HOG non-normalize feature extraction with only shift and addition operators

Huy-Hung Ho,Ngoc-Sinh Nguyen, Duy-Hieu Bui, Xuan-Tu Tran
SIS Laboratory, VNU University of Engineering and Technology (VNU-UET),
144 Xuan Thuy road, Cau Giay, Hanoi, Vietnam. Email: {H3email?,sinhnn_55,hieubd,tutx}@vnu.edu.vn

***Abstract*—Histogram of Oriented Gradient (HOG) is a popular feature description for the purpose of object detection. However, HOG algorithm requires performance system because of the complex operations set: arctan, square root, float multiply. Especially, the cell histogram extraction in HOG (feature extraction) is one the most complex part. In this paper, we propose a methodology to improve the cell histogram extraction part by using only shift and addition operations. For each pixel, it spends about 30 addition and 40 shift operations. Simulation results shows that the percentage error of calculations are always less than 2% with 8-bits length in fraction part. Implementing the propose methodology into hardware presents hw result**

I. Introduction

The histogram of oriented gradients (HOG) [1] is a feature descriptor used in computer vision and image processing for objection detection. It gets quantity of strong characteristic from the shape change of the object by dividing a local domain into plural blocks, and making the incline of each the histogram. Practically, HOG feature achieves very high accuracy level, it is up to 96.6% of the detection rate with 20.7% of the false positive rate [2]. Hence, it is key role of wide range application domains including robotic, security surveillance, etc. However, the complexity level of HOG is the most difficult problem when implementing it into embedded systems and battery systems.

The HOG algorithm can be separated into 3 phases: cell histogram generation, block normalization, and the SVM classification. The first phase plays as the most complexity one, and consumes the most energy. Because, extracting feature of each pixel requires series of high complex operation: arctan, square root, float multiply. Moreover, in the age of cloud computing, the two last phases can be done in the server for IoT devices. It means that the local devices only need to do cell histogram generation part, then transferring all voted bins to the servers. It allows extending the life of battery systems,

In this work, we study about the cell histogram generation and would like optimize this phase to be able to run in low resource embedded systems. Our methodology is transforming all complex operations arctan, square root and float multiply into series of shift and addition operator. As our experiments, voting to the two standard bins of pixels requires about 30 addition operations and 40 shift operations. About the accuracy of computing, the percentage errors are only about 1% of d_x d_y. hw results

The rests of the paper are organized as follows. Section II introduces the cell histogram generation and some previous optimization works. Section III shows the detail of propose methodology. Section IV presents details of hardware implementation of propose methodology and simulation results. Finally, section V gives summary and our expectation future.

II. Conventional non-normalize feature extraction in HOG

Histogram of Oriented Gradient (HOG), pioneered by DALAL and TRIGGS [1], become one of the most popular methods for feature extraction. In the conventional HOG, the cell histogram generation part is the most dominant power consumption, up to 58% HOG power in Figure 1 [3] . Authors in [4] analyzed the cell histogram generation part account for 91% workload in detection window-based approach.

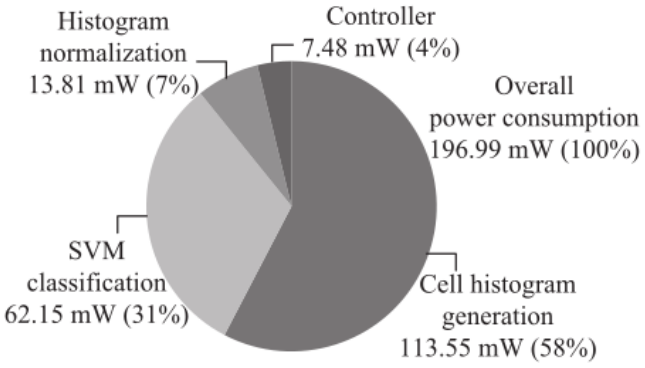

Fig. 1: Power consumption in FPGA [3]

$$d_x(x,y) = I(x,y+1) - I(x,y-1) \quad (1)$$

$$d_y(x,y) = I(x+1,y) - I(x-1,y) \quad (2)$$

$$M(x,y) = \sqrt{d_x^2 + d_y^2} \quad (3)$$

$$\phi = \arctan\left(\frac{d_y}{d_x}\right) \quad (4)$$

$$M_j = M(x,y)\frac{\phi - c_j}{w} \quad (5)$$

$$M_{j+1} = M(x,y)\frac{c_{j+1} - \phi}{w} \quad (6)$$

Where:

