Hardware Acceleration of Convolutional Neural Networks for Drowsiness Detection Applications

Samuel Bruner
Ming Hsieh Department of Electrical Engineering
University of Southern California
sbruner@usc.edu

Juan P. Mejia
Ming Hsieh Department of Electrical Engineering
University of Southern California
mejiasil@usc.edu

Jun Yup Lee
Ming Hsieh Department of Electrical Engineering
University of Southern California
junyuple@usc.edu

***Abstract*—Over the past few years, Convolutional Neural Networks (CNN) have been successfully used in many image recognition applications. CNN image and pattern recognition capabilities can be applied to the prevention and detection of drowsy driving. The operation of commercial and passenger vehicles while drowsy, fatigued, or asleep is a dangerous problem in the United States. In this paper, the CNN OpenCV eye detection algorithm will be implemented to indicate when a driver is entering a drowsy state. The bottlenecks in execution are identified using the runtime performance in Python. A 3x3 mesh-based NoC architecture is implemented in Verilog to facilitate hardware acceleration using parallelization.**

1 Introduction and Motivation

Drowsy driving continues to be a dangerous problem in the United States. In 2013, The National Highway Traffic Safety Administration estimated that, as a result of drowsy driving [1][2], there were 72,000 crashes and 800 deaths, which accounts for 2.5% of all fatal motor vehicle crashes. However, these statistics are underestimated according to the Government Highway Safety Association / StateFarm August 2016 report in which it is estimated that drowsy drivers were responsible for 5,000 fatal crashes and an estimated annual societal cost of $106 billion. Machine learning can be used to solve this problem. Convolutional neural networks can be trained to identify drowsiness by monitoring driver eye activity which results in increasing driver's safety. The potential challenges of this project include: the accuracy of detecting drowsiness, identifying bottlenecks in the performance of the CNN execution in Python, and translating portions of the CNN algorithm code to Verilog.

2 Previous Work

Sleep alertness for vehicle operators is a well-known problem that companies have combated through vehicle assistant driving or swerve control. Our approach is to implement a solution that utilizes Convolutional Neural Networks to help assist vehicle operators. There are many studies and applications involving Convolutional Neural Networks (CNN), especially in recent years. CNN's are the main approach to track facial expression recognition, including eye openness.

3 Approach

The drowsiness detection algorithm is initially implemented in Python using OpenCV code. The code leverages a Dlib Python package 68-point facial landmark predictor, which is a pretrained facial recognition shape predictor. This facial landmark predictor includes six eye landmarks for each eye given by figure 1.

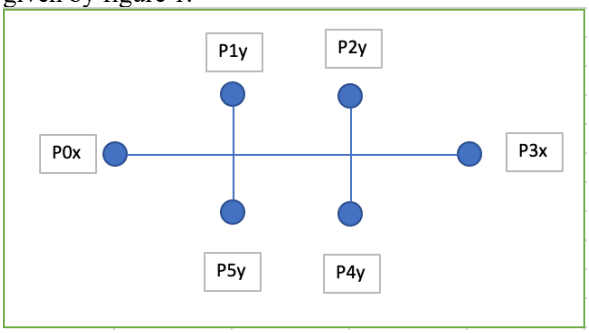


Figure 1: Eye Landmarks for EAR Calculation

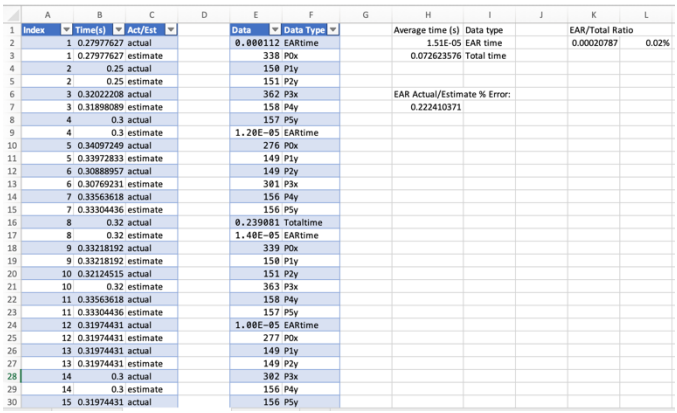
These six eye landmark coordinates are extracted from the algorithm. The distance of the opposite pairs of each eye

landmark are calculated, and the distance values are used to calculate the Eye-Aspect Ratio given by (1).

