The Implementation of Support Vector Machine (SVM) using FPGA for Human Detection

Hadee Madadum[1], Yasar Becerikli[2]

[1] Department of Computer Engineering, Faculty of Engineering, Kocaeli University
hadee.madadum@kocaeli.edu.tr
[2] Department of Computer Engineering, Faculty of Engineering, Kocaeli University
ybecerikli@kocaeli.edu.tr
[2] Department of Ankara Digital Forensic Ankara, Forensic Medicine Institution, Ankara, Turkey
yasar.becerikli@adalet.gov.tr

Abstract

Human or pedestrian detection is an attractive headline and has been proposed in computer vision and machine learning fields. Real time detection and low power system is a critical challenges. Support Vector Machine algorithm with Histograms of oriented gradients (HOG) feature descriptor is given a high successful result, fast and reliable, for human detection. Therefore, this paper demonstrates how to implement HOG feature descriptor with Support Vector Machine (SVM) using FPGA and presents a report that includes FPGA's resource utilization, time consuming, power consumption and SVM accuracy results.

1. Introduction

Human detection has been important in computer vision to understanding image and analyses the video. Human detection is applied to variety applications such as video surveillance system[1, 2], Pedestrian detection for driver assistance system [3]and mechanic system for robot [4] and etc. In human detection system, proper evaluation method is a crucial topic. There are several techniques has been proposed with high accuracy results. Rapidly processing and be able to perform real time using Haar-like and Adaboost algorithm was represent in [5, 6]. There are an efficiency algorithm using HOG and Support Vector Machine (SVM) and it was illustrated in [7, 8]. This method has an accuracy results but the training and testing phase is complicated and consume a time.

In the present, hardware machine learning implementation research headline are interested and challenging topic. Many works have been developed SVM on FPGA for human or object detection applications. A real time high resolution (1920x1080 pixels) pedestrian detection using HOG and SVM classification was implement on FPGA Xilinx Vertex5 in [9]. The detection system based on time multiplex method then using multi scale to detect moving object. SVM hardware is composed of dot product calculation, memory access controller, intermediate adder and bias adder. Hardware implementation of human detection using HOG and SVM was introduced in [10]. They use modification binary process instead of normalization process in HOG. Therefore, multiplication operation was replaced by addition operations. The result show that the resources was reduced and possible to implement to low end Xilinx Spartan3.

CoHOG,HOG with a pair of histogram, and SVM was represented in [11]. To achieve real time pedestrian detection on embedded, hardware architecture based on FPGA was designed for CoHOG. The result show that architecture can process on 38 fps with 320x240 pixels. Real time object detection for high resolution video (1920x1080 pixels) based on FPGA was proposed in [12]. HOG feature was simplified with cell based scanning, simultaneous SVM calculation, using pipeline architecture and parallelism modules. Video-based pedestrian detection for a roadside assistance unit was implemented using HOG and SVM in [13]. The hardware part is personal computer combines with FPGA and GPU. FPGA provides for HOG descriptor, GPU was used to run SVM algorithm and personal computer manages a connection between FPGA and GPU. It is giving a flexible hardware to produces a real-time processing.

According these reviews, this paper is going to implement Human detection using HOG for feature extraction method with SVM for classification algorithm in FPGA.

2. Human detection system

2.1. Histograms of oriented gradients (HOG) feature descriptor

The concept of HOG is that distribution of local intensity gradient or the edge direction (oriented gradient) can be a feature that is used to characterize an object shape and object appearance from an image. There are four parts to produce feature extraction following section below.

