An Attempt at Face Detection on an SRC-6

David Meixner, Volodymyr Kindratenko
NCSA, UIUC
dmeixner@uiuc.edu, kindr@ncsa.uiuc.edu

1. Introduction

Object detection is an important area of image processing. Specifically, face detection finds uses in image retrieval, surveillance, and many other applications. Because real-time processing may be desirable in some applications, it is useful to find fast algorithms and hardware that can perform this operation. One such algorithm for face detection that has proven to be fast is found in Intel's Open Source Computer Vision Library (OpenCV) [1].

While the OpenCV face detection implementation works well, it still does not execute in real-time. In this work, we describe an attempt at implementing the same algorithm as used in OpenCV on an SRC-6 reconfigurable computer [2]. The SRC-6 MAP Series E processor is comprised of two FPGAs, which allows us to take advantage of parallelization and obtain increased performance from pipelined loops. These are two features commonly found in signal processing designs, so it was expected that this algorithm would execute faster on SRC-6. In the end, we found however that due to limitations on FPGA resources and the algorithm structure, the desired speedup was not achieved. However, this work has led to a better understanding of the algorithm structure and types of codes that work well on SRC-6.

2. The Algorithm

The face detection algorithm used here was originally developed by Viola and Jones [3]. It is a statistical method that uses a set of Haar-like features and a cascade of boosted classifiers. As a general overview, the algorithm begins by scanning an image with a search window of size 20x20 pixels. It then compares the search window to a set of features, grouped into stages. If the window and features of the first stage are statistically close enough, then the window is compared to a set of features in the next stage. If at any stage, a given similarity threshold is not met, the window is declared to be a non-face. On the other hand, if all stages pass, then the window is declared a face (Figure 1). This process is repeated throughout the entire image and for increasing window sizes. The features used are a set of black and white rectangles (Figure 2).

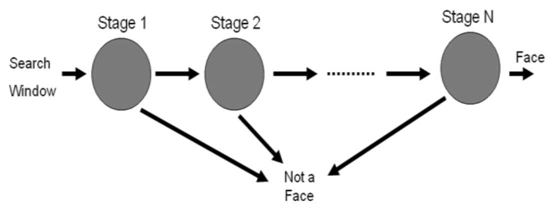


Figure 1: Cascaded classifier

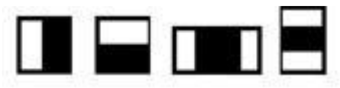


Figure 2: Examples of Haar-like features

The search window is compared to the features by summing up the pixel values over the entire feature region with a negative weighting, and summing up the pixel values over the black areas of the region with a positive weighting (Figure 3).

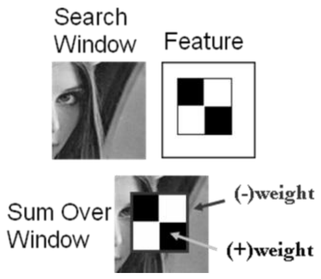
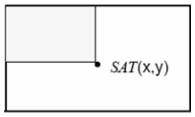


Figure 3: Comparing the search window to a feature

Therefore, it is necessary to have a fast way to compute the sum of pixels in a rectangular region. This is done by creating a summed area table of the entire image. A summed area table simply contains the sum of all pixels to the left of and above a given location (Figure 4) and the area of any rectangular region (r) is given by four table lookups:

