INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN: 2278 – 0211 (Online)

An Optimized Implementaion of Haar like Feature Based Object Detection

Satishkumar Boggarapu
M.Tech (Vlsi&Es), Sri Vasavi Engineering College, Tadepalligudem
T.Sreenivasu
Assistant Professor, Sri Vasavi Engineering College, Tadepalligudem

Abstract:
The AdaBoost (adaptive boosting) algorithm is widely used algorithm in computer vision and machine learning systems. It is a general method for generating a strong classifier out of a set of weak classifiers. The object detection algorithm by Viola and Jones [1] with Haar-like features as weak classifiers used AdaBoosting to construct a strong classifier cascade. This popular object detection algorithm runs in real time on desktop processors running in the range of 2GHz clock frequency. But the floating point arithmetic becomes a bottleneck for embedded and mobile platforms which has limited clock speeds for low power. This paper presents an optimized fixed point alternative to the floating point arithmetic used in the time critical classifier evaluation functions. The Open-CV library is used as the base software platform. The optimized fixed point implementation is tested with several test images consisting of one or more frontal faces. The results shows that the proposed implementation has a performance improvement from 3.81 to 5.84 fps.

Keywords: *Open-CV; Computer Vision; AdaBoost; Haar Features;*

1.Introduction

Face detection is the method of identifying faces of interest in images regardless of size, position, and circumstance. A successful algorithm will find the locations and sizes of all faces in the image stream that belong to a given class with no or few "false positives". Potential face detection applications include monitoring and surveillance, human computer interfaces, smart rooms, intelligent robots, and biomedical image analysis. Face detection proposed by Viola and Jones is the first approach for real-time face detection [1]. This approach utilizes the AdaBoost algorithm [2], which identifies a sequence of rectangle features that indicate the presence of a face. The Viola and Jones algorithm is most often used for face detection, e.g., in the OpenCV library [3][4], however is applicable in other domains. This algorithm requires considerable computational power due to the sheer number of rectangle features that must be identified to detect a face. One face is comprised of a substantial amount of features, which are typically computed over a window of 24×24 pixels. To reduce computation, the detection is performed in stages so that windows in an image that do not contain something that looks similar to a face do not require computation of all features. There are many proposed approaches for face detection in a wide variety of images. While they can successfully detect frontal upright faces, many natural images include rotated or profile faces that are not reliably detected in the real world. The popular Viola Jones object detection algorithm runs in real time on desktop processors running in the range of 2GHz clock frequency. But the floating point arithmetic becomes a bottleneck for embedded and mobile platforms which has limited clock speeds for low power. This paper presents an optimized fixed point alternative to the floating point arithmetic used in the time critical classifier evaluation functions.

2.Face Detection Algorithm

The Viola and Jones [1][7] face detection algorithm is used as the basis of our design. While the input image is scanned across location and scale, this algorithm utilizes pattern classification to determine the presence of a face. Viola and Jones use a boosted collection of features to classify image windows by using the AdaBoost algorithm [2]. In the Adaboost algorithm, a set of weak binary classifiers is learned from a training set. Each classifier is a simple feature made up of rectangular sums followed by a threshold as shown in Fig. 1.

