A Research on Face Detection

Miaoshu Wangxu
Mechanical and Mechatronics Engineering Dept.
University of Waterloo
Waterloo, Canada
mwangxu@uwaterloo.ca

Md Omor Faruk
Electrical and Computer Engineering Dept.
University of Waterloo
Waterloo, Canada
mofaruk@uwaterloo.ca

Abstract - Face Detection has its wide range of applications in Face Recognitions, Facial Motion Capture, Photography, Lip Reading, and Emotional Inference. Different detection techniques have been set up over the years including knowledge-based algorithms, feature based algorithms, and appearance-based algorithms. In this report, three different face detection techniques will be implemented and compared for their speed, complexity, and cost. First algorithm is a knowledge-based detection technique known as color segmentation, where RGB & YCbCr pixels are highlighted for processing if they are in the range of skin colors. Second algorithm is called Viola-Jones, a feature-based detecting technique, where key features on a grey scale image such as eyes and ears will be extracted if they match the given patterns. Third one is an appearance-based face detection technique called Multi-Task Cascaded CNN (MTCNN), where three different CNNs will be applied to an image for detecting faces. As a result, color segmentation algorithm is found to be the best fit for hardware related design due to its less complexity and fastest speed, while Viola-Jones and MTCNN are mainly used in CPU-based software application and can both achieve a very high accuracy despite of their complexity.

Keywords - knowledge-based algorithms, feature-based algorithms, appearance-based algorithms, colour segmentation, RGB, YCbCr, Morphology Filter, FPGA. VHDL, RTL, ROM, MIF, Haar feature, AdaBoost, Integral image, Cascade Classifier, P-Net, R-Net, O-Net, MTCNN.

I. Introduction

Color segmentation algorithm first represents the image in RGB and YCbCr formats, then takes the face pixels out by checking if they are in the correct range (skin color), and in the end it applies morphology filters to take out any misclassified region [1] [3] [4] [5] [7] [8]. To detect a face, Viola-Jones uses different classifiers with adaptive boost technique to extract certain features from an object - it focusses on various sub-regions from an image and attempts to discover a face by searching for explicit features in each subregion. This is known as Haar-like features [9] [11] [12]. For Multi-task Cascaded Convolutional Neural Network (MTCNN), three independent neural networks, Proposal Network (P-Net), Refine Network (R-Net) and Output Network (O-Net), are combined [18] [19] [20] [21]. The main idea of MTCNN is to apply different size kernel matrices in CNNs to create bounding boxes for capturing faces. In this report, color segmentation technique will be implemented by Field Programmable Gate Array (FPGA). Besides, Viola-Jones and MTCNN will be implemented by Python for comparison in terms of speed, complexity, and cost.

II. Hardware Implementation of Color Segmentation Algorithm - Knowledge-Based

In general, hardware implementation of color segmentation algorithm for face detection share the same process as Matlab/Octave, but image is handled differently during hardware processing. First, the highlighted red process indicates the main difference of data processing in Field Programmable Gate Array (FPGA) compared with CPU based programming such as Matlab/Octave and will be discussed in detail. Secondly, hardware programmers pay more attention to clock latency and clock period during the Register Transfer Level (RTL) design - that is to keep the latency (# of clock cycles) minimum from input to output as well as to run each clock cycle as small as possible by using pipelining and floor planning etc. These two main differences will be used throughout the whole paper when comparing color segmentation approach with the other two software approaches.

