Face Detection based on Fixed Point Calculations

Shivang Dave

University of Houston - Clear Lake

Abstract

The primary objective for this research is to find a way to implement one of the integral steps of face recognition - Face detection. In this report, I talk about my implementation of the popular Viola-Jones algorithm [3]. My implementation emulates all the necessary operations of this algorithm with an alteration. These operations are restructured to adapt fixed point calculations. In addition to this, I release the source code of my implementation to the public domain. I hope that this will allow other researchers to come up with their own results and encourage further research and educational ideas in this area.

Source Code: The source code and demo of this experiment are accessible at the link mentioned at the end of this report[5].

1. Introduction

Face recognition has become significant interest in many applications of intelligent systems such as video surveillance, automotive systems and robotics [2]. The main challenge of these applications is to recognize objects with an acceptable accuracy while being able to execute it within a few hundred milliseconds on low-cost hardware devices. This research is built upon the idea of performing necessary algorithmic operations over a FPGA board as efficiently as possible. Thus, inspiration of limiting these operations to fixed-point calculations comes from one major drawback of FPGA board(s) that they require more resources to perform floating-point operations.

1.1. ***Overview of Viola-Jones Algorithm:***

1. *Input:*
   For training → Set of Labeled Images (x_1,y_1) ... (x_n,y_n)
   For detection → Image(s) (x_{test})
2. Generate Integral Image(s) from input
3. *Training* →
   For each image in image dataset →
   - Apply haar-like features and update weights for each window
   - Build a cascade of strong classifiers by combining weak classifiers (Adaboost)
   - Pass each window through stages to determine probability of finding a face
   - If initial stage fails then all other stages will be skipped (Optimization)
4. *Detection* →
   For each window in the image →

```
        - Apply cascade of classifiers
        - If it succeeds in all stages; face detected
5. End;
```

1.2. ***Background: Detection framework***

One of the first methods suggested by Paul Viola and Michael Jones in 2001 to provide object detection at very rapid rates [3]. To increase the detection efficiency adaboost machine learning algorithm is also being used. Main features of the algorithms are: Integral Image, Adaboost and Cascade of classifiers.

Integral image is necessary to compute Haar features in a constant time [1]. An integral image can be computed using equation 1. With integration of machine learning, the classification of images becomes much faster as it discards unimportant features from set leaving only the relevant ones. Each feature set is used to create classifiers.

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

$$s(x, y) = s(x, y-1) + i(x, y)$$

$$ii(x, y) = ii(x-1, y) + s(x, y)$$

where s(x,y) = cumulative row sum, s(x,-1) = 0, ii(-1,y)=0

Equation 1. Computing Integral Image [3]

Now, these classifiers are applied on each sub-window. But to improve efficiency, a cascade of strong classifiers is used and they are divided into stages. If the window fails in any of the stages then the next following stages are skipped. This can be considered as an optimization parameter. In each stage, Haar-like features of the sub-windows are calculated and compared to the threshold of that stage to see if it contains a face or not [1]. If features satisfies the threshold value only then sub-window advances to the next stage of the cascade. And once it passes all the stages then the sub-window is labeled as positive detection.

An optimal algorithm is required to work with reasonable resources. Techniques such as integral image and attentional cascade make the Viola-Jones algorithm [3] highly efficient. Using OpenCV's precomputed haar cascade classifier (frontal face) for building face detection system allows very fast execution of the algorithm.

2. Implementation

The proposed implementation mainly focuses on face detection by performing essential operations in fixed-point manner. To accomplish this task, we needed to make sure that each operation and all the values involved in such operations are bounded to a fixed point by certain precision. Python's *decimal* library enabled conversion of floating point values to fixed-point with a certain precision.

Initially, an image is provided to the program. Once that image loads successfully, pixel density matrices are created. These matrices are then used to generate integral image of the input image. In addition to the integral image, we are training a haar cascade classifier by parsing pre-compiled list of stages from OpenCV's frontal face feature set. As soon as the classifier finishes parsing, a 20x20 sub-window is generated. This sub-window performs classification by calculating and comparing threshold values for each stage. This way it is able to evaluate if that window contains a face or not.