$$RecSum(r) = SAT(x,y) + SAT(x+w,y+h) - SAT(x,y+h) - SAT(x+w,y).$$

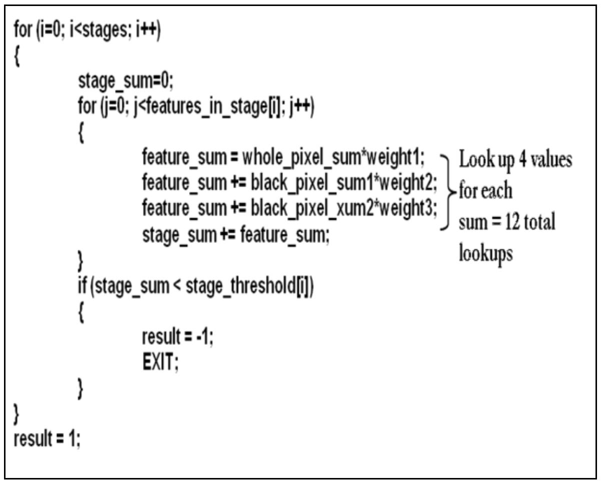


$$SAT(x,y) = \sum_{x' \leq x, y' \leq y} I(x',y')$$

Figure 4: Calculating the summed area table

3. Implementation on the SRC-6

One way to implement this algorithm is to simply move a search window through an entire image, comparing the window to the features, and repeating the process with increasing window sizes. The pseudo code for one comparison is shown in Figure 5. As stated in the previous section, to calculate the sum of pixels for a rectangular region, 4 table lookups are needed. Since each feature can be described by up to three rectangular regions (1 white and 2 black), a total of 12 lookups are needed in order to enable a fully pipelined FPGA implementation with a single clock cycle access to all the necessary data. Because the SRC-6 has only 8 onboard memory banks, all 12 lookups cannot be performed in one clock period, so it is not possible to pipeline this loop using only one clock per iteration. Also, because of the accumulator for the stage sum, as well as the early termination condition, this loop cannot be fully unrolled.



```
for (i=0; i<stages; i++)
{
        stage_sum=0;
        for (j=0; j<features_in_stage[i]; j++)
        {
                feature_sum = whole_pixel_sum*weight1;    ⎫ Look up 4 values
                feature_sum += black_pixel_sum1*weight2;  ⎬ for each
                feature_sum += black_pixel_xum2*weight3;  ⎭ sum = 12 total
                stage_sum += feature_sum;                   lookups
        }
        if (stage_sum < stage_threshold[i])
        {
                result = -1;
                EXIT;
        }
}
result = 1;
```

Figure 5: Feature comparison subroutine

4. An Alternative Implementation

In the original OpenCV implementation of the algorithm, the size of the search window was increased at each iteration, requiring larger and larger search windows. If instead the search window is kept fixed (at 20x20 pixels) and the image is shrunk on each iteration, the search window size remains small. This means that the contents of the search window can be stored in the available BRAM instead of the onboard memory, which implies one is no longer limited by having only 8 memory banks to work with. Instead, several search windows can be analyzed simultaneously, and now the limitation is the number of search windows that can be run simultaneously given limited FPGA resources.

By using both of the FPGAs available in the SRC-6, a total of seven search windows can be compared simultaneously. This method gives a speedup over the original implementation because of the parallelization, as well as the loop in Figure 5 now being fully pipelined with one clock per iteration. However, for an image of size 640x480, this implementation only runs at about 0.5 frames per second (fps) whereas an ideal real-time application would run at 30 fps.

5. Conclusions

In this work, we examined the implementation of a face detection algorithm on an SRC-6 reconfigurable computer. It was originally expected that considerable speedup would be achieved on this particular hardware. However, limitations from the code that could be fully pipelined and the limited FPGA resources prevented this hardware from achieving faster execution. While nothing can be done about the former condition, the latter can be improved if more FPGA resources were available. If this were the case, more search windows could be analyzed simultaneously, implying the execution speed is proportional to the size of the FPGA.

6. References

[1] G. Bradski, A. Kaehler, V. Pisarvesky, "Learning-Based Computer Vision with Intel's Open Source Computer Vision Library", *Intel Technology Journal*, May 2005.

[2] SRC Computers Inc., SRC Systems and Servers Datasheet, Colorado Springs, CO, 2005.

[3] P. Viola, M. Jones, "Rapid Object Detection using a Boosted Cascade of Simple Features," *IEEE CVPR*, 2001.