$$(1) EAR = \frac{(A+B)}{(2*C)}, \ where\ A = P5y - P1y, B = P4y - P2y, C = P3x - P0x$$

Through utilizing the OpenCV packages, the following statechart will be implemented, figure 3. The initial state is set by detecting if the eyes are within the cameras vision. When the eyes have been detected, the system continues to check if they become closed by comparing the eye aspect ratio to the threshold value. When the aspect ratio falls below the threshold value for 1200ms (three times the average blink duration, 400ms), the statechart transitions to the alert state where an alarm is sounded. The alarm continues to sound in this state until the eyes are reopened, and the startchart transitions back to the open state.

An analysis of 302 EAR calculations in software (Python), using the CPU clock, revealed that the average EAR calculation takes 15.1µs, figure 2.



Index	Time(s)	Act/Est	Data	Data Type		Average time (s)	Data type		EAR/Total Ratio	
1	0.27977627	actual	0.000112	EARtime		1.51E-05	EAR time		0.00020787	0.02%
1	0.27977627	estimate	338	P0x		0.072623576	Total time			
2	0.25	actual	150	P1y						
2	0.25	estimate	151	P2y						
3	0.32022208	actual	362	P3x		EAR Actual/Estimate % Error:				
3	0.31898089	estimate	158	P4y		0.222410371				
4	0.3	actual	157	P5y						
4	0.3	estimate	1.20E-05	EARtime						
5	0.34097249	actual	276	P0x						
5	0.33972833	estimate	149	P1y						
6	0.30888957	actual	149	P2y						
6	0.30769231	estimate	301	P3x						
7	0.33563618	actual	156	P4y						
7	0.33304436	estimate	156	P5y						
8	0.32	actual	0.239081	Totaltime						
8	0.32	estimate	1.40E-05	EARtime						
9	0.33218192	actual	339	P0x						
9	0.33218192	estimate	150	P1y						
10	0.32124515	actual	151	P2y						
10	0.32	estimate	363	P3x						
11	0.33563618	actual	158	P4y						
11	0.33304436	estimate	157	P5y						
12	0.31974431	actual	1.00E-05	EARtime						
12	0.31974431	estimate	277	P0x						
13	0.31974431	actual	149	P1y						
13	0.31974431	estimate	149	P2y						
14	0.3	actual	302	P3x						
14	0.3	estimate	156	P4y						
15	0.31974431	actual	156	P5y						

Figure 2: Software Analysis

In order to compute Euclidian distance more effectively with the hardware acceleration, we need to avoid sequential computations. Instead, we will use 3X3 mesh NoC map that we can able to compute the ALU operations in parallel. In the first batch of operations (sub1, sub2, sub3), all the points on eye area will be subtracted (P2-P6, P3-P5, P4-P1) to be used in the future operations. Next, the results from sub1 and sub2 will be added, and at the same time, mul operation will be executed with the result from sub3 and an integer 2. For the third batch, the results from previous multiplication will be divided by the previous add to get the intended result. The formation of the 3X3 NoC map is presented above; node 0, 1, and 2 will not be used. Node 3 to node 5 will be dedicated for subtract operations, node 6 for add, node 7 for division, and lastly node 8 will be used for multiplication. Therefore, this NoC map can emulate the Euclidean distance computation, Eucl = (A+B)/(2*C).

