ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 3, March 2017

Convolution and Stochastic Pooling Algorithm

[1] Monish Katari, [2] Reethu Gali Ramesh, [3] MSA Srivatsava
[1] Department of Electrical Engineering and Computer Science,
[2] Department of Electrical Engineering, [3] Department of ECE
[1] Graduated from Texas A&M University – Kingsville, [2] Graduated from Arizona State University,
[3] PhD Scholar of ECE, JNTUCEA – JNT University - Anantapuramu

***Abstract:* -- This paper presents an efficient hardware implementation of Convolution and Stochastic pooling algorithm. The main objective of the design is to minimize the area and power, while maximizing the throughput. Performing Convolution and Stochastic pooling in CPU consumes high amount of power and low performance relative to hardware accelerator implementation. In this paper, we propose a hardware accelerator to minimize area, power and maximize throughput. We also propose architectural techniques like Interleaving and folding to improve power and area. The optimized approach which is mapped to a 28nm ASIC target demonstrated significant power and area reduction when compared to traditional model. On the other hand, the optimized approach mapped to a FPGA target has increased source utilization when compared to conventional model.**

***Keywords:* - Convolutional Neural Network (CNN), Central processing unit (CPU), Graphics processing unit (GPU)**

I. INTRODUCTION

Convolutional Neural Network (CNN), a well-known deep learning architecture extended from artificial Neural Network, has been extensively adopted in various applications, which include video surveillance, mobile robot vision, image search engine in data centers, etc. Convolution and Stochastic pooling layers are one of the mainly used layers in CNN.

Convolution is a computationally intensive function that has been widely used in image processing and pattern recognition. Also, in the recent development of deep learning and Convolutional Neural Networks (CNNs) for image applications, Convolution is most heavily used operation. In image processing, Convolution represents weighted sum computation of a source image and a Convolutional kernel. Due to the specific computation pattern of CNN, general purpose processors are not efficient for CNN implementation and can hardly meet the performance requirement. Thus, various accelerators based on ASIC and FPGA designs have been proposed recently to improve performance of CNN designs. For any CNN algorithm implementation, there are a lot of potential solutions that result in a huge design space for exploration.

General approaches for sub-sampling used Max-pooling and Average pooling operations. Max-pooling only captures the strongest activation of the filter template with the input for each region and Average pooling captures the mean activation of the filter template.

In this project, we used a sub-sampling operation called Stochastic pooling to reduce the image or feature dimensions without losing critical information. As Max-pooling and Average pooling eliminates all the activations except strongest and mean activations, there may be additional activations in the same pooling region that should be taken into account when passing information up the network and Stochastic pooling ensures that these non-maximal activations will also be utilized.

Convolution of the source image and kernel is performed by sliding 3x3 kernel onto the 4x4 image with one pixel stride horizontally and vertically. This generates four 3x3 activation maps. These activation maps are convoluted with 3x3 kernel to generate four Convolution outputs.
Stochastic pooling is applied on to these four Convolution outputs and probability of these Convolution outputs is calculated and sorted with SMC blocks. Then a random output is selected from all Convolution outputs using random generator block in SMC.

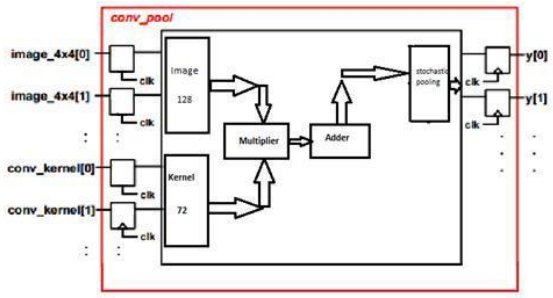

Figure 1: Hardware Implementation of Convolution and stochastic pooling

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 3, March 2017

The above figure gives a brief introduction about hardware implementation of Convolution and Stochastic pooling design. Each Input image of size 4x4 is convoluted with kernel of size 3x3 using a multiplier and all the dot products are summed up using adder. The outputs of the adder are then given to Stochastic pooling block which then selects random output. The inputs and outputs are synchronized using registers.