2.1.1. Gradient Computation

The first process is to compute the magnitude and the direction of gradient. This is achieved by image filtering operation. There are several discrete derivative masks can be applied. However, the simple 2-dimensional sharpening filter known as Sobel operator which is [-1,0,1] seem as the best solution and using the larger masks is also decrease classification performance. To calculate the point derivative, we use Sobel operator in (1) and (2).

$$G_x(x,y) = M_x * I(x,y) = P(x+1,y) - P(x-1,y) \quad (1)$$

$$G_y(x,y) = M_y * I(x,y) = P(x,y+1) - P(x,y-1) \quad (2)$$

where $M_x = [-1,0,1], M_y = [-1,0,1]^T$
$G_x(x,y), G_y(x,y)$ is gradient value, P(x,y) is illumination pixel data at I(x,y) position, and M_x, M_y is Sobel operator. Next, Equation 3 is applied to find magnitude of gradient.

$$|G(y,x)| = \sqrt{(G_x(x,y)^2 + G_y(x,y)^2)} \quad (3)$$

After that, we calculate the direction or orientation of gradient using formula 4.

$$\theta = tan^{-1}\frac{G_y(x,y)}{G_x(x,y)} \tag{4}$$

This process is to produce as emphasize the edge details in image and remove none essential information by applying a sharpening filter operation and prepare for the next operation.

2.1.2. Orientation Binning

This operation generates a histogram of gradient from magnitude and direction of gradient. It accumulates onto spatial region which is called cell. Cell is usually a rectangle that create from divide a 64x128 pixels image to 8x8 pixels which size is enough to be able to represent an interesting feature. It also creates a histogram bins which is unsigned gradient direction between 0-180 degrees or signed gradient direction (0-360 degree). The several number of bins and the angle can be used but in [16] is shown that 9 number of bins over 0-180 degree gives a highest performance. To produce a 9 bins histogram, [14] is also shown that using magnitude itself without square or square root is giving the best result.

2.1.3. Block Normalization

The objective of this process is to remove a variation that is affect to gradient magnitude such as illumination and foreground-background contrast. The normalization turn out to be essential improvement for reject the gradient affect variations. Instead of scanning every cell to normalize, [14] dedicates that it is not necessary to normalize each histogram of cell but it can combine a cell as 2x2 cell and then calculate a normalization operation and the next window cell to normalize is moved by one cell (8 pixels) and then normalization process is repeated.

2.1.4. HOG Descriptor

This process calculates the final HOG vector for entire image. The final vector is the concatenation of normalization blocks. There are 105 block of 16x16 block size and each block has 36 vectors then we will get final vector which has $105 \times 36 = 3780$ dimensions. The final HOG vector is used for classification algorithm.

2.2. Support vector machine (SVM)

Support Vector Machine is one of high accuracy supervised machine learning algorithm and it was first introduced by Cortes and Vapnik in 1995. The purpose of this method is to separate the data to two different classes by a hyper plane for N-dimensional data. The model has linear classification and can perform nonlinear classification using kernel trick. Consider data has two different (+, -), Fig.1 illustrate 2-dimensional feature using linear line (hyper plane in N-dimension) to separate different two class. Hyper plane is characterized by $wx+b=0$ while b is constant. The dot line is known as boundary and a distance between two dot lines ($\frac{2}{||w||}$) is called margin. A data on a dot line or boundary are called Support Vectors (SVs).

To separate a different class, this method search for the direction of linear line (determined by w) that gives the maximum margin. Consider as mathematic model, the positive data labelled as 1, negative data labelled as -1 then we can classify

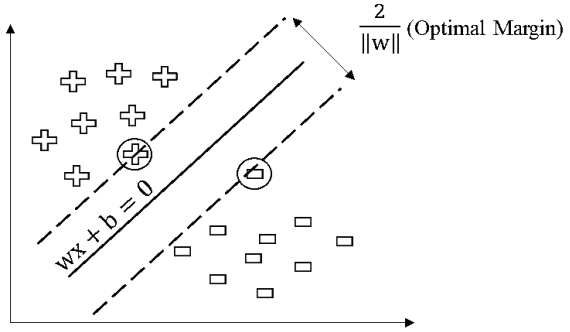

Figure 1. An example linear line separates two different class.

class by formula $w^T x + b \geqslant 0$, then x is in 1 (positive class) and $w^T x + b \leqslant 0$, then x is in -1 (negative class).