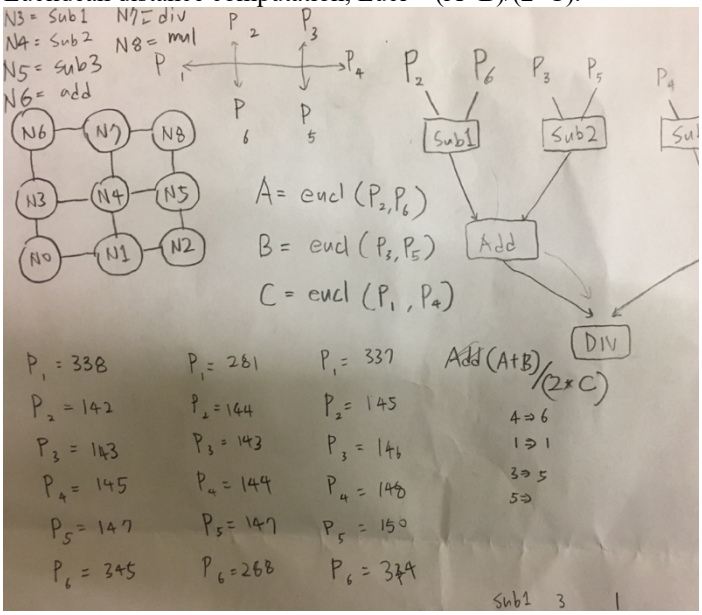


Figure 3: 3x3 Meshed NoC Mapping

For the Verilog implementation of 3x3 NoC mapping, the very first step is to instantiate the network. This process requires finding out the connecting signals between processing elements, interfaces, and the MKnetwork. The example pseudocode looks like the following:

```
module top_module(clk, reset, p1, p2, p3, p4, p5, p6, result, result_valid)
declare inputs
input clk;
input reset;
input [63:0] p1, p2, p3, p4, p5, p6;
output [63:0] result;

//Processing Elements Interconnects (PE to Interface)
wire pi_data for all operations
wire pi_valid for all operations
wire pi_dst for all operations

//Interface Interconnects (Interface to mkNetwork)
wire im_data for all operations
wire im_valid for all operations

//Network Interconnects (mkNetwork to PE)
wire [70:0] mp_flit0 to 8

//Processing Elements Instantiate

PE_sub1 sub1_m (.clk(clk), .reset(reset), .p2(p2), .p6(p6), .i_data(pi_data_s
                .i_data_valid(pi_valid_sub1), .i_dst(pi_dst_sub1));

               repeat for all operations (total 6)

//Interface Instantiate
interface_ if_sub1(.clk(clk), .reset(reset), .i_data(pi_data_sub1), .i_data_v
                .i_dst(pi_dst_sub1), .o_data(im_data_sub1), .o_data_valid

               repeat for all operations (total 6)

//mkNetwork Instantiate
mkNetwork network1 fill in the input outputs according to its port


//Final Output to SSD
result from DIV operation, mp_flit7[63:0]
result_valid = valid bit from DIV operations
```

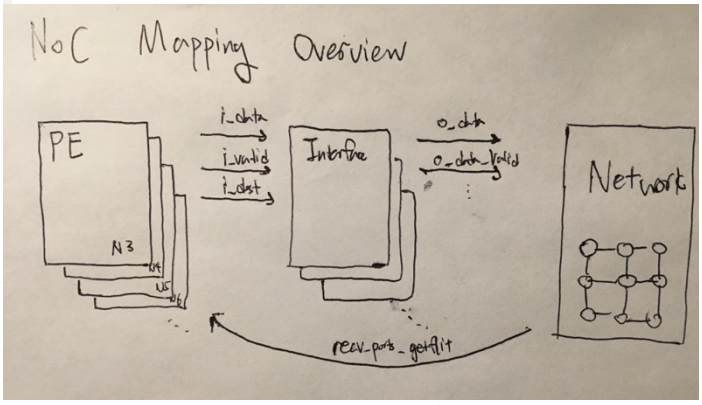


Figure 4: NoC Instantiation and Pseudo Code.