- $M(x,y)$ is the magnitude of the gradient of pixel $I(x,y)$
- ϕ is the angle of the gradient of pixel $I(x,y)$

- c_j, c_{j+1} are two voted standard bins.
- M_j, M_{j+1} is is the magnitude of two voted standard bins.
- w is the distance between two continuous bins, $w = 20$

The high complexity and power consumption of the cell histogram generation come from the set of complex operations: arctan, square root, float multiply, and float division. As shown, the Equation 3 - 4 use arctan and square root for computing the real gradient(angle and magnitude) of pixel $I(x, y)$. Additionally, voting the gradient of pixel $I(x, y)$ into the two continuous standard angle (bin) requires series of float multiplies and float divisions as Equation 5 and 6

In order to reduce the complexity of cell histogram generation, some publications try to avoid the nonlinear arctangent operation in the Equation 4. Several approximation methods are proposed such as piece-wise [5], look up tables (LUT) [6] [7], and using CORDIC [8] [3] [4] . They save up the power consumption and reduce the complexity of via pre-computing arctan values. However, pre-computing arctan values requires large memory area, and only solves the arctan problem. The other operations: square root, float multiply and float division in computing magnitude of pixel $I(x, y)$ and voting are still bottleneck at both performance and power consumption sides.

A more effective approach in [2] [9] has been proposed, which try to detect the two voted bin directly instead of computing the gradient of pixel $I(x, y)$ firstly. This methodology uses the fixed-point arithmetic with a 10-bit fraction part for angle calculation. This arithmetic system is equivalent to integer arithmetic in which all the values are left-shifted by 10-bits as the Equation 8. However, multiplying 1024 is not cover all cases of comparison between $\frac{d_y}{d_x}$ and $\tan\phi$. Because, both the $\frac{d_y}{d_x}$ and $\tan\phi$ are able to be infinite fraction part. Let take an example of popular bins [10..20..170](which are not same as the case in [2] [9]), $\frac{56}{97} = 0.57732$ expected to contribute its gradient to two bins: 10 and 30, but the $\frac{97}{168} = 0.577381$ should be two bins at angle 30 and 50. In another hand, multiplying 1024 is still larger than the FIXME, the expected values. The same as the LUT solution, calculating magnitude to twos voted bins in this methodology is still complex.

$$\tan\phi_1 < \frac{d_y}{d_x} < \tan\phi_2 \tag{7}$$

$$1024\tan\phi_1 \times d_x < 1024 d_y < 1024\tan\phi_2 \times d_x \tag{8}$$

III. Proposed non-normalize feature extraction

In this work, we propose a robust and efficient methodology of finding the two voted angles (bins) and calculating the magnitude of the two voted bins, which only uses a set of very simple operations: shift and addition.

As the Equation 1, 2 in the Section II, the d_x, d_y range are finite and depend on the range of pixel value. With the 8-bit length pixel, the d_x, d_y are integers in the range [-255, 255]. Consequently, the $\frac{d_y}{d_x}$ are float numbers in the set [-255, 255]. By examining all the cases of $\frac{d_y}{d_x}$ and the $\tan\phi$ values with ϕ in the first quadrant, we archive Table I. This table shows the

TABLE I: $\tan\phi$ and its nearest values $\frac{d_y}{d_x}$

Smaller and Nearest $\frac{d_y}{d_x}$	$\tan\phi$	**Greater and Nearest $\frac{d_y}{d_x}$**
$\frac{43}{244}$	tan(10)	$\frac{3}{17}$
$\frac{56}{57}$	tan(30)	$\frac{97}{168}$
$\frac{230}{193}$	tan(50)	$\frac{87}{73}$
$\frac{91}{250}$	tan(70)	$\frac{11}{4}$

$\tan\phi$ values, its smaller and nearest $\frac{d_y}{d_x}$, and its greater and nearest $\frac{d_y}{d_x}$. Those numbers will play as the critical thresholds to determine the two voted bins of a pixel $I(x, y)$. In fact, all the divisor in $\frac{d_y}{d_x}$ are smaller than the 1024 in the Equation 8.