A. Why a hardware accelerator is needed for this algorithm?

For this algorithm to process on a general purpose CPU or GPU becomes very computational intensive with high power being consumed. For various applications it needs to send preprocessed data to the cloud to allow computation to complete and send post processed data back which causes increase in latency using the internet.
The implementation of a Hardware Accelerator enables computation to be done locally, therefore reduces latency which is essential for some object recognitions application e.g. autonomous cars.

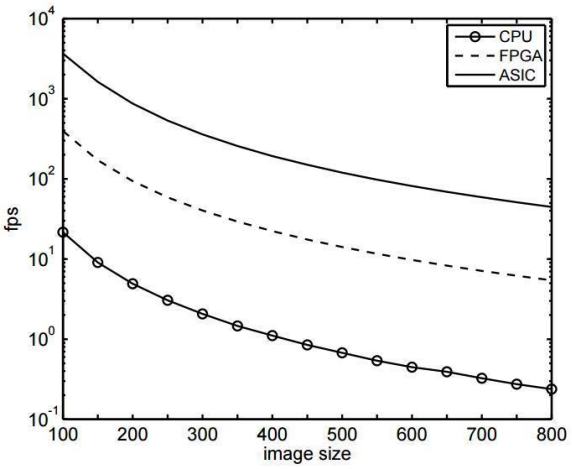

Figure 2: Image size vs fps for various hardware implementations

Previous research has been done comparing algorithm implementation on different target specifications. Figure 1 show how the frames per second are affected with different hardware implementations relative to image size.

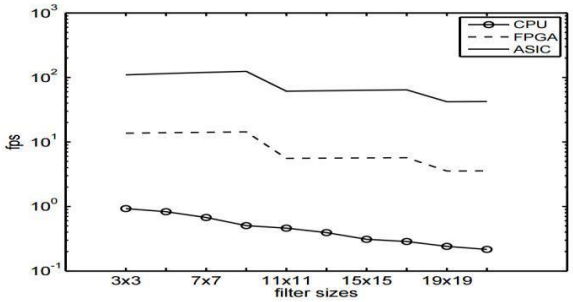

Figure 3.Kernel size vs fps for various hardware implementations

Previous research also shows that depending on the filter size effects the fps for different target specifications.
Figure 2 and Figure 3 show results comparing CPU, ASIC and FPGA. From this research we determined that ASIC implementation would be the best choice to decrease power and area while maximizing throughput.

II. DESIGN SPECIFICATIONS

The optimized architecture aims at reducing the number of processing elements when compared to the conventional model which greatly reduces total area and power consumption upon mapping to 28nm ASIC target. Our conventional model is a pipelined design, which increased throughput and increased area and power consumption. Optimized approach when mapped to a FPGA target witnessed 20 % reduction in the usage of I/O ports, LUTs and Non I/O registered bits.

III. ALGORITHM DESIGN

Essentially, the target input image is a grey scale, 16x16 pixel which will be sub-divided into sixteen 4x4 images for the target hardware architecture in this project. Each pixel of the input image is represented by an 8-bit positive integer whose values range from 0 to 255. The sharpening kernel is a 3x3 array of integer value. Each pixel s represented by 8 bit two's complement integer whose values range from -128 to 127. We designed a Convolution and Stochastic pooling module to perform the Convolution task between each 4x4 input source image and 3x3 sharpening kernel. Basically, The 3x3 kernel slides on the image with one pixel stride horizontally and vertically and generates four activation maps of size 3x3x1. A Convolution layer is applied on these activation maps and the Output is generated.