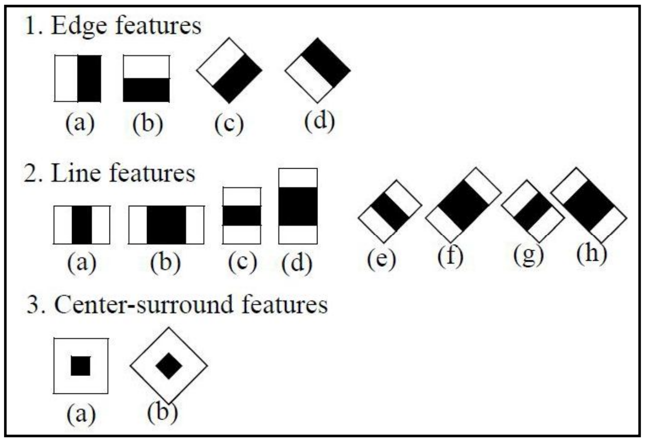

Figure 1: Feature prototypes haar-like and center-surround features

The main purpose of using features instead of raw pixel values as the input to a learning algorithm is to reduce the in-class while increasing the out-of-class variability compared to the raw data and thus making classification easier. Features usually encode knowledge about the domain, which is difficult to learn from the raw and finite set of input data. A very large and general pool of simple haar-like features combined with feature selection therefore can increase the capacity of the learning algorithm. From [2], the number of features derived from each prototype is quite large and differs from prototype to prototype and can be calculated as follows Let X=[W/w] and Y=[H/h] be the maximum scaling factors in *x* and *y* direction. A upright feature of size *w*x*h* then generates $XY(W+1-w\frac{X+1}{2})\ (H+1-h\frac{Y+!}{2})$ features for an image of size *W*x*H*, while a 45° rotated feature generates $XY(W+1-z\frac{X+1}{2})\ (H+1-z\frac{Y+1}{2})$ with z = w+h.

2.1.Integral Image

The speed of feature evaluation is also a very important aspect since almost all object detection algorithms slide a fixed-size window at all scales over the input image. As we will see, our features can be computed at any position and any scale in the same constant time. All the features can be computed very fast and in constant time for any size by means of two auxiliary images. For upright rectangles the auxiliary image is the *Summed Area Table SAT(x, y). SAT(x, y) is* defined as the sum of the pixels of the upright rectangle ranging from the top left corner at (0, 0) to the bottom right corner at (*x*, *y*) (see Figure 3a) [5]:

$$SAT(x, y) = (x + a)^n = \sum_{x' \le x, y' \le y} I(x', y')$$

It can be calculated with one pass over all pixels from left to right and top to bottom by means of,

SAT(x, y) = SAT(x, y–1)+SAT(x–1,y)+I(x, y)–SAT(x–1,y–1)
SAT(–1,y)=SAT(x,–1)=0

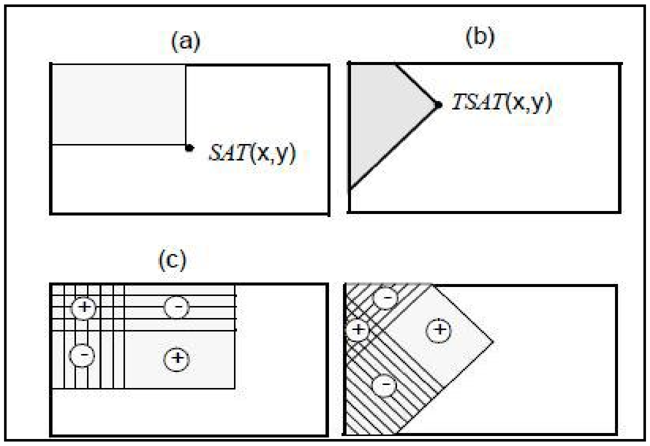

Figure2: (a) Upright *Summed Area Table* (*SAT*) and (b) *Rotated Summed Area Table* (*RSAT*); calculation scheme of the pixel sum of upright (c) and rotat ed (d) rectangles.

From this the pixel sum of any upright rectangle *r*=(*x*,*y*,*w*,*h*,0) can be determined by four table lookups (see also Figure 2(c):

Rec Sum (r)= SAT(x–1,y–1)+SAT(x+w–1, y+h–1)
–SAT(x–1,y+h–1)–SAT(x+w–1,y–1)

For 45° rotated rectangles the auxiliary image is defined as the Rotated Summed Area Table RSAT (x, y). It gives the sum of the pixels of the rectangle rotated by 45° with the right most corner at (x, y) and extending till the boundaries of the image (see Figure 2b):

RSAT(x, y) = $\sum_{x' \leq x, x' \leq x-|y-y|} I(x', y,)$

It can be calculated with two passes over all pixels. The first pass from left to right and top to bottom determines