The maximum margin is given by maximize $\frac{2}{||w||}$. In other word, we can minimize $||w||$ to find maximum margin. It turn out this is a quadratic optimization problem with equality constraints.Lagrange multiplier was applied to solve this problem. The final optimization function come up with Equation 5.

$$\max_{\lambda}(\sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_i\sum_j \lambda_i\lambda_j y_i y_j (x_i x_j)) \tag{5}$$

Subject to the constraints (6).

$$\sum_{i=1}^{N}\lambda_i y_i = 0 \quad and \quad C \geqslant \lambda \geqslant 0 \tag{6}$$

The constant C is used for adjusting to make the margin large and gives a small number of margin failures.Additional, The vector w is optimal solution in linear combination of feature vector (x) that relates to λ_i when $\lambda_i \neq 0$ which has a form as Equation 7.

$$W = \sum_{i=1}^{N}\lambda_i y_i x_i \tag{7}$$

2.3. Sequential minimal optimization

Sequential minimal optimization (SMO) technique, which is represent by [15] was introduced to reduce a resource for complexity computation and consuming time. SMO is able to compute without extra matrix or numerical QP optimization step. Instead of numerical QP calculation, SMO breaks the problems into a series of smallest QP problem which is able to solve by analytically. Due to the smallest QP problem involves with two Lagrange multipliers then SMO choose two Lagrange multiplier for optimize the QP problem. At each iteration, SMO selects one Lagrange multiplier to join optimization condition then selects second Lagrange multiplier and find optimal value for these multipliers. When all Lagrange multiplier satisfy the optimization condition then the problem has been solved.

To implement SMO technique, we simplify algorithm and present as pseudo code. Assume we have input as C: regularization parameter, tol: numerical tolerance, iter : iteration time, and example data $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), .., (x^{(m)}, y^{(m)})$. The output is $\lambda \in R^m$: Lagrange multipliers, b: bias for solution.

Pseudo code SMO simplification:

- Initialize $\lambda_i = 0, b = 0$

- While (counter $<$ iter)
  - For i =1,...,m
  - Calculate $E_i = f(x_i) - y_i$
  - If ($y_i E_i < -tol \; and \; \lambda_i < C$) or ($y_i E_i > tol \; and \; \lambda_i > 0$)
    - Select J using heuristic random
    - Calculate $E_j = f(x_j) - y_j$
    - Store old λ_i
    - Compute L and H
    - If (L==H) then continues next i
    - Compute μ and if ($\mu \geqslant 0$) then continue next i
    - Compute and clip new value of λ_j
    - if ($|\lambda_j - \lambda_j^{old}| < tol$) then continue next i
    - Determine value of λ_i
    - Compute b
    - Update counter
  - End if
- End while loop

2.4. Hardware Implementation Tools

We agree that FPGA has an efficiency as low power consumption parallelism and flexibility configuration by partial reconfigurable function[16]. Unfortunately, there is a disadvantage of FPGA which is consuming a time for designing hardware and developing project. There are Verilog or VHDL which is a high-level programming language for implementation of FPGA. However, it is still uncomfortable for who does not understand in low level design as digital circuit design.

According this reason, High Level Synthesis (HLS) was introduced for hardware design process on FPGA. HLS is known as the compilation of high level language, which is C or C++, that moving the digital circuit design to be more abstract. In other word, it is a compiler for transfer a high-level language to digital circuit design or HDL. In fact, there are several software tools based on HLS that is commercial and open source [17]. This project is implementing on Xilinx FPGA so we are focusing on HLS software which is called Vivado HLS [18] that provided by Xilinx company.

3. Implementation of SVM and HOG with FPGA

This section consist of the details of data set preparing for human detection, HOG feature extraction using MATLAB and implementing SVM classification for FPGA.

3.1. Asset Preparation

The objective of this project is to detect a people from a footage video such as CCTV camera which is using for security or surveillance system in urban city. First step of implementation we prepare dataset that was collected from [19] and contains of training dataset (1821 positive images, 800 negative images) and testing dataset (182 positive images, 197 negative images).

