results will be discussed further in the discussion.

Sensitivity and specificity are statistical measures of performance, where sensitivity relates to the test's ability to identify positive results, and specificity relates to the test's ability to identify negative results. The equations for calculating the sensitivity and specificity are as follows:

$$\text{Sensitivity} = \frac{\text{True positive}}{\text{True positve} + \text{False negtive}}$$

$$\text{Specificity} = \frac{\text{True negtives}}{\text{True negtives} + \text{False negtives}}$$

The sensitivity of the detector was 76.8%. The specificity, however, cannot be calculated because the true negative number is unknown. It is unknown because the MIT+ CMU database does not only contain faces; it also contains backgrounds. The following figure 7 shows the confusion matrix of this detector.

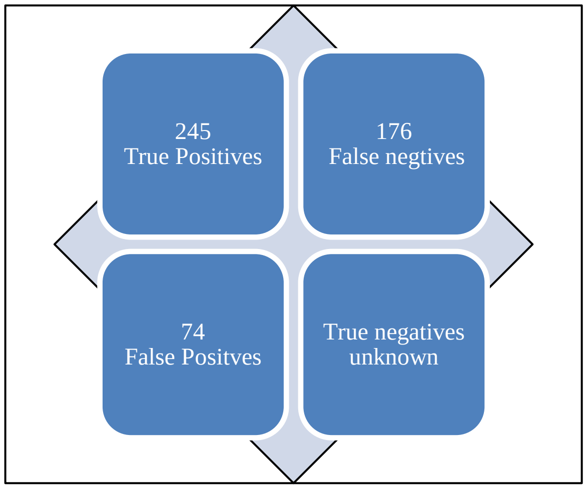

Fig. 7 Confusion matrix

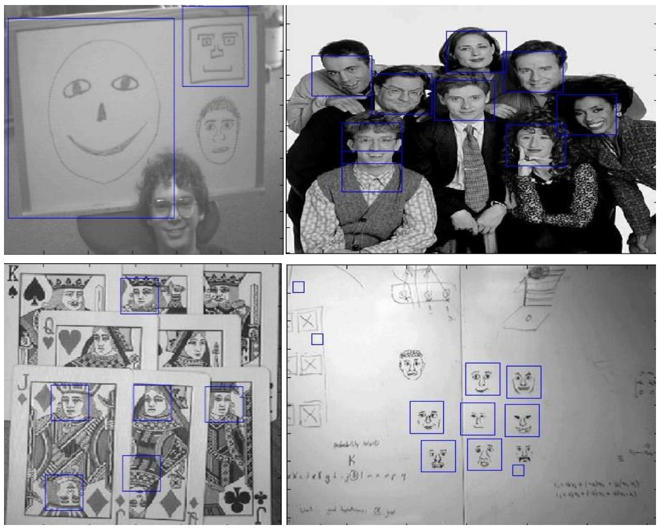

Fig. 8 Test images from MIT +CMU test set

5. Discussion

Performance measurements Criteria for any face detector are accuracy (detection rate and false negatives) and speed (computation time). A detector is classified as 'very good' if its detection rate is more than 80%. 60% detection rate is not a very good result compared to results achieved with other face detectors, especially the Viola and Jones' detector, which had achieved a 93.9% detection rate. However, there is always the trade-off issue.

The Viola and Jones' detector gives 167 false detections, while this detector only detects 74 false detections using the same testing database. The classifiers' threshold and the number of cascaded stages affect the trade-off between the detection rate and false detection. A 38-stage cascade classifier was trained in Viola and Jones' detector, which includes 4297 features. However, this detector only has 22 stages and 2135 features. The stage's number and the feature's number slightly affect the efficiency of the classification and significantly affect the computational time. Thus, Viola and Jones' detectors achieve better detection rates with the cost of more processing time.

The classifiers' threshold must be lower to achieve higher detection rates so that more faces are detected with the cost of numerous false detections. On the other hand, to lower the false detection, the classifiers' threshold must increase. Other factors affect the detector's accuracy, such as scale iteration and update. There is no specific method to set these parameters. The choice of these parameters is not defined explicitly by Viola and Jones—the more scale iteration, the higher the chance of detecting the face. However, the more scales iteration, the longer it takes to process them. The same trade-off occurs with the scale update. Lowering the scale update will increase the chance of finding faces, but it will also increase the computational time.

In the early stages of pre-processing on the image, the algorithm defined the maximum image size before continuing to further stages. The maximum image size is set at 512x512. Thus, any image with dimensions larger than defined is resized to the standard size. Therefore, the maximum scale iteration for the standard size is 18 scales, with a scale update of 0.8. This is because scale iteration directly depends on image dimension. Images that are smaller than the standard size will not be resized. As a result, their scale iteration will be less than 18. For example, 200x200 images will have 13 scale iterations. During the design, the maximum scale iteration was initially set at 18, which proved sufficient for evaluating the image in different scales. Then, based on that, the equation in Chapter 3.3 was applied, and the standard image size was set accordingly at 512x512.