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 3, March 2017

$$c0 = \left(\sum_{i=0}^{2} \sum_{j=0}^{2} [x(i,j) * k(i,j)] \right) \quad , \quad c1 = \left(\sum_{i=0}^{2} \sum_{j=0}^{2} [x(i,j+1) * k(i,j)] \right)$$

$$c2 = \left(\sum_{i=0}^{2} \sum_{j=0}^{2} [x(i+1,j) * k(i,j)] \right) \quad , \quad c3 = \left(\sum_{i=0}^{2} \sum_{j=0}^{2} [x(i+1,j+1) * k(i,j)] \right)$$

The Convolution part is implemented in hardware using the above mathematical formulae. In our design, each Convolution and Stochastic pooling between one 4x4 array and one 3x3 kernel can be processed using one processing element (PE). Our system includes 16 PE blocks which allow the module to process all sixteen 4x4 image arrays in parallel. Figure 4 below illustrates the structure of one PE. The following section will describe how each components of one PE performs the Convolution and the pooling.

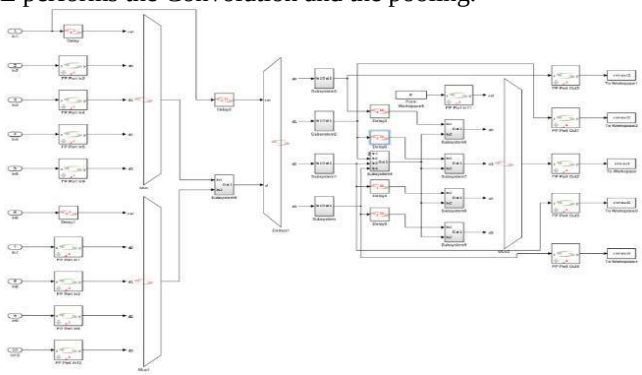

Figure 4: Structure of one PE

In order to perform the Convolution task, four activation maps of size 3x3 and the sharpening kernel of 3x3 are converted into five one-dimension array of 1x9 elements (pixels) using SMC Reshape blocks. Two 4x1 Multiplexers are utilized to sample the data from these five above mentioned 1x9 arrays.

The first multiplexer intakes data from the four activation map of 1x9 arrays at each input port to select one pixel at each sample time. On the other hand, the same sharpening kernel 1x9 vector are repeatedly fed into each of the four input ports of the second Multiplexer. The Select lines of the two multiplexers are controlled using up Counters. These Counters are designed in such a way that allows the two Multiplexers to sample each corresponding pixel of the kernel and the input image at the same time.

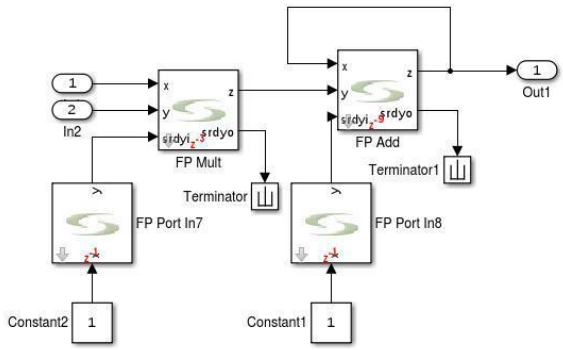

Figure 5: Convolution subsystem

In figure 5 above, the Convolution subsystem, which comprises of a Floating Point Multiplier and Floating Point Adder, is utilized to process pairs of corresponding pixels. Basically, the computation of the dot product of all the corresponding pixels between the input image vector and the kernel vector can be performed by the Floating Point multiplier. These dot product results are accumulated by recursively using a Floating Point Adder through a feedback loop to determine the total value, which is the desired Convolution output result. Figure 6 below illustrates the process of computing a Convolution output value.

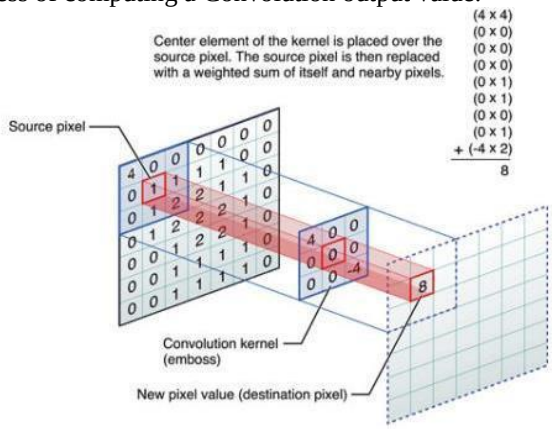

Figure 6: Convolution of Image and Kernel

Figure 7 shows the convoluted output when an Image is convoluted with sharpening kernel