Back to the ideas of HOG, it tries to represent the real gradient of pixel $I(x, y)$ into form of two standard bins as Figure 2 and Equation 9. The two Equations 10 and 11 are the consequent equations of the Equation 9 in the Ox and Oy axis, respectively. If we know about the d_x, d_y, the ϕ_i and the ϕ_{i+1}, the magnitude of two voted bins will be the roots of system of two Equations 10-11.

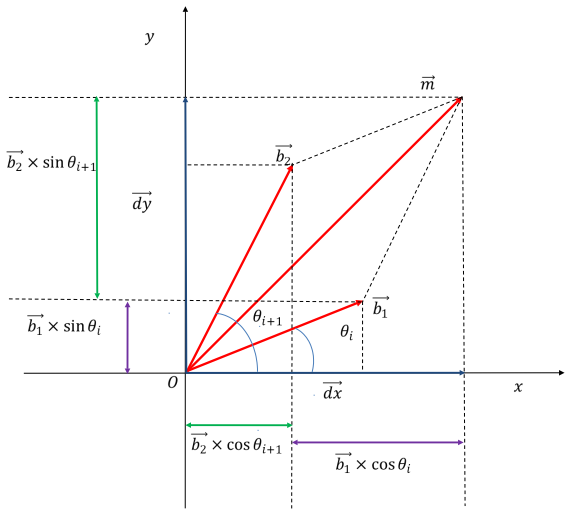

Fig. 2: Convert a vector into form of two vectors

$$\vec{M} = \overrightarrow{B_{\phi_i}} + \overrightarrow{B_{\phi_{i+1}}} \tag{9}$$

$$\begin{aligned} \|\vec{M}\|\cos(\alpha) &= d_x \\ &= \|\overrightarrow{B_{\phi_i}}\|\cos(\phi_i) + \|\overrightarrow{B_{\phi_{i+1}}}\|\cos(\phi_{i+1}) \end{aligned} \tag{10}$$

$$\begin{aligned} \|\vec{M}\| \times \sin(\alpha)| &= d_y \\ &= \|\overrightarrow{B_{\phi_i}}\|\sin(\phi_i) + \|\overrightarrow{B_{\phi_{i+1}}}\|\sin(\phi_{i+1}) \end{aligned} \tag{11}$$

$$\|\overrightarrow{B_{\phi_i}}\| = \frac{sin(\phi_{i+1})d_x - cos(\phi_{i+1})d_y}{sin(20)} \tag{12}$$

$$\|\overrightarrow{B_{\phi_{i+1}}}\| = \frac{cos(\phi)d_y - sin(\phi)d_x}{sin(20)} \tag{13}$$

From the experiments, which are shown in the two previous paragraph, we propose a methodology of calculating cell

histogram generation with only shift and addition operations. Figure 3 shows the details of the data flow of the proposed methodology. As shown, the input contains 4 neighbor pixels $I(x+1)(y), I(x-1,y), I(x,y+1), I(x,y-1)$ of pixel $I(x,y)$. The first step, the d_x and d_y are computed. At the second step, we have absolute values of d_x and d_y, and the position of the gradient of pixel I. If the sign of d_x is opposite sign of d_y, the gradient of $I(x,y)$ is in the second quadrant. Otherwise, the gradient is in the first quadrant. Because the left side of Oy axis is a mirror of right side of Oy, we only compute the magnitude of twos voted bin in the first quadrant. Next, finding the two voted angles of the pixel $I(x,y)$ is via comparison between $A \times d_x$ and $B \times d_y$, where A is the dividend , and B is the divisor in the in the Table I. In this data flow, we uses only the greater and nearest value fraction $\frac{d_y}{d_x}$ of $\tan\phi$. Depend on application, you are able to chose the left side of the Table I. After have two voted angles, the magnitude of both angle are the roots of system of two Equations 10 - 11. Finally, the $sign(d_x)$ and $sign(d_y)$ are used to determine the angle of the current pixel and its twos voted angles, which are in the first quadrant or the second quadrant? If those angles are in the second quadrant, converting the ϕ in the first quadrant into the second quadrant by equation $\phi = 180 - \phi$.