Resizing larger images to smaller ones has some consequences. The pixels' intensity is not the same after the resizing due to the effect of noise and errors. As a result, some features in the original image may not appear after the resizing. Standard deviation was used to remedy this issue and improve the detector's efficiency. It works by calculating the standard deviation of the pixels inside the feature and multiplying it by the threshold to produce a new threshold. The new threshold will take into account the noise and errors that were introduced during the resizing.

The final detection result has no overlapping rectangles, which have been turned into single detection using the function ‘prune Rectangles. As a result, much false detection was eliminated. The function looks for any significant overlap between two rectangles and compares it with the defined overlap ratio. The overlap ratio was set at 0.8 by trial and error analysis. Then, if an overlap occurred, the rectangle with the smaller area was eliminated, leaving the larger one. Deciding which rectangle to use by replacing it with another was one of the design choices. The average corners value of the overlapped rectangles can be used instead of choosing the most significant area to convert the multi-detection into a single one.

6. Future Work

This project has covered the implementation of the Viola and Jones face detector using cascade classifiers provided by the open-source computer vision library. Overall, the detector has achieved an acceptable detection rate and a meagre false positives rate. The face detector performance can be improved by training the classifiers instead of using external ones. Moreover, the new classifier could be trained not only to detect frontal faces but also to detect rotated faces.

7. Conclusion

The objective of this paper has been achieved by implementing a face detector that detects and locates human faces within images. The implemented algorithm was based on Paul Viola and Michael Jones’ approach based on Haar-like features. This approach minimizes computation time while achieving a high detection rate. The implemented face detector achieved a 60% detection rate and a significantly low false detection rate. The paper involved design choices that affected the trade-off between the system’s accuracy and speed, such as scale iteration and scale update.

Methodology

Standard Terminology

- Detection rate : The number of faces correctly detected to the number of faces determined by a human expert.
- False positives : This is when an image region is declared a face, but it is not.
- False negatives : This is when an image region that is a face is not detected at all.
- False detections : False detections are the sum of False positives and False negatives.

References

[1] P. Viola, and M. Jones, “Rapid Object Detection using a Boosted Cascade of Simple Features,” *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 23, no. 1, pp. 51-64, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[2] D. Betteena Sheryl Fernando et al., “Face Recognition for Home Security,” *SSRG International Journal of Computer Science and Engineering*, vol. 6, no. 10, pp. 7-12, 2019. [CrossRef] [Publisher Link]

[3] Open Source Computer Vision Library, 2022. [Online]. Available: https://en.wikipedia.org/wiki/OpenCV

[4] Prasanna Rajendra et al., “Smart Surveillance using Open CV, Motion Analysis and Facial Landmark,” *SSRG International Journal of VLSI & Signal Processing*, vol. 7, no. 1, pp. 11-14, 2020. [CrossRef] [Publisher Link]

[5] MATLAB 7.14, The MathWorks Inc., Natick, MA, 2012.

[6] Sunil M P, and Hariprasad S A, “Facial Emotion Recognition using a Modified Deep Convolutional Neural Network Based on the Concatenation of XCEPTION and RESNET50 V2,” *SSRG International Journal of Electrical and Electronics Engineering*, vol. 10, no. 6, pp. 94-105, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[7] Ali Tarhini, (2011), Efficient Face Detection Algorithm using Viola Jones Method. [Online]. Available: https://www.codeproject.com/Articles/85113/Efficient-Face-Detection-Algorithm-using-Viola-Jon

[8] Bhumika Pathya, and Sumita Nainan, “Performance Evaluation of Face Recognition using LBP, PCA and SVM,” *SSRG International Journal of Computer Science and Engineering*, vol. 3, no. 4, pp. 58-61, 2016. [CrossRef] [Publisher Link]

[9] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky, “Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection,” *Pattern Recognition*, vol. 2781, 2003. [CrossRef] [Google Scholar] [Publisher Link]

[10] V. Muthuvel Vijai, and P. A. Mathina, “An Effective Ring Partition and Half Toning Combined Face Morphing Detection,” *International Journal of Computer and Organization Trends*, vol. 11, no. 4, pp. 10-14, 2021. [CrossRef] [Publisher Link]

[11] Cascade Classifier Training, Open Source Computer Vision. [Online]. Available: http://docs.opencv.org/doc/user_guide/ug_traincascade.html

[12] Sonia Mittal, and Sanskruti Patel, “Age Invariant Face Recognition Techniques: A Survey on the Recent Developments, Challenges and Potential Future Directions,” *International Journal of Engineering Trends and Technology*, vol. 71, no. 5, pp. 435-460, 2023. [CrossRef] [Publisher Link]

[13] Chandan A D et al., “Survey Paper on Vehicle Security using Facial Recognition & Password,” *SSRG International Journal of Electronics and Communication Engineering*, vol. 9, no. 6, pp. 5-9, 2022. [CrossRef] [Publisher Link]

[14] Face Detection Databases, (2004), Carnegie Mellon University. [Online]. Available: https://www.citationmachine.net/apa/cite-a-website