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 3, March 2017

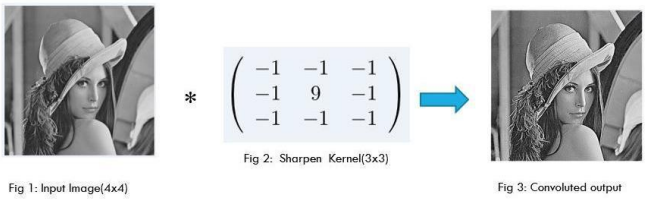

Figure 7: Hardware Implementation of Convolution and Stochastic pooling

After determining the Convolution output value, the output of the adder block are also connected to the input of a 1-to-4 De-multiplexer block in order to collect the result. Each of the outputs of the De-multiplexer is connected to a SMC sample and hold block (shown in figure 8 below) to store the result of the Convolution. Since the sample and hold block allow the output to be selected when the enable signal is set to high. We use a comparator to control the enable signal of the sample and hold unit. Due to the fact that each of the four results is arriving in sequential order with one period difference, the timing of the 4 sample and hold blocks can be determine to sample the correct result.

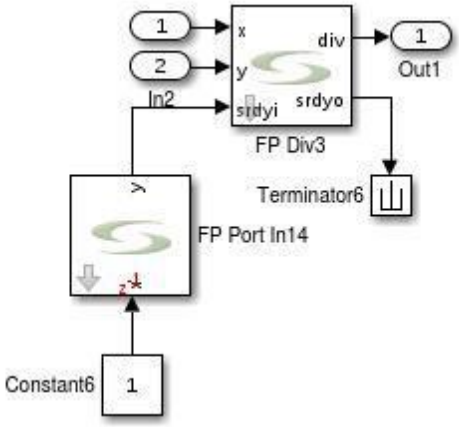

Figure 8: Sample and hold block

Basically, one input of the comparator is set to be the number of cycles it would take to sample the final result of one Convolution while the other input is controlled by a counter block to count up to that same number of cycles of sampling. When the two inputs of the comparator are equal, the output of the comparator makes a transition from 0 to 1. Consequently, this comparator output signal makes the enable signal change from 0 to 1, allowing the Convolution output result to be store in the sample and hold block.

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

To start the Stochastic pooling process, we first find the sum of the four Convolution results then take the quotient between each Convolution result and the sum of the four results as shown in the formula above. The sum of the four convolution results stored in the sample and holds blocks is computed using the three adders in the subsystem (shown in figure 9 below). After this, we use the divider block to calculate the quotient between each convolution output and the Sum to determine the probability of each convolution output

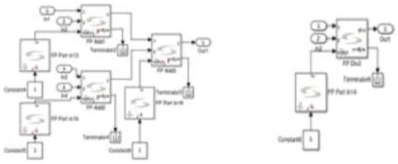

Figure 9: Subsystem to find probability

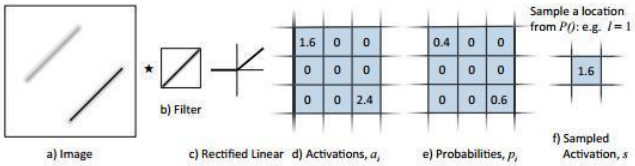

Figure 10: Stochastic pooling

We then sample from the multinomial distribution based on p to pick a location l within the region

$$s_j = a_l \quad \text{where } l \sim P(p_1, \ldots, p_{|R_j|})$$

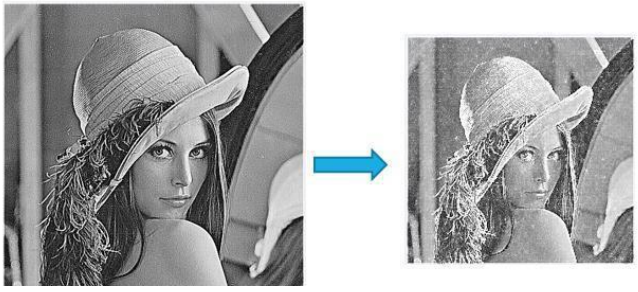

Figure 11: Stochastic pooling output Image reconstruction

Figure 11 above shows the output of Stochastic pooling which is generated by sub-sampling convoluted image.