The next Verilog implementation is formatting the data in the processing element files according to given nodes. Since we have 6 nodes with its unique ALU operations, we used 6 modules to accommodate each functionality. The following pseudo code represent sub1 PE module which is similar to the other 5 modules with different ALUs:

```
module PE_sub1(clk, reset, p2, p6, i_data, i_data_valid, i_dst);

declare inputs and outputs;
input [63:0] p2, p6;
output i_data, i_data_valid;
destination to Node 6

always @(posedge clk)
begin
    if (reset)
    begin
        initial values
    end

    else
    begin
        if(counter == 0)
        begin
        add p2 and p6
        input_valid = 1;
        counter = counter + 1;
        end
        else if(counter == 1)
        begin
            input_valid = 0;
        end
    end

end

***repeat for total 6 modules (sub1, sub2, sub3, add, mul, div)
```

Figure 5: NoC Processing Element Formatting and Pseudo Code.

Depending on where the inputs are coming from, we set the counter differently so the data_in from the previous clock can recirculate back to execute ALU operation with the one coming in currently. The example above is sub1 processing element, which has two inputs directly coming in from the testbench and ready to execute the subtraction at the next clock. Here, we only need to count once. However, the add operation will take two input values from the previous node. Since when the values come in, we will distinguish if the previous flit with the integer values have a valid bit 1. If so, we will recirculate back to the add node until the next integer with valid flit comes into the add node. If the two inputs are present at the time, we then execute the ALU operation, such operation (add, div) takes one extra count for the flit recirculation.

Lastly, we need to organize the bit location of each flit so that the valid bit, tale bit, data bits, and destination bits can be allocated before sending flits to the network map. The flit has 71 bits total; 64 bits belong to data, 4 bits belong to destination node, each valid, tail, and virtual channel has 1 bit.

```
reg [70:0] o_data_bf;
reg [70:0] flit_complete;

assign o_data = flit_complete;

//Generating Clock
/*
parameter HalfClkPeriod = 5;
localparam ClkPeriod = 2*HalfClkPeriod;
initial clk = 0;
always #(HalfClkPeriod) clk = ~clk
*/


//Creating Flit
always @ (posedge clk)
begin
    o_data_valid = 0;
    if(i_data_valid)
    begin
        o_data_bf [70] = 1;
        o_data_bf [69] = 1;
        o_data_bf [68:65] = i_dst;  //de
        o_data_bf [64] = 0;
        o_data_bf [63:0] = i_data;  //da
        flit_complete = o_data_bf;
        o_data_valid = 1;
    end
end

endmodule
```

Figure 6: NoC Interface and its Pseudo Code.

For simulation, we used total 10 data sets that could represent the average execution time for both software and hardware sides. The following table is

P1 tb	P2 tb	P3 tb	P4 tb	P5 tb	P6 tb
338	145	147	345	143	142
281	114	147	228	143	144
337	148	150	344	146	145
283	146	149	290	144	145