2.1. ***Pseudo-code for implementation:***

```
1. Input: Test Image
2. Output: Image with rectangle drawn around face if detected
3. Generate Integral Image from input image
4. Detection:
       For each sub-window in the image →
           Apply cascade of classifiers
               Feature detection in sub-window
               If feature_sum doesn't match threshold →
                   Reject sub-window
               Else:
                   Move to next stage
           If sub-window passes all stages:
               Accept it as a face
       Draw a rectangle
5. Exit;
```

If the sub-window is able to validate through all the essential stage thresholds, then it is classified as a facial image. Whenever a sub-window passes stage evaluation, its position is stored in an array. This array should contain redundant values as the sub-window size is very small compared to input image. Hence, array is then cleaned to get rid of those redundant results after classification is complete. After cleaning the array it becomes easier to annotate the face by drawing a rectangle around it. Output is shown in Figure 2.

2.2. ***Dependencies:***

- Language: Python 2.7
- Library: PIL (used for image processing)
- Library: lxml (used for parsing haar features from OpenCV's XML file)
- Library: Decimal (used for converting data points to fixed points)
- Detector: Haar cascade frontal face by OpenCV (used for trained classifier)

Note: Decimal library is used to bound floating point values to decimal fixed-point representation. Each arithmetic operation such as division, square, square roots etc are bounded to fixed-point calculations.

2.3. ***How to run:***

- Move to the project directory in terminal and type following:

  python exec.py

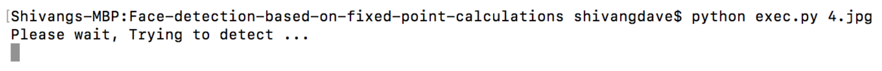
  Options:
  Execute it for all the images in *img* folder:

  python exec.py

  Execute it for a specific image in *img* folder:

  python exec.py 1.jpg

- Hit enter!

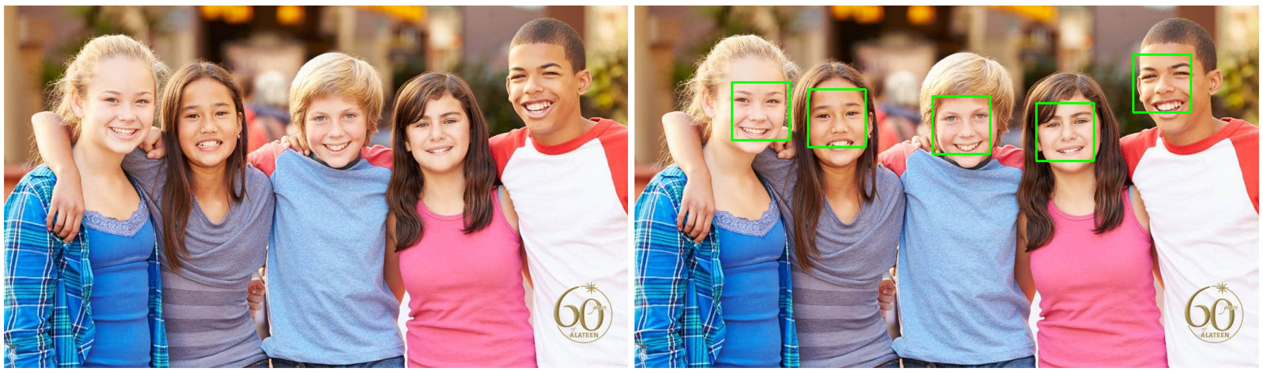


```
Shivangs-MBP:Face-detection-based-on-fixed-point-calculations shivangdave$ python exec.py 4.jpg
Please wait, Trying to detect ...
```

Figure 1: Input for single image

3. Results:

As expected due to increase in number of operations, time complexity suffers a lot. As we can see in Table 1, Fixed-point operations take almost 8x the time floating-point operations take on same image with similar detection accuracy. We can reduce the time complexity by



(a) Input Image (b) Output Image

Figure 2: Output

Image	Fixed-point	Floating-point
1.jpg	8.57	1.38
2.jpg	28.0	3.73
3.jpg	10.58	1.67
4.jpg	35.60	4.30
5.jpg	23.18	3.10