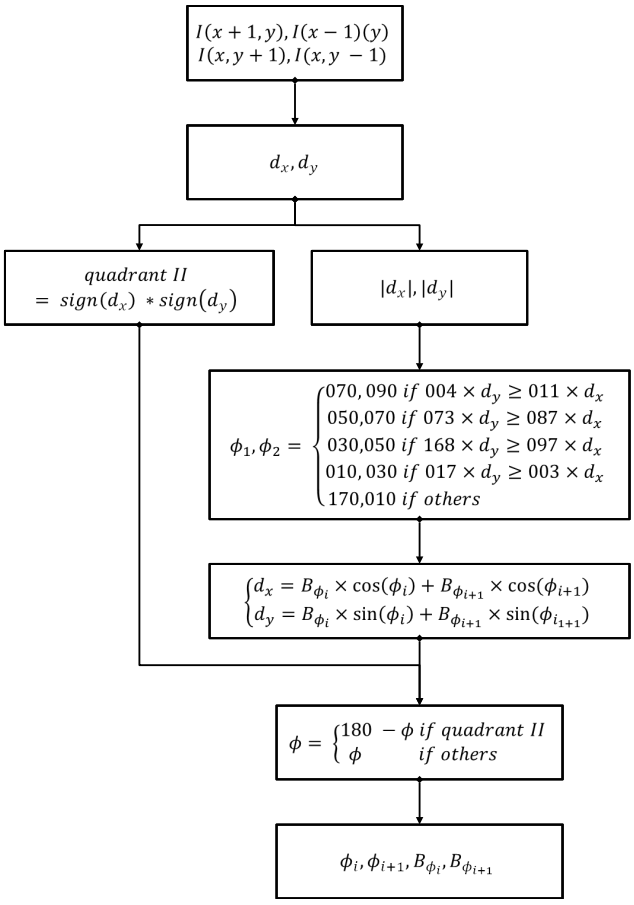

Fig. 3: Data flow of proposed methodology

As shown in the Figure 3 and Equations 12 13, all of multiply operations, we have already known one of two numbers. It allows us converting all multiples into series of shift and addition. Table II shows all of multiply cases in cell histogram generation in form of shift and addition. With $\sin$ and $\cos$ functions, 8-bits length of fraction part is used, it is equivalent to multiply 256. To change the size of fraction part, we only need to change the $\cos, \sin$ parts in the Table II. For example, if an application needs 10-bits length of fraction, the $\cos, \sin$ functions need to multiply 1024 instead of 256. From the Table II and the data flow in Figure 3, the total number of operations to calculate two voted bins spends above 30 additions and nearly 40 shifts.

IV. Implementing Proposed Methodology into Hardware and Experimental Results

A. Hardware implementation

In this section, we implement the proposed methodology into hardware and examine the results. Figure 4 shows the hardware module of the proposed methodology. It includes 4 input pixels $I(x+1,y), I(x-1,y), I(x,y+1), I(x,y-1)$ with 8-bits pixel length. Output contains the 2 couples of angle and magnitude, which represents the voted bins of current pixel $I(x,y)$. The angle output is 4-bits length, and the magnitude is 17-bits length with 8-bits of fraction part.