A. Optimization

Initially, the design of one PE is consisted of 4 multipliers and does not require any multiplexer to sample

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 3, March 2017

the data from the input image and the sharpening kernel. However, since the parallel computing of the Convolution part is limited by the feedback loop of the adder to accumulate the results of all output multiplier. We decided to use Multiplexers to sample the input data using Interleaving technique and implementing the parallel computing by using 16 PEs in parallel for processing 64 (16x4 3x3) activation maps.

For each PE, an interleaving factor of 4 is applied to process all activation maps. In order to perform correct accumulation in the feedback loop, each PE has to store four intermediate dot products in the feedback loop. As a result, we can have at least four registers to add delay of 4 units in the feedback loop. However, since the Floating Point Adder comes with an internal delay of 9 units, this factor sets the minimum latency of the loop to 9 cycles.

In this architecture, for an interleaving factor of 4, we have to waste 5 cycles to feed in the second element of first activation map. By interleaving, we reduce the number of multipliers used in PEs. As a result, the amount hardware resources required to perform Convolution and Stochastic pooling design is reduced and this reduced area drastically. We have not added any additional feedback registers to balance the latency as the adder has in-built delay of 9 units. Technically, this is the best design where we can achieve best power and area.

We cannot further increase the interleaving factor more than 4 units and reduce the number of PEs as Stochastic pooling operation is performed only on four Convolution outputs. Along with interleaving, we also implemented folding architectural technique by feeding four activation maps into a multiplexer at a time in every PE rather than using multiple PEs which greatly reduced the area by decreasing the number of PEs. As it is highly pipelined design, throughput of the design is approximately the same as the initial model.

Even though we have a recursive algorithm, we can pipelining before recursive adder by adding registers. Without any pipelining in the design, clock frequency is relatively slow, while the whole Convolution and Stochastic pooling design took one cycle. With pipelining, clock frequency is relatively fast, and number of Convolution and Stochastic pooling operations happen in flight, increasing the throughput. However, area of the design is increased due to addition of hardware. This additional overhead increases power consumption of the design.

IV. IMPLEMENTATION RESULTS

As we discussed earlier, we implemented convolution and stochastic pooling in parallelism and interleaving architectures and mapped them to 28nm ASIC and FPGA targets. Each design has its own advantages and disadvantages. Finding a sweet spot that reduces area, power consumption and does not degrade throughput is very critical.

A. Asic Implementation

Following table gives a brief idea about Area and Power comparisons between optimized and conventional models.

Design of Experiment	Optimized Design and Comparison in ASIC		
	Optimization Technique	*Total Area(um²)*	*Power(uW)*
Initial	Parallelism	1484742.7	4.12773
Final	Interleaving	1065228.6	3.7465

Table 1: Represents area and power consumption of various techniques

As we observe from the following figure total area and cell area is the least in the case of interleaved & folded design whereas area is increased in parallelism architecture.

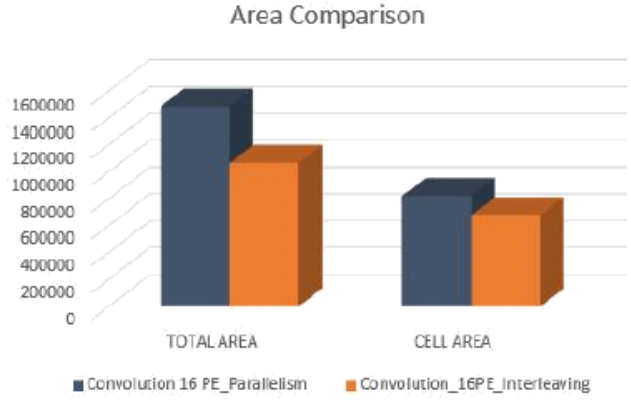

Figure 12: Comparison of area between parallelism and Interleaving

ISSN (Online) 2394-6849

International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)
Vol 4, Issue 3, March 2017