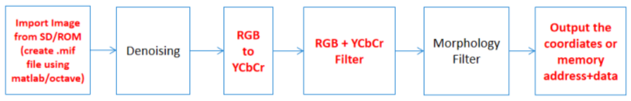
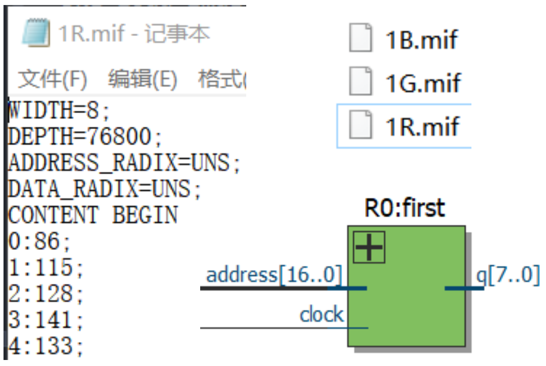

Import Image from SD/ROM (create .mif file using matlab/octave)
Denoising
RGB to YCbCr
RGB + YCbCr Filter
Morphology Filter
Output the coordiates or memory address+data

Fig.1: color segmentation process

First of all, in FPGA, images are represented in form of memory address and data, which is different from Matlab where images are handled by matrix operations. For example, a grey-scale 320 x 240 image with 0-255 each pixel can be represented by a 17-bit memory bus and a 8-bit data bus. For only functional and simulation purpose, here ROM will be used for reading data out. The below image clearly shows the data streaming process, where input of a 17-bit memory address is fed into the ROM and output of a 8-bit data can then be read out from the ROM. For color segmentation algorithm based on RGB-scale, 3 ROMs (R, G, B) will be processed independently but are driven by the same clock in hardware.



```
WIDTH=8;
DEPTH=76800;
ADDRESS_RADIX=UNS;
DATA_RADIX=UNS;
CONTENT BEGIN
0:86;
1:115;
2:128;
3:141;
4:133;
```

Fig.2: Reading Image from ROM to FPGA

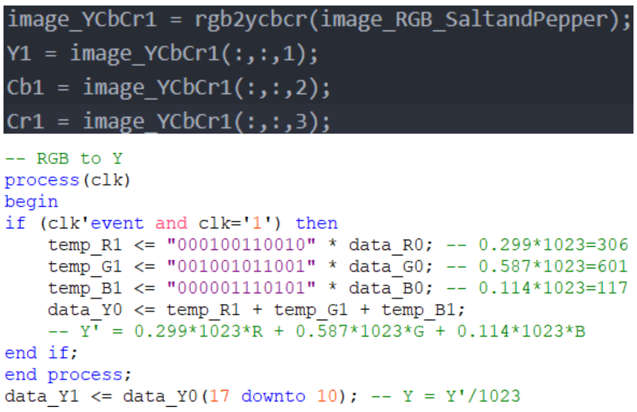
When importing an image, different from MatLab/Octave where .jpg file format is supported, in FPGA only .mif or .hex file format are accepted by complier and simulator. So, the first step is to create 3 .mif files for input matrix R, G, and B respectively. The Memory Initialization File (mif) format is strictly defined above where depth indicates the total number of pixels in an image and width is the range of pixel value in term of binary bits. For example, in test image 320 x 240, the depth is 320 x 240 = 76800 and width is 0-255 which is 8-bit.

Secondly, in FPGA, the RGB to YCbCr conversion formula is same as Matlab/Octave; however, because hardware only processes ‘0’ or ‘1’, floating point can not be used in hardware implementation. The good thing about this color segmentation algorithm is that there is only a few floating-point calculations which can then be easily eliminated using simple multiplication and division. The below formula shows RGB to YCbCr conversion which is concluded as follows by B. Iyer, S. Nalbalwar and R. Pawade. As a result, both Matlab/Octave code and Very High-Speed Integrated Circuit Hardware Description Language (VHDL) code are implemented as below.

$$Y = 0.2990R + 0.5870G + 0.1140B$$

$$Cb = -0.1687R - 0.3313G + 0.5000B + 128$$

$$Cr = 0.5000R - 0.4187G - 0.0813B + 128$$



```
image_YCbCr1 = rgb2ycbcr(image_RGB_SaltandPepper);
Y1 = image_YCbCr1(:,:,1);
Cb1 = image_YCbCr1(:,:,2);
Cr1 = image_YCbCr1(:,:,3);
```

```
-- RGB to Y
process(clk)
begin
if (clk'event and clk='1') then
      temp_R1 <= "000100110010" * data_R0; -- 0.299*1023=306
      temp_G1 <= "001001011001" * data_G0; -- 0.587*1023=601
      temp_B1 <= "000001110101" * data_B0; -- 0.114*1023=117
      data_Y0 <= temp_R1 + temp_G1 + temp_B1;
      -- Y' = 0.299*1023*R + 0.587*1023*G + 0.114*1023*B
end if;
end process;
data_Y1 <= data_Y0(17 downto 10); -- Y = Y'/1023
```