RSAT(x, y) = RSAT(x-1, y-1)+RSAT(x-1,y)+I(x, y)-RSAT(x-2,y-1)
RSAT(-1,y) = RSAT(-2,y) = RSAT(x,-1) = 0,

Whereas the second pass from the right to left and bottom to top calculates

RSAT(x, y)=RSAT(x ,y)+RSAT(x–1,y+1)–RSAT(x–2,y)

From this the pixel sum of any rotated rectangle $r=(x,y,w,h,45°)$ can be determined by four table lookups (see also Figure 2(d))

Rec Sum (r)=RSAT(x+w, y+w)+RSAT(x–h, y+h)

–RSAT(x, y)–RSAT(x+w–h, y+w+h)

Let us assume that the basic unit for testing for the presence of an
object is a window of WxH pixels. Also assume that we have a very
fast way of computing the sum of pixels of any upright and 45°
rotated rectangle inside the window. A rectangle is specified by the
tuple r=(x,y,h,alpha) with x, y lies within the boundary of the rectangle width and height W, H respectively.and its pixel sum is denoted by RecSum(r). Two examples of such rectangles are given in Figure 2.

featureI =RecSum(□ri□)

Our raw feature set is then the set of all possible features of the form

3.OpenCV Visual Studio Setup

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision, developed by Intel, and now supported by Willow Garage. It is free for use under the open source BSD license. The library is cross-platform. It focuses mainly on real-time image processing. OpenCV includes both its traditional C interface as well as a new C++ interface. The main OpenCV site is on SourceForge at http:// SourceForge.net/ projects/ opencvlibrary.

The OpenCV is downloaded and a project is created in Visual studio. The interested libraries in the provided libraries are objdetect.lib, imgproc.lib, highgui.lib and core.lib. The objdetect.lib provides functions and data structures required for object detection. In this section the main C functions used for face detection are explained.

3.1.cvLoad

The function loads an object from a file. It provides a simple interface to *Read*. After the object is loaded, the file storage is closed and all the temporary buffers are deleted. Thus, to load a dynamic structure, such as a sequence, contour, or graph, one should pass a valid memory storage destination to the function. The function is used to load a trained cascade of haar classifiers from a file or the classifier database embedded in OpenCV. The object detection classifiers are stored in XML or YAML files.

3.2.cvHaarDetectObjects

The function finds rectangular regions in the given image that are likely to contain objects the cascade has been trained for and returns those regions as a sequence of rectangles. The function scans the image several times at different scales Each time it considers overlapping regions in the image and applies the classifiers to the regions using *RunHaarClassifierCascade* function. It may also apply some heuristics to reduce number of analyzed regions, such as Canny prunning. After it has proceeded and collected the candidate rectangles (regions that passed the classifier cascade), it groups them and returns a sequence of average rectangles for each large enough group.

3.3.Integral

This function computes integral image of a given input image.

3.4.RunHaarClassifierCascade

The function runs the Haar classifier cascade at a single image location. Before using this function the integral images and the appropriate scale (window size) should be set using SetImagesForHaarClassifierCascade. The function returns a positive value if the analyzed rectangle passed all the classifier stages (it is a candidate) and a zero or negative value otherwise.

3.5.CV setimagesforhaarclassifiercascade

The function assigns images and/or window scale to the hidden classifier cascade. If image pointers are NULL, the previously set images are used further. Scale parameter has no such a “protection” value, but the previous value can be retrieved by the GetHaarClassifierCascadeScale function and reused again. The function is used to prepare cascade for detecting object of the particular size in the particular image. The function is called internally by HaarDetectObjects, but it can be called by the user if they are using the lower-level function RunHaarClassifierCascade.