3.2. HOG Feature Descriptor

The extractHOGFeatures MATLAB function was used to extract image to HOG feature vector. The output of this function is HOG feature vector following by defining input argument. There are various argument function that you can set it such as Cellsize (Size of HOG cell), Block size (Number of cell in block), NumBins (Number of orientation histogram bins) etc. The vector size is depending on Cellsize factor. If we choose larger Cellsize then the vector size in smaller and if we choose small Cellsize then the feature vector is larger. However, the Cellsize is represent a shape of image so if the Cellsize is small then it turns out that more clearly represent a shape of image.

3.3. Implementation of SMO in Hardware

In term of implementing in FPGA, we have to re-arrange our code from C++ language style to be more suitable for hardware design. It does not provide all function or command as general C++ to synthesize as a circuit. Furthermore, it is getting a trade-off between performance algorithm and hardware cost. The better performance of algorithm is more compensate hardware cost. There are various possible technique to perform as a hardware but there are constraints and limitation for design that we have to follow as:

- In term of hardware c++, we have to think a function in general c++ as a module which has an input and output. The size of input and output must be specific and has a limitation during size and board specification. Then we have to constraint the input data set size (X) and output from SMO which has size of output $\lambda \in R_m$ is Lagrange multipliers and B bias.
- Floating point variable is available to used but it requires a huge number of slice resources in FPGA. Vivado HLS supports float and double variable type with IEE754 standard compliance which is single precision 32 bits (24 bits fraction, 8 bits exponent) and double precision 64 bits (53 bits fraction, 11 bits exponent).
- Unable to use global variable and sub function should not have a return value. Instead of using return value, it recommends using a passing pointer variable. In addition, the recursive function does not allow to implement.
- General function such as malloc(), printf(), scanf(), pow() and etc are not be able to synthesize.

According to those limitation, SMO algorithm must be satisfy these constraints. We have done a design and the explain critical part and important part following as:

3.3.1. Input and output interface

Our SMO design has input X,Y. Output is Lagrange multiplier (λ) and bias (b). Input data X has size 60x36 (60 data set and each data set has 36 features) and input Y has size 60x1. The interface, connection with other module in FPGA, of input X and Y can be various methods. However, the different method has different hardware size. Output port consist of alpha vector λ with size is 60 and bias value. The interface of both output is axi_lite because this interface is able to transfer data between the processor and IPcore module in FPGA.

3.3.2. Data variable type

According to most of data are floating point number that consume high resources. Instead of consuming a lot of resource

by using floating point, HLS has AP_fixed variable that we can specify the number of bits for floating point number. AP_fixed has a declaring definition as AP_fixed<W,I> which W is word length in bits, I is number of integer bit. For example, if we use AP_fixed<6,3> that means we create data type length 6 bits word with 3 bits for integer number and 3 bits for floating point number.

The number of W is trade-off with resources. If we choose larger number to get more precision data then the resources are more consumed. In this project, AP_fixed<16,8> was applied because FPGA resources can support and it is acceptable precision for our dataset.

3.3.3. Memory management

According to input and output data are a vector, look up table (LUT) resources in FPGA can be perform as a vector memory. If we define a variable as an array in HLS then the program will synthesize using LUT resource. However, LUT was also require for other design such as process operation, variable memory or loop operation. Thus, if we design data using array variable then the LUT resources was over consumed. In order to decrease consuming resource, we use Vivado HLS video library function that is called Memory buffer window.

Memory buffer window is a class that allow you to declare and manage two-dimensional memory window. User can define a row and column size and it support all data type. There is sub function that provide for write and read window buffer with various way. In this work, Window buffer with size 60x30 was defined for input X data, window size 60x1 for input Y. After synthesize, it shows that the result of resource used was decreased comparing with using general array variable.

4. Results

The Zed board Zynq Evaluation and Development Kit was chosen in order to implement a prototype. Zed board was designed based on xc7z020clg484-1 FPGA from Xilinx. We have done implementation only linear kernel function. The results contain of hardware implement results section and SVM performance section.