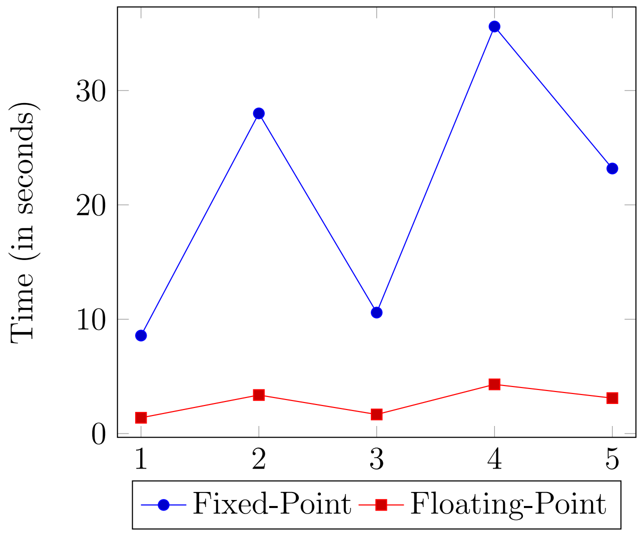
Table 1: Execution time (in seconds)

giving up the detection accuracy. So, it proves the trade-off between time complexity and detection accuracy.

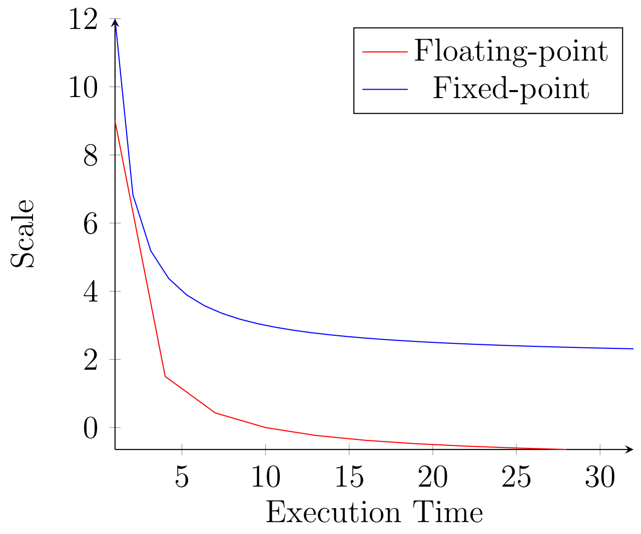
As we can see in Graph 2, Increased window scaling deteriorates detection accuracy but improves execution time significantly. Thus, in the end to decide whether to improve accuracy or improve time complexity will depend on the type of application and its use cases.

4. Conclusion:

Viola-Jones Algorithm [3] introduced an approach for object detection which minimized computation time while achieving high detection accuracy. But along with the high accuracy it required resource hungry computations. With my implementation, I have tried simplify these computations by bounding them to a certain fixed-point. This implementation can be used as a reference to design different applications for many resource restricted platforms such as FPGA boards, IoT devices and more. I hope that this detailed report will enable easy replications and comparison of results. And more importantly, will encourage further research.



Graph 1: Execution Time



Graph 2: Scale vs Execution Time

5. Image Credits:

1. Google Image Search.
2. OpenCV.
3. The USC-SIPI[7] Image Database.

6. Instructors:

1. Dr. Xiaokun Yang, UHCL
2. Dr. Kewai Sha, UHCL

7. References:

[1] Abdul Mohsen Abdul Hossen, Raheem Abd Alsaheb Ogla, Maitham Mahmood Ali, Face Detection using OpenCV's Viola-Jones Algorithm based on coding eyes *University of Technology, Baghdad, Iraq.* Iraqi Journal of Science, Vol. 58, No.2A, 2017.

[2] Imran Naseem, Roberto Togneri, Mohammed Bennamoun, Linear regression for Face Detection *Senior Member, IEEE.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, Issue: 11, Nov. 2010.

[3] Jones, M. and Viola, P., Rapid Object Detection using a Boosted Cascade of Simple Features. *Senior Member, IEEE.* Computer Vision and Pattern Recognition. IEEE Computer Society Conference, 2001.

[4] Yi-Qing Wang, An analysis of the Viola-Jones Face Detection Algorithm *CMLA, France.* Published in Image Processing On line. ISSN 2105-1232, 2014.

[5] Shivang Dave, Face detection based on fixed-point calculations *University of Houston - Clear Lake* 2018. Source Code: https://github.com/ShivangDave/Face-detection-based-on-fixed-point-calculations