3.6.icvEvalHidHaarClassifier

This function is the critical function that evaluates and checks if a rectangle with top left cornet point (x,y) is passes a given classifier’s tree. The function reads the integral image data for the rectangles given in each node of the tree, calculates area of the rectangles and produces the difference, then it compares the area difference with the threshold of the node, if the area difference is less than the threshold at any node the evaluation moves onto the next left node of the tree or else the right node is chosen in the tree, .if the end of the tree is reached the weight of the tree (alpha) is chosen. This alpha will be summed with the alpha values obtained in all other classifier trees evaluated for this classifier stage. As we discussed, if the sum of alpha values at a stage is less than the classifier stage threshold then the rectangle is said to contain the intended object.

4.Profiling Results

To improve the performance of the application, the most frequently executed portions of an application's data flow should be identified. Because optimization of these portions will directly accelerate the application execution. Hence the face detection application that uses the Haar object detection component of the OpenCV library is profiled using Microsoft Visual C++ tool. Table 1 shows the results.

Function Name	Percentage of total execution time
icvEvalHidHaarClassifier	86
cvSetImagesForHaarClassifierCascade	2
Integral	1
All other miscllenious functions	11

Table 1: Profiling Results

From the profiling results shown, we come to know that 86% of the application total execution time is consumed in the icvEvalHidHaarClassifier function that evaluates the classifier equations. The second most frequent time consuming function is setting the integral image pointer for the cascade to be evaluated at a scale level. This function consumes 2% of the total application execution time. Integral image computation takes 1% and the rest of the functionality e.g. cascade creation, loops to iterate through all the classifiers takes 11%. Thus the classifier evaluation and image setting functions are the right candidates for optimization.

5.Fixed Point Optimization

The pseudo code for the icvEvalHidHaarClassifier function is given below.

```
do
    {
       node = next_node;
       t = node->threshold * variance_norm_factor;
       sum = Area(rect[0])*rect[0].weight;
       sum += Area(rect[1])*rect[1].weight;
       if( node has third rectangle )
          sum += Area(rect[2])*rect[2].weight;
       next_node = sum < t ? left_node : right_node;
    }
    while( next_node != NULL );
    return alpha value of the last node found;
```

From the pseudo code we can identify various operations involved in the time critical icvEvalHidHaarClassifier function. The operations are listed in Table 2.

Operation	Purpose	Count
Floating Point Additions	Area calculation of three rectangles	6
Floating Point Subtractions	Area calculation of three rectangles	6
Floating Point Additions	Producing the variable 'sum'	3
Floating Point Multiplications	Variance normalization	1
Floating Point Multiplications	Weighting the three rectangle areas	3
Floating Point Comparisons	Comparing sum with threshold	1

Table 2: Various Operations In icvEvalHidHaarClassifier

From Table II we come to know that the function involves 20 floating point operations for every node of the classification tree. During our simulations, it is observed that for lena test image with resolution 352X288, the function is called 3 million times. Since the haar training data used for simulations has 2 nodes in all the weak classification trees, total 6 million classifier nodes are evaluated. This means for the 352X288 test image, 20X6=120 million floating point operations have to be computed. Since the desktop computers runs at high clock speed in the range of 2 GHz, and have high speed multi cores, the face detection application is running at a decent speed of 5.01 fps. For embedded or mobile platforms the floating point operation will become bottleneck as each floating point operation takes several cycles latency. Also the floating point arithmetic consumes relatively more chip area. Therefore we proposed fixed point alternative to the floating point arithmetic used in the icvEvalHidHaarClassifier and related functions. The main drawback of this optimization is, the fixed point arithmetic lowers the detection rate due to a lack of accuracy. This problem is overcome by choosing the bit precisions of the fixed-point variables in such a way that the detection rate is preserved. Experiments are conducted to determine the safe bit widths for the data path signals of the various parameters, at the same time, the bit width of these parameters are chosen in such a way that they are byte aligned which is very important in processor based designs. The bit precisions of different variables of the algorithm are shown in Table 3

Variable Name	Total Bit Width	Fixed Point Format	
		Integer Bits	Fractional Bits
Integral Image Data	32	32	0
Weak Classifier Threshold	24	9	23
Variance Normalization Factor	16	32	0
Pixel Area	32	32	0
Feature Rectangle Weight	32	17	15
Weighted Sum	32	8	24

Tale 3: Bit Precision Details

6.Result

The fixed point optimized code is tested with multiple test images having one or more frontal faces. Figure 3 shows the face detection output for the lena test image. Table IV shows the comparison between the OpenCV floating point software implementation with our fixed point optimized implementa-tion.