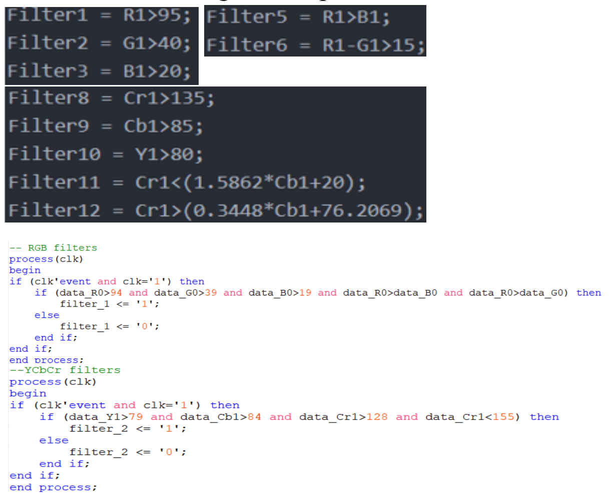
Fig.3: RGB to YCbCr (VHDL Code and Matlab/Octave Code)

Clearly in Matab/Octabe, rgb2ycbcr() function is used, that is to apply the formula to an RGB image and output an YCbCr image. In comparison, when coding FPGA, a multiplication factor of 1023 will be applied to each floating-point multiplication to eliminate the three decimal numbers and obtain an integer. For example, when calculating the luminance Y, 0.299 is multiplied by 1023 and then rounds up to 306. Then, its binary form “000100110010” is used for multiplying with the 0-255 Red pixel value. In the end, after summing up three multiplication results, Y is actually 1023 times larger and therefore its lower 10 bits need to be taken out. For this RGB to YCbCr conversion in FPGA, theoretically an error of ±6 exists due to frequently rounding-up operations and could affect the face detecting algorithm a lot. During the functional waveform simulation stage, -5 error for Cr value was found in some address locations which blocks one of the detecting condition Cr>130 and causes many pixels undetected.

Another difference is that in HDL coding, instead of coding Y=a*R+b*G+c*B in one stage, a*R, b*G, and c*B can be processed at the first clock period and the summation takes place at the second clock period. The reason of separating them into two stages is that multiplication operation takes more time than addition operation in gate/register level. For example, if multiplication operation takes 50ns and addition takes 20ns, then pack Y = a*R+b*G+c*B into only one single clock period will cost 50+2*20=90ns instead of having a faster clock period of 50ns. Most situation in hardware coding, clock period is kept as small as possible so the entire program can run at maximum frequency. This is known as pre-pipelining in HDL coding that plays an important role in hardware implementation of any (Register Transfer Level) RTL design.

Thirdly, the difference between Hardware Description Language (HDL) and Matlab/Octave is the order of code execution. CPU-based programming language scans the orders in series; it will not scan the next until the current line/loop ends. In comparison, codes/loops in FPGA-based RTL design can be processed in parallel if no dependency among them. Therefore, for FPGA-based color segmentation algorithm, all the detecting conditions can be processed at the same time which can even result in 1 or 2 cycles depends on the gate level design.