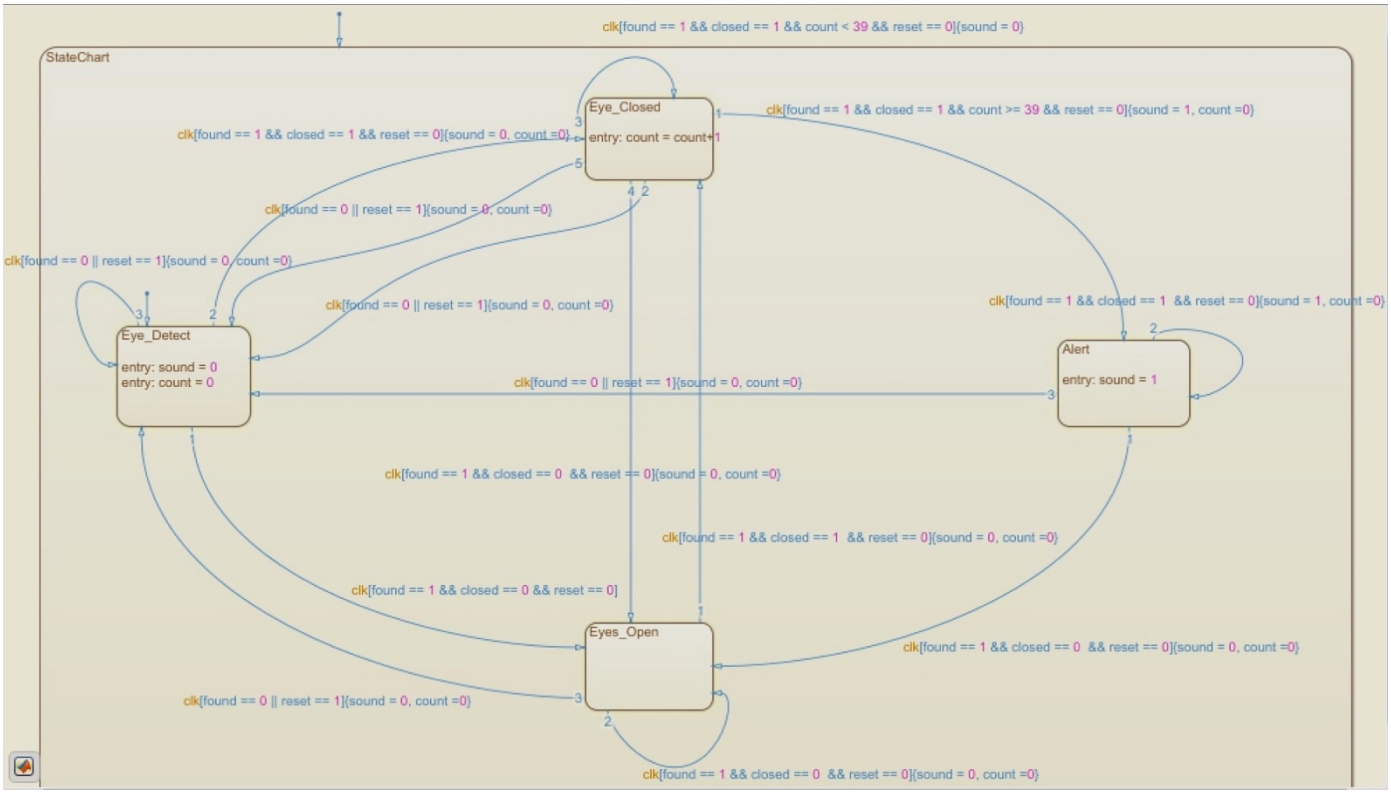


Figure 3. Eye-detection StateChar

4 Challenges

The first challenge encountered was which performance bottleneck would be the most feasible to accelerate with hardware implementation. Initially, it was determined that the real time image processing was the target bottleneck to address. However, the image processing is encapsulated within the Python OpenCV packages, which convolutes the ability to implement this aspect of the algorithm in hardware. Instead, the Eye-Aspect Ratio calculation proved to be more feasible to implement in hardware. While implementing the 3x3 mesh-based architecture using parallelization, two challenges arose. First, the EAR is based on the Euclidean distance between three pairs of eye landmarks from the eye detection algorithm.

The Euclidean distance given by:

$$(2)\ d(\boldsymbol{p},\boldsymbol{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}\ where\ \boldsymbol{P} = (p_1, p_2)\ and\ \boldsymbol{Q} = (q_1, q_2)$$

requires a node dedicated for square root operations. This requirement was circumvented by making a distance approximation such that each pair of points is in the same plane. Therefore, the distance formula is simplified to a 1D distance given by the difference of a single axis:

$$(3)\ d(\boldsymbol{p},\boldsymbol{q}) = (p_1 - q_1)$$

The percent error of using this assumption was calculated from 338 calculations of the Euclidean EAR value vs. the approximated EAR value. The data from this analysis yielded a percent error of 0.22241037% which supports the use of this assumption to approximate the eye landmark distances. Additionally, the EAR calculation required floating point division as the final computation. In order to avoid this complication, the hardware acceleration only included calculating the numerator and denominator of the EAR.