4.1. Hardware design results

IP core was generated by HLS and synthesize results was estimated in term of timing performance and resource utilization. BRAM interface and AXI_lite was applied. However, the results that represent in next section does not include techniques for reduction a time processing or resource management because we desired to present the minimum requirement for implementing with SVM.

4.1.1. Consuming time

Time processing for SMO IP core is represented in Table 1. The default clock timing is 10 ns or 100 MHz in frequency term. Latency time is a number of clock cycles require for a function to compute all data value. If hardware design has a loop then it will calculate time each iteration and get summation. The min and max might be different because of condition branch in code design. Interval time is a number of clock cycles require for a new data is be able to compute. In general hardware, the next data can compute after last data is finished one clock cycle that mean interval time is more than latency time one clock cycle.

At 10 ns (100 MHz) clock timing, SMO IP core require minimum 1071905 clocks or 1.07 ms for processing time and maximum clock is 269755 or 2.69 ms. According to latency clock, the interval time is latency clock + 1 for both min and max requirement. There is a pipeline technique to reduce time processing of iteration loop. However, It will consumes more hardware resources.

Table 1. Processing time of SMO IP core

Clock Timing (ns)	Latency		Interval	
	Min	Max	Min	Max
10 ns	1071905	269755	1071906	2697556

4.1.2. Resource utilization

The report of resource used was represented by xilinx vivado software. For our SMO IP core was illustrate in Table 2. The significant resources consist of Block ram (BRAM), Digital signal processing (DSP48), FIFO (FF) and look up table (LUT).

Table 2. Resource utilization of SMO IP core

Resource	Estimation	Available	Utilization (%)
BRAM_18K	2	280	~ 0
DSP48E	19	220	8
FF	14312	106400	13
Look up table (LUT)	19241	53200	36

This report is using a resource based on xc7z020clg484-1 FPGA. LUT was used for 36% due to IPcore has iteration loop and if-else condition to process a whole data input. FF was used 13% for window memory that provides for HLS video library. DSP48 is 8% used to produce a mathematics operation such as adder, multiplication and division. BRAM was applied to write and read the array variable that we use it for vector data. FPGA resource can be reduced or increased depend on many factors. For example, pipeline technique will reduce processing time but it will increase a resource as a double number. another factor is if we define a bigger number of data that the output result will give more accurate but you have to pay with more resource used. Additional, different data type is also effect to resource used. In stead of using general floating number, we use fixed point number that was mention in previous chapter then we are able to reduce a number of resource used but the data precision is lower than general floating point.

4.1.3. Power consumption

Power consumption was estimated in synthesize process and it represent in Table 3. Static power is a power consumed when FPGA is powered up without running circuit. This power value is based on FPGA model. Dynamic power is a power consumed when circuit is running.

Table 3. Power consumption of SMO IP core

On chip power	Power estimation (Watt)
Static power	0.129 W
Dynamic power	0.437 W

For SMO IP core, Static power is 0.129 Watt and dynamic power is 0.437 Watt. Therefore, total power when performing SMO IP core is 0.566 Watt.

4.2. SVM performance

The experiment of training SVM model was created using linear kernel function. Data set was separated to two groups for training and cross validation. We implement with various C (Slack variable) which has values between C=$\{2^{-3}, 2^{-1}, ..., 2^{8}\}$ in order to find the lowest miss classification that will give us highest classification result. The results were illustrated in Table 4.

Table 4. Training and test results using Linear kernel function

Implementation	Training accuracy (%)	Test accuracy (%)
Software	85.16	84.17
Hardware	88.52	81.96

In software implementation (general c++), it show that training accuracy and testing accuracy was 85.16% and 84.17% respectively. Meanwhile in hardware implementation (C++ in HLS), Training accuracy is 88.52% which was higher than software implement. However, Testing accuracy in hardware, which is 81.96%, was lower than software implementation because of in hardware design we have to drop floating number to be fixed number which is poorer accuracy.