From the below HDL coding, R>94, G>39, B>19, R>B, and R>G are all processed at the same time which certainly occupies more hardware resources but can keep all the detection within a single clock period.



```
Filter1 = R1>95;
Filter2 = G1>40;
Filter3 = B1>20;
```

```
Filter5 = R1>B1;
Filter6 = R1-G1>15;
```

```
Filter8 = Cr1>135;
Filter9 = Cb1>85;
Filter10 = Y1>80;
Filter11 = Cr1<(1.5862*Cb1+20);
Filter12 = Cr1>(0.3448*Cb1+76.2069);
```

```
-- RGB filters
process(clk)
begin
if (clk'event and clk='1') then
    if (data_R0>94 and data_G0>39 and data_B0>19 and data_R0>data_B0 and data_R0>data_G0) then
        filter_1 <= '1';
    else
        filter_1 <= '0';
    end if;
end if;
end process;
--YCbCr filters
process(clk)
begin
if (clk'event and clk='1') then
     if (data_Y1>79 and data_Cb1>84 and data_Cr1>128 and data_Cr1<155) then
         filter_2 <= '1';
     else
         filter_2 <= '0';
     end if;
end if;
end process;
```

Fig.4: RGB & YCbCr Filters (VHDL Code and Matlab/Octave Code)

From below highlighted red part of the functional simulation, RGB values of 185, 181, and 182 are fed into FPGA at memory address 23542 with two cycles latency. Then after another two clock cycles, RGB values are converted into YCbCr. Finally, the signal named Filter_3 captures the valid pixel ‘1’ two clock cycles after. Therefore, in total the whole color segmentation takes 6 clock cycles in FPGA per memory address. Since the input data continuously feeds into the FPGA, the total clock cycles for testing the 320 x 240 image would take 76800 + 6 = 76806 cycles.

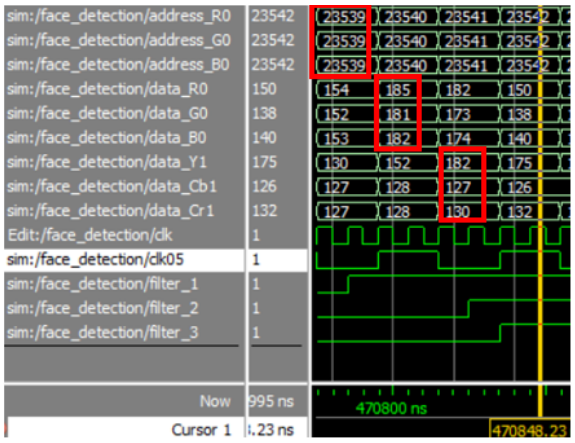

sim:/face_detection/address_R0 23542 23539 23540 23541 23542
sim:/face_detection/address_G0 23542 23539 23540 23541 23542
sim:/face_detection/address_B0 23542 23539 23540 23541 23542
sim:/face_detection/data_R0 150 154 185 182 150
sim:/face_detection/data_G0 138 152 181 173 138
sim:/face_detection/data_B0 140 153 182 174 140
sim:/face_detection/data_Y1 175 130 152 182 175
sim:/face_detection/data_Cb1 126 127 128 127 126
sim:/face_detection/data_Cr1 132 127 128 130 132
Edit:/face_detection/clk 1
sim:/face_detection/clk05 1
sim:/face_detection/filter_1 1
sim:/face_detection/filter_2 1
sim:/face_detection/filter_3 1
Now 995 ns
Cursor 1 .23 ns
470800 ns
470848.23

Fig.5: Functional Simulation of color segmentation on FPGA

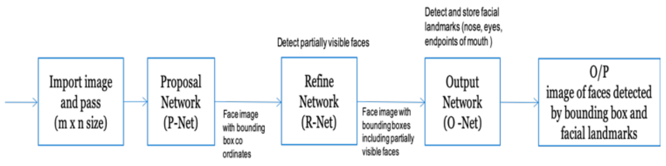
Lastly, the output from FPGA is either '1' or '0' for each memory address, which is similar to the output matrix in Matlab/Otave. Again, just like input data streaming, the output data streaming shall be handled in the same manner - 1-bit output for each memory. For the testing image of size 320*240, the output 1-bit data will either be '1' or '0' for memory address from 0 o 76799. For ease and convenience, only output the memory address with data value '1' is preferred for least wiring and logic.