5 Novelty

The novelty of our project focuses on accelerating the Eye-Aspect Ratio (EAR) calculation. We reproduced the EAR calculations in Verilog instead of Python. In our Verilog code, we implemented a 3x3 mesh-based NoC architecture that allowed us to perform the EAR calculations in parallel. We then recorded the time it took perform the EAR calculations in software (Python) and then in hardware using our approach. As we predicted, the hardware approach was faster by two magnitudes.

6 Conclusion

We have found CNNs to be the main approach used in facial expression recognition technology. Thereby, we decided to use a Python machine learning package that utilizes CNNs to detect drowsiness/eyelid closure from a given image or set of images. As a team, our objective was to optimize the machine learning algorithm using hardware acceleration. We were successful in implementing such solution by taking advantage of our 3x3 mesh-based NoC's ability to perform parallelized operations. However, the real-world applications of our solution are limited because the added cost that our solution incurs. This project helped us get a much better understanding of convolutional neural networks and how they are used in facial recognition. We also became very familiar with OpenCV and other open source python packages that utilize deep learning and image processing to detect real time facial attributes. As a group we collectively became more apt at implementing NoC architecture and more knowledgeable about facial recognition technology.

References

[1] Wheaton AG, Chapman DP, Presley-Cantrell LR, Croft JB, Roehler DR. Drowsy driving – 19 states and the District of Columbia, 2009-2010.[630 KB] MMWR Morb Mortal Wkly Rep. 2013; 61:1033

[2] Wheaton AG, Shults RA, Chapman DP, Ford ES, Croft JB. Drowsy driving and risk behaviors—10 states and Puerto Rico, 2011-2012.[817 KB]MMWR Morb Mortal Wkly Rep. 2014; 63:557-562.

[3] Trutschel, U., Sirois, B., Sommer, D., Golz, M., & Edwards, D. (n.d.). *PERCLOS: An Alertness Measure of the Past*. University of Iowa. https://doi.org/10.17077/drivingassessment.1394

[4] *An introduction to biometric recognition - IEEE Journals & Magazine*. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6165309. [Accessed: 06-Dec-2018].

[5] *An introduction to biometric recognition - IEEE Journals & Magazine*. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=554195. [Accessed: 06-Dec-2018].

[6] M. Kostinger, P. Wohlhart, P. M. Roth, and H. Bischof, "Annotated Facial Landmarks in the Wild: A large-scale, real-world database for facial landmark localization," *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011.

[7] M. Halvorsen, "Convolutional Neural Network in Hardware," *Hardware Acceleration of Convolutional Neural Networks*, Jun. 2015.

[8] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using Haar classifiers," *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA 09*, 2009.

[9] V. Ingle and M. Gaikwad, "Review of Mesh Topology of NoC Architecture using Source Routing Algorithms," *International Journal of Computer Applications*.

[10] A. Rosebrock, "Drowsiness detection with OpenCV," *PyImageSearch*, 26-Apr-2017. [Online]. Available: https://www.pyimagesearch.com/2017/05/08/drowsiness-detection-opencv/. [Accessed: 07-Dec-2018].

[11] Albiol, A., Monzo, D., Martin, A., Sastre, J., & Albiol, A. (2008). Face recognition using HOG–EBGM. Pattern Recognition Letters, 29(10), 1537–1543. https://doi.org/10.1016/j.patrec.2008.03.01

[12] LBPH, H. (2018). Faces Recognition Using HAARCASCADE, LBPH, HOG and Linear SVM Object Detector. In Proceedings of the Sixth International Conference on Green and Human Information Technology: ICGHIT 2018 (Vol. 502, p. 232). Springer.