As shown in Figure 4, the modules includes 4 states. Firstly, the difference d_x and d_y are computed from the 4 input pixels. In the second state, the absolute of d_x, d_y are computed, and the quadrant position of the gradient of the pixel $I(x,y)$ is detected. After having those absolute values, the pre-calculating state calculates parallel all the multiples in the Table II. At the fourth state, the two voted angles are detected by only comparison operations and one subtraction, and the magnitude is from one subtraction as and only 1 multiply with $\frac{1}{\sin(20)}$ as the Equation 12 or 13.

B. Experimental results

Figure 5 shows a verification model of the proposed methodology. From the voted bin, we calculate a model of d_x and d_y by Equation 10 11: d'_x and d'_y. With the d'_x, d'_y, d_x, d_y, the percentage error of calculation at both Ox e_x and Oy e_y axises are computed. Tested with all cases, the results prove that our proposed methodology provides The percentage error of calculations e_x e_y are always less than 2% with 8-bits length of fraction part.

hw results

V. Conclusion

In this paper, we have proposed the methodology to do cell histogram generation part with only set of addition and shift operation. With this methodology, we spend about 30 additions and 40 shifts to compute the two voted bin of a pixels. Another important characteristic of this methodology is that it provides very low percentage errors, it is always less than 2%. And to change the percentage errors, we only need to change the size of fraction part via multiplying $\cos$ $\sin$ function with another number.

TABLE II: Converting all multiply to shift and addition operations with 8-bit of fraction part

$B \times d_y$	**Shift and addition**	$A \times d_x$	**Shift and addition**
$4 \times d_y$	$d_y << 2$	$11 \times d_x$	$(d_x << 3) + (d_x << 1) + d_y$
$73 \times d_y$	$(d_y << 6) + (d_y << 3) + d_y$	$87 \times d_x$	$(d_x << 6) + (d_x << 4) + (d_x << 3) - d_x$
$168 \times d_y$	$(d_y << 7) + (d_y << 5) + (d_y << 3)$	$97 \times d_x$	$(d_x << 6) + (d_x << 5) + d_x$
$17 \times d_y$	$(d_y << 4) + d_y$	$3 \times d_x$	$(d_x << 1) + d_x$
$cos(10) \times 256 \times d_y$	$(d_y << 8) - (d_y << 2)$	$sin(10) \times 256 \times d_x$	$(d_x << 5) + (d_x << 3) + (d_x << 2)$
$cos(30) \times 256 \times d_y$	$(d_y << 8 - (d_y << 5) - (d_y << 1)$	$sin(30) \times 256 \times d_x$	$(d_x << 7)$
$cos(50) \times 256 \times d_y$	$(d_y << 7) + (d_y << 5) + (d_y << 2) + d_y$	$sin(50) \times 256 \times d_x$	$(d_x << 7) + (d_x << 6) + (d_x << 2)$
$cos(70) \times 256 \times d_y$	$(d_y << 6) + (d_y << 4) + (d_y << 3)$	$sin(70) \times 256 \times d_x$	$(d_x << 8) - (d_x << 4) + d_x$
$cos(90) \times 256 \times d_y$	0	$sin(90) \times 256 \times d_x$	$d_x << 8$
$\frac{1}{sin(20)}$	$2 + \frac{15}{16}$		