Sadly, it is a relatively slow process if all memory address needs to be streamed out because total clock cycles of output data streaming = # of pixels for face region + 6 clock cycle latency; could easily go >100,000 clock cycles per image. For example, in micro SD card data storage application, output data streaming of 100,000 memory addresses require ~4000 cycles per memory address based on the SPI 4-wire communication rules which would end up with 400,000,000 clock cycles.

To deal with it, according to reference paper, a proposed method of only outputting 4 coordinates x1, x2, y1, and y2 for marking the face region is proved to be significantly efficient - 1 clock cycle possible. In this case, extra computation for converting memory address back to x,y coordinates plus more output pins are needed - for 320x240 image, 34-bit output pins = x1(9bits) + x2(9bits) + y1(8bits) + y2(8bits). Although there will be a lot more hardware resources/cost, the output data streaming issue is perfectly addressed.

III. MULTI-TASK CASCADED CNN (MTCNN) - APPEARANCE-BASED

Earlier in review project, Viola-Jones face detection algorithm was introduced. Although this technique has a very high accuracy, the false positive percentage for face detection is quite noticeable when several distortions occurs in an image such as partial visibility of faces, low illuminance, noise, blurriness, low resolution etc. To deal with these distortion factors, Multi-task Cascaded Convolutional Neural Network (MTCNN) has been proposed. MTCNN is a machine learning based approach which consists of three neural networks: Proposal Network (P-Net), Refine Network (R-Net) and Output Network (O-Net). With a combination of three neural networks, this technique makes an attempt to detect the face in each neural network independently by using common kernel matrix [18] [19].


Import image and pass (m x n size)
Proposal Network (P-Net)
Face image with bounding box co ordinates
Detect partially visible faces
Refine Network (R-Net)
Face image with bounding boxes including partially visible faces
Detect and store facial landmarks (nose, eyes, endpoints of mouth)
Output Network (O -Net)
O/P image of faces detected by bounding box and facial landmarks
Fig.6: Steps of Face detection with MTCNN

A. P-Net

Firstly, the most crucial part is to rescale the image in different sizes, i.e. making different copies of same image in various sizes to let the kernel detect different size faces from an image. Secondly, an image (M x N size) will be passed to the P-Net with kernel size 12 x 12. Starting from top left corner of the image, it will select a section from (0,0) to (12,12). The process with sections (0+2x,0+2y) to (12+2x, 12+2y) will be continued through all copies of the images by shifting the 12 x 12 kernel 2 pixels (stride) right or down simultaneously to find out whether it detects a face or not. If it detects a face, then it returns a bounding box with coordinates. As we have fed multiple images, the network will return multiple images with multiple bounding box coordinates. To select just one bounding box, an elimination method called Non-Maximum Suppression (NMS) is applied which only picks the largest bounding box. For the case of overlapping bounding boxes, if the overlap is large, then small box remains and vice versa. Finally, the image is converted back to full scale image with bounding box [18].

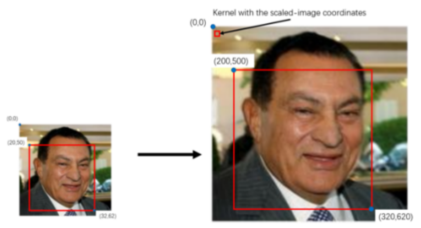


Fig.7: (a) smaller, scaled image. (b) original unscaled image [18]

B. R-Net

The main goal of this network is to get bounding box more accurate as well as to detect faces which is only partially located inside a frame. The internal process is very similar to the P-Net. In R-Net the kernel size is 24 x 24. To detect partially visible faces, the pixels of the visible face portion is copied to a new matrix with 0 padding to create a bounding box. Afterwards, the pixel values are normalized by using the following formula [18] [20].

$$x'' = 2\frac{x - \min(x)}{\max(x) - \min x} - 1$$

where $x'' \in \mathrm{R}:\ -1 < x'' < 1$ and $x \in \mathrm{R}:\ 0 < x < 255$ Finally, the image is passed to the R-Net to gather bounding box output.