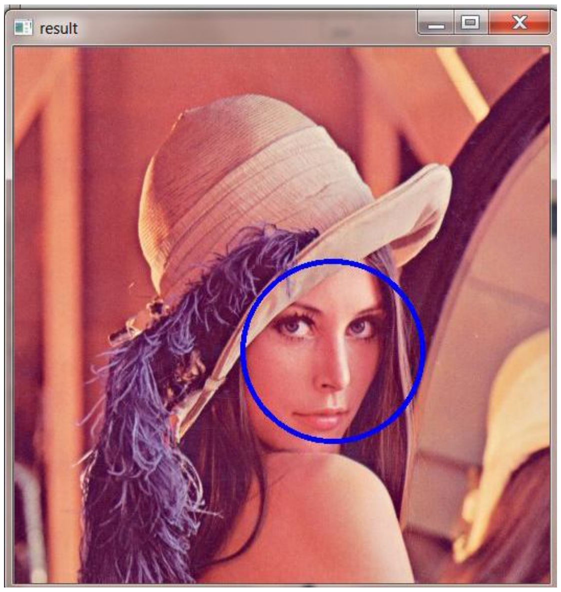

Figure 3: Face detection output for lena test image

Image Resolution	Frame Rate (fps)	
	Floating Point Implementation	**Fixed Point Implementation**
352X288	3.81	5.84
512X512	1.5	2.0

Table 4: Performance Comparison

7.Conclusion

In this paper presents an optimized fixed point alternative to the time consuming floating point arithmetic used in the time critical classifier evaluation functions of OpenCV object detection module. The fixed point implementation improved the performance from 3.81 fps to 5.84 fps.

8.Acknoledgement

I would like to thank Mr. Venkata Ganapathi Puppala, Quartics Technologies for his guidance in this research.

9.Reference

1. Viola, P. & Jones, M. (2001), "Rapid object detection using a boosted cascade of simple features," IEEE Computer Vision and Pattern Recognition (pp. I:511–518).
2. R. Lienhart and J. Maydt, "An Extended Set of Haar-like Features for Rapid Object Detection," IEEE Conference on Image Processing, vol. 1, pp. 900-903, 2002.
3. Andreas Hoffmann and Achim Nohl, "The dusk of ASIC the dawn of ASIP," Embedded Systems Conference (ESC), April 2006.
4. Junguk Cho, Bridget Benson, and Ryan Kastner, "Hardware Acceleration of Multi-view Face Detection," IEEE 7th Symposium on Application Specific Processors, 2009. SASP '09.
5. Najwa Aaraj, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha "Architectures for Efficient Face Authentication in Embedded Systems" IEEE Proceedings on Design, Automation and Test in Europe, 2006. DATE '06.
6. T. Theocharides, N. Vijaykrishnan, M. J. Irwin, "A Parallel Architecture for Hardware Face Detection," isvlsi, pp.452-453, IEEE Computer Society Annual Symposium on VLSI: Emerging VLSI Technologies and Architectures (ISVLSI'06), 2006
7. G. Bradski and A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc., 2008.
8. Computer Architecture: A Quantitative Approach, 4th Edition by John L. Hennessy and David A. Patterson
9. OpenCV Library. http://sourceforge.net/projects/opencvlibrary/.
10. Nakahara K, Sugano H, Nakamura Y and Miyamoto R, "A Specialized Processor Suitable for AdaBoost-Based Detection with Haar-like Features," IEEE Conference on Computer Vision and Pattern Recognition, June 2007. CVPR 2007.