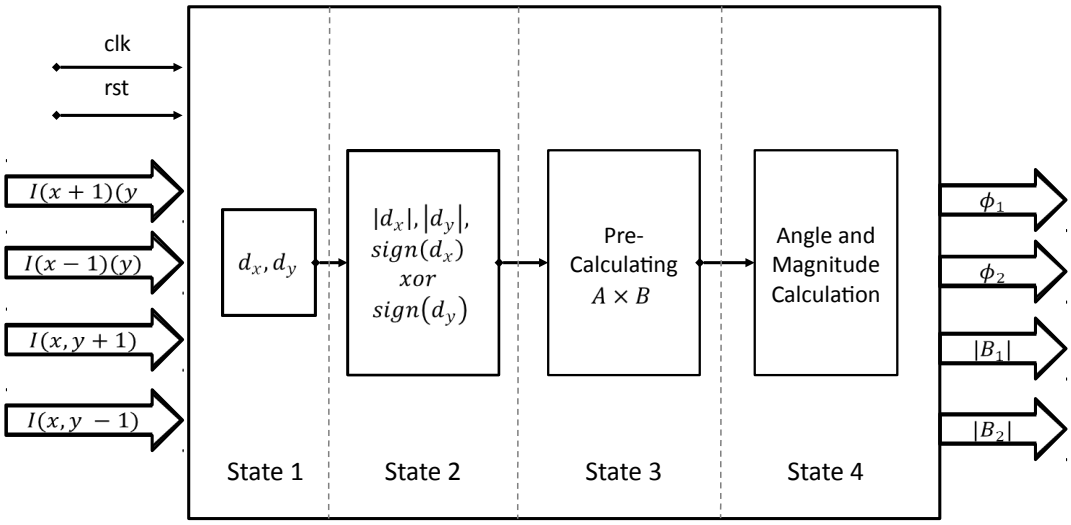

Fig. 4: Hardware implementation of proposed methodology

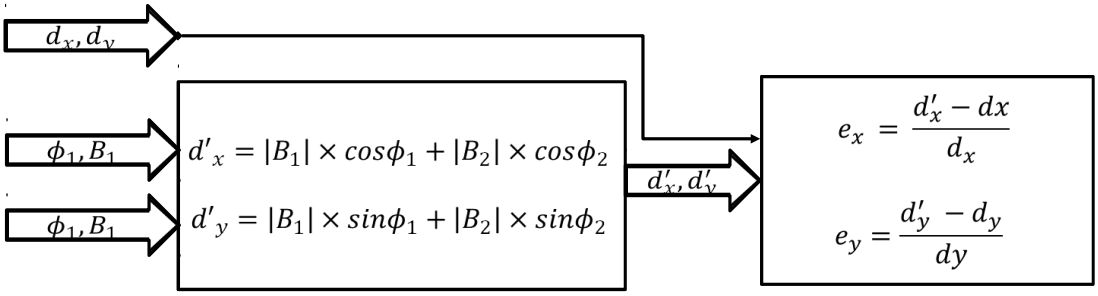

Fig. 5: Verification model

Our future work includes implementing full HOG algorithm into hardware with our optimization in cell histogram generation and the other optimizations in the normalization and SVM phases.

REFERENCES

[1] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," vol. 1. IEEE, 2005, pp. 886–893.

[2] K. Negi, K. Dohi, Y. Shibata, and K. Oguri, "Deep pipelined one-

chip FPGA implementation of a real-time image-based human detection algorithm." IEEE, Dec. 2011, pp. 1–8.

[3] K. Takagi, K. Mizuno, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A sub-100-milliwatt dual-core HOG accelerator VLSI for real-time multiple object detection." IEEE, May 2013, pp. 2533–2537.

[4] K. Mizuno, Y. Terachi, K. Takagi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "Architectural Study of HOG Feature Extraction Processor for Real-Time Object Detection," in *2012 IEEE Workshop on Signal Processing Systems*. Canada: 2012 IEEE Workshop on Signal Processing Systems, Oct. 2012.

[5] F. Karakaya, H. Altun, and M. A. Cavuslu, "Implementation of HOG algorithm for real time object recognition applications on FPGA based embedded system." IEEE, Apr. 2009, pp. 508–511.

[6] Q. Gu, T. Takaki, and I. Ishii, "Fast FPGA-Based Multiobject Feature Extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 1, pp. 30–45, Jan. 2013.

[7] F. N. Iandola, M. W. Moskewicz, and K. Keutzer, "libHOG: Energy-Efficient Histogram of Oriented Gradient Computation." IEEE, Sep. 2015, pp. 1248–1254.

[8] M. Peker, H. Altun, and F. Karakaya, "Hardware emulation of HOG and AMDF based scale and rotation invariant robust shape detection." IEEE, Oct. 2012, pp. 1–5.

[9] A. Suleiman and V. Sze, "Energy-efficient HOG-based object detection at 1080HD 60 fps with multi-scale support." IEEE, Oct. 2014, pp. 1–6.