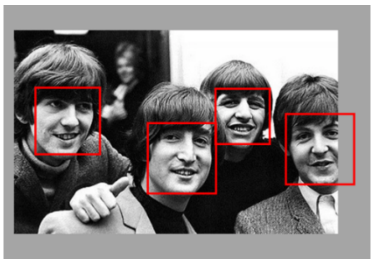


Fig.8: Detecting partly visible face with MTCNN [19]

C. O-Net

After getting the bounding box output from R-Net, the kernel matrix will be resized as 48 x 48. O-Net also follows the same strategy as P-Net except a slight difference in output. Unlike P-Net and R-Net, O-Net provides coordinates of the 5 facial landmarks – eyes (2), nose (1) and edges of mouth (2) [18] [20] [21].

IV. ADVANTAGES & DISADVANTAGES

Most algorithms can achieve very high accuracy but are too complex to be implemented using hardware due to the complexity/cost. The good thing about color segmentation algorithm is that this approach can be implemented using hardware such as Field Programmable Logic Array (FPGA) due to its less computation complexity. FPGA allows parallel computations which significantly speeds up the detection by processing independent matrix operations all at a time. In comparison, Viola-Jones and MTCNN algorithms are generally too much floating-point calculation involved and many iterations to go through therefore not possible to implement on hardware due to the cost [17]. Although the color segmentation approach is much faster and cheaper to implement, its accuracy is relatively low due to the capability of fitting all human skin tones and several limitations.

One limitation found during FPGA implementation is that the previously mentioned problem of ±6 error has a huge influence on the face region when only outputting the 4 coordinates. Another limitation for colour segmentation method is that morphology filter is always required to eliminate none-face region such as human body and people tend to use simple morphology filters for a fast detection. These simply morphology filters, such as only keep the largest detected regions or take out the lower half part of the image assuming face should be on the upper half [4], are usually limited to certain applications and could cause the whole algorithm to just corrupt due to the wrong assumptions.

For Viola-Jones face detection algorithms, although it is a relatively slow approach, researchers have found formulas for reducing the complexity of matrix operation such as calculating the sum of pixel values in a sub-region as well as set up the AdaBoost approach which can quickly and automatically fix any misclassified pixels that contain noise. One limitation for Viola-Jones approach is that it sometimes difficult to detect features due to complex background and different orientations. Another issue found is that it can be hard to locate facial features because of several corruption such as illumination [16]. Viola-Jones is relatively faster, while MTCNN is slow but has highest accuracy and widest range of applications. Both Viola-Jones and MTCNN have been implemented in python with the help of pre-trained model for face detection. After selecting some face images with several distortions, it is found that, in most cases, MTCNN gives better result. In MTCNN, as discussed previously, the main limitation is to recalculate all indexes corresponding to the stride with size 2. Therefore, it is computationally expensive.

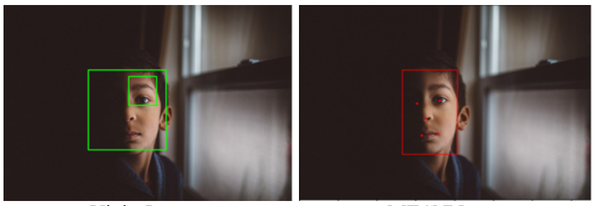


Fig.9: Viola-Jones approach detects two faces from the image as it depends on haar-like features while MTCNN successfully anticipates the that it's a single face.

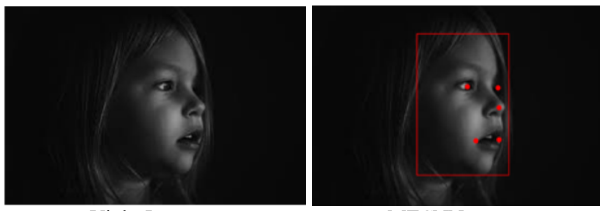


Fig. 10: Though only one side of the image is visible, MTCNN successfully detects the face.