5. Conclusions

The objective of this paper is to represent an implementation pattern recognition algorithm in hardware device for embedded application. Therefore, human detection using HOG feature extraction and Support vector machine (SVM) classifier are chosen. Sequential minimal optimization (SMO) technique was applied to be algorithm for solving quadratic programming problem (QP). In hardware part, FPGA was selected as developing device because of low power consumption and capable fast computation by parallelism ability. HLS software was used for developing tools as c++ compiler and synthesizer to VHDL code. The results of hardware design implies that FPGA is capable to be efficiently hardware device for human classification problem and complexity application especially machine learning algorithm.

6. References

[1] C. Rougier, J. Meunier, A. St-Arnaud, and J. Rousseau, "Robust video surveillance for fall detection based on human shape deformation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 5, pp. 611–622, 2011.

[2] C. Conde, D. Moctezuma, I. M. De Diego, and E. Cabello, "HoGG: Gabor and HoG-based human detection for surveillance in non-controlled environments," *Neurocomputing*, vol. 100, pp. 19–30, 2013.

[3] L. Guo, P.-S. Ge, M.-H. Zhang, L.-H. Li, and Y.-B. Zhao, "Pedestrian detection for intelligent transportation systems combining AdaBoost algorithm and support vector machine," *Expert Systems with Applications*, vol. 39, no. 4, pp. 4274–4286, 2012.

[4] O. H. Jafari, D. Mitzel, and B. Leibe, "Real-time RGB-D based people detection and tracking for mobile robots and head-worn cameras," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 5636–5643, IEEE, 2014.

[5] W. Xing, Y. Zhao, R. Cheng, J. Xu, S. Lv, and X. Wang, "Fast pedestrian detection based on haar pre-detection," *International Journal of Computer and Communication Engineering*, vol. 1, no. 3, p. 207, 2012.

[6] G. Rakate, S. Borhade, P. Jadhav, and M. S. Shah, "Advanced pedestrian detection system using combination of haar-like features, adaboost algorithm and edgelet-shapelet," in *Computational Intelligence & Computing Research (ICCIC), 2012 IEEE International Conference on*, pp. 1–5, IEEE, 2012.

[7] Y. Pang, Y. Yuan, X. Li, and J. Pan, "Efficient HOG human detection," *Signal Processing*, vol. 91, no. 4, pp. 773–781, 2011.

[8] M. Bertozzi, A. Broggi, M. Del Rose, M. Felisa, A. Rakotomamonjy, and F. Suard, "A pedestrian detector using histograms of oriented gradients and a support vector machine classifier," in *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pp. 143–148, IEEE, 2007.

[9] M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, "FPGA-based real-time pedestrian detection on high-resolution images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 629–635, 2013.

[10] S. Xie, Y. Li, Z. Jia, and L. Ju, "Binarization based implementation for real-time human detection," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, pp. 1–4, IEEE, 2013.

[11] M. Hiromoto and R. Miyamoto, "Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pp. 894–899, IEEE, 2009.

[12] 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 *Signal Processing Systems (SiPS), 2012 IEEE Workshop on*, pp. 197–202, IEEE, 2012.

[13] S. Bauer, S. Kohler, K. Doll, and U. Brunsmann, "FPGA-GPU architecture for kernel SVM pedestrian detection," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pp. 61–68, IEEE, 2010.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893, IEEE, 2005.

[15] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," 1998.

[16] H. Mayurapong and W. Suntiamorntut, "Development of Remote Partial Reconfigurable Circuits Implementation on FPGA," in *The 29th International Technical Conference on Circuit/Systems Computers and Communications (ITC-CSCC)*, (Phuket, Thailand), pp. 440–443, July 2014.

[17] D. Koch, F. Hannig, and D. Ziener, *FPGAs for Software Programmers*. Springer, 2016.

[18] "Vivado Design Suite User Guide High-Level Synthesis UG902," May 2014.

[19] "Object Detection." http://www.cs.cornell.edu/courses/cs4670/2013fa/projects/p5/. Accessed: 2017-05-05.