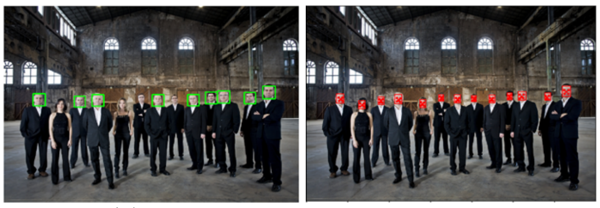


Fig. 11: This is a low resolution image and have different size faces in different location. MTCNN detected all of the faces whereas Viola-Zones approach failed in some cases.

V. Conclusion and Future Work

In conclusion, color segmentation algorithm is implemented in FPGA with only 6 clock cycle latency + m x n clock cycles for input data streaming. Parallel computation is used throughout the whole design to keep the latency as small as possible. Besides, Pre-pipelining is added for maximizing the clock frequency. More future work on a more practically level, such as synchronizing and floor-planning, needs to be done on FPGA in order to run one clock period as fast as it can.

The following tables shows the benchmark of Viola-Jones and MTCNN implementation running on a Dual Core Intel i5 CPU @ 2.4GHz, with a CPU-based Tensorflow 1.15.0. For MTCNN, the future challenge is to make this technique more efficient.

Image Size	**Total pixel**	**Process Time (Viola-Jones)**	**Process Time (MTCNN)**
460x250	118,630	0.109 s	0.139 s
600x800	633,000	0.324 s	0.489 s
1920x1080	2,104,200	0.791 s	1.127 s

Table 1: CPU time calculated for pictures containing a single frontal face

Image Size	**Total pixel**	**Process time (Viola-Jones)**	**Process Time (MTCNN)**
460x260	119,030	0.110 s	0.141 s
600x790	631,200	0.321 s	0.526 s
1780x1020	2,030,500	0.797 s	1.298 s

Table 2: CPU time calculated for pictures containing a 5-frontal face

Acknowledgment

Of the many who provided us with constructive comments, we are especially grateful to Prof. Dr. Zhou Wang Amar and Mr. Shahrukh Athar for their kind feedback and clear direction. We would like to pay our sincere gratitude to all the authors of the main paper on which we based our work.

References

[1] Patil, Prajakta M., and Y. M. Patil, "Robust Skin Colour Detection and Tracking Algorithm", International Journal of Engineering Research and Technology Vol. 1. No.8 (October-2012), ISSN: 2278-0181 (2012).

[2] KB Shaik, P Ganesan, V Kalist, BS Sathish, JMM Jenitha:"Comparative study of skin color detection and segmentation in HSV and YCbCr color space", Procedia Computer Science57 ,41-48(2015).

[3] B. Iyer, S. Nalbalwar and R. Pawade (Eds.) ICCASP/ICMMD-2016. Advances in Intelligent Systems Research.Vol. 137, Pp. 324-332. © 2017-The authors. Published by Atlantis Press.

[4] Dipankar Narendra Arya, Sivanji K.L.V., Raghunadha Reddy, Sivanantham S and Sivasankaran K, "A Face Detection System Implemented on FPGA based on RCT Colour Segmentation", 2016 Online International Conference on Green Engineering and Technologies (IC-GET), Department of Micro and Nanoelectronics School of Electronics Engineering VIT University Vellore – 632014, Tamilnadu, India

[5] M. P. Ooi, "Hardware implementation for face detection on Xilinx Virtex-II FPGA using the reversible component transformation colour space", Electronic Design, Test and Applications. Third IEEE International Workshop, Jan 2006.

[6] A. M. G. B. Reinhard, E. and P. Shirley, "Color transfer between images," IEEE Computer Graphics and Applications, vol. 21, pp. 34 – 41, 2001.

[7] Chi-Jeng Chang, Pei-Yung Hsiao, Zen-Yi Huang (2006). 'Integrated Operation of Image Capturing and Processing in FPGA', IJCSNS International Journal of Computer Science and Network Security,

[8] S.Vidya Dharan, Mohamed Khalil-Hani, Nasir Shaikh-Husin, "Hardware Acceleration of a Face Detection System on FPGA", 2015 IEEE Student Conference on Research and Development (SCOReD), VeCAD Research Laboratory, Faculty of Electrical Engineering,Universiti Teknologi Malaysia, 81310 UTM Skudai, Johor Bahru, Malaysia

[9] Sharifara, Ali, et al. "A general review of human face detection including a study of neural networks and Haar feature-based cascade classifier in face detection." Biometrics and Security Technologies (ISBAST), 2014 International

[10] Monali Chaudhari, Gauresh Vanjare, Dhairya Thakkar, Malay Shah and Amit Kadam,‖ Intelligent Surveillance and Security System‖, Vol. 3, Issue 3, March 2015, pp. 2291- 2299.

[11] Paul Viola, Micheal Jones, "Rapid object detection using a Boosted Cascade of Simple features" CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, 2001

[12] Cordiner, A.; Ogunbona, P.; Wanqing Li, "Face detection using generalised integral image features," Image Processing (ICIP), 2009 16th IEEE International Conference on , vol., no., pp.1229,1232, 7-10 Nov. 2009.

[13] Lienhart and J. Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP 2002

[14] R. Meir and G. R ̈atsch. —An introduction to boosting and leveraging‖. S. Mendelson and A. J. Smola Ed., Advanced Lectures on Machine Learning, Springer-Verlag Berlin Heidelberg, pages118–183, 2003.

[15] W.-C. Hu, C.-Y. Yang and D.-Y. Huang, "Feature-based Face Detection against Skin-color Like Backgrounds with Varying Illumination", Journal of Information Hiding and MultimediaSignal Processing, 2011

[16] Z. Li, L. Xue and F. Tan, "Face detection in complex background based on skin color features and improved AdaBoost algorithms", Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on, vol. 2, 2010, December 10-12, pp.723, 727.

[17] E. Hjelmas and B. K. Low, "Face Detection: A Survey," Computer Vision and Image Understanding, vol. 83, 2001, April, pp. 236-274.

[18] S. Farfade, M. Saberian and L. Li, "Multi-view Face Detection Using Deep Convolutional Neural Networks", *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval - ICMR '15*, 2015. Available: 10.1145/2671188.2749408.

[19] R. Yao, W. Chen and B. Benuwa, "Robust Face Detection using Convolutional Neural Network", *International Journal of Computer Applications*, vol. 170, no. 6, pp. 14-20, 2017. Available: 10.5120/ijca2017914855.

[20] W. Yang, L. Zhou, T. Li and H. Wang, "A Face Detection Method Based on Cascade Convolutional Neural Network", *Multimedia Tools and Applications*, vol. 78, no. 17, pp. 24373-24390, 2019. Available: 10.1007/s11042-018-6995-0.

[21] J. Yu and K. Sim, "Face Classification Using Cascade Facial Detection and Convolutional Neural Network", *Journal of Korean Institute of Intelligent Systems*, vol. 26, no. 1, pp. 70-75, 2016. Available: 10.5391/jkiis.2016.26.1.070.

[22] K. Luu, C. Zhu, C. Bhagavatula, T. Le and M. Savvides, "A Deep Learning Approach to Joint Face Detection and Segmentation", *Advances in Face Detection and Facial Image Analysis*, pp. 1-12, 2016. Available: 10.1007/978-3-319-25958-1_1.

[23] Xueyi Ye, X. Chen, H. Chen, Yafeng Gu and Qiuyun Lv, "Deep learning network for face detection", *2015 IEEE 16th International Conference on Communication Technology (ICCT)*, 2015. Available: 10.1109/icct.2015.7399887.

[24] J. Mehta, E. Ramnani and S. Singh, "Face Detection and Tagging Using Deep Learning", *2018 International Conference on Computer, Communication, and Signal Processing (ICCCSP)*, 2018. Available: 10.1109/icccsp.2018.8452853.

[25] D. Triantafyllidou and A. Tefas, "Face detection based on deep convolutional neural networks exploiting incremental facial part learning", *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016. Available: 10.1109/icpr.2016.7900186.