Figure 13: Comparison of Power between parallelism and Interleaving

In the above figure, power is greatly reduced when the design is interleaved and power consumption is high in the case of parallelism design

B. FPGA Implementation

FPGA mapping was done using Synplify Pro to evaluate the resource utilization. The FPGA Target used is Virtex-7 FPGA XC7VX485T-2FFG1761.

No.	Conventional (Parallelism)			Optimized (Interleaving & Folding)		
Reso Urce	*LUTs*	*DSP48 E*	*I/O ports*	*LUTs*	*DSP48E*	*I/O ports*
1	54089	192	16610	3202 6	144	10242

Table2: Represents resource utilization of various techniques

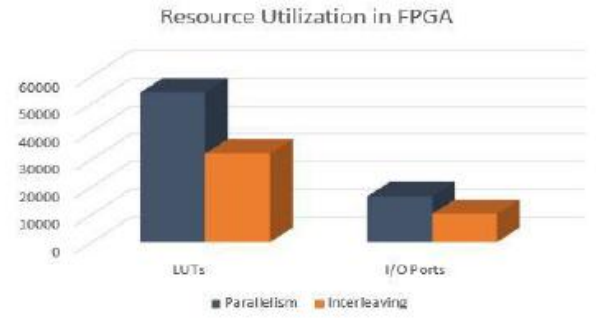

Figure 14: Comparison of resource utilization between various techniques

V. CONCLUSION

In this work, we produced efficient designs using optimization techniques to study the trade-offs between throughput, area and power. Initially, we created a design that utilizes parallel computing to target the high throughput. However, this design increased area and power consumption. From that throughput baseline, we optimized the design to reduce area and energy consumption by using interleaving technique in sampling the input data and also by using the folding technique to reuse the same FP Multiplier in one processing element. We realized that the final design has nearly the same throughput (1 cycle delay difference) but consumes 20-30 % time lower amount of energy and requires 20 % times, 40 % times lower amount of resources and area, respectively. Finally, we conclude that following Interleaving and Folding techniques are best techniques that can be used to achieve less area and minimum power while achieving decent throughput for this target design.

REFERENCES

[1] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stagearchitecture for object recognition?" in Proc. International Conference on Computer Vision(ICCV'09). IEEE, 2009.

[2] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching forrecognizing natural scene categories," in CVPR '06: Proceedings of the 2006 IEEE ComputerSociety Conference on Computer Vision and Pattern Recognition. Washington, DC, USA: IEEEComputer Society, 2006, pp. 2169–2178.

[3] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition withcortex-like mechanisms," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.29, no. 3, pp. 411–426, 2007.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient- basedlearning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, November 1998.
http://cadlab.cs.ucla.edu/~cong/slides/fpga2015_chen.pdf2.

[5] http://e-lab.github.io/data/papers/iscas2010CVP.pdf

[6] http://people.idsia.ch/~juergen/ijcai2011.pdf

[7] http://yann.lecun.com/exdb/publis/pdf/farabet-iscas-10.pdf

[8] S. Arya and D. M. Mount, "Algorithms for fast vector quantization," in Proc. DCC 93: Data Compression Conf., J. A. Storer and M. Cohn, Eds. Piscataway, NJ: IEEE Press, pp. 381– 390.

[9] H.-U. Bauer and K. R. Pawelzik, "Quantifying the neighborhood preservation of self-organizing feature maps," IEEE Trans. Neural Networks, vol. 3, pp. 570–579, 1992.

[10] Y. Bengio, Y. Le Cun, and D. Henderson, "Globally trained handwritten word recognizer using spatial representation, space displacement neural networks, and hidden Marko models," in Advances in Neutral Information Processing Systems 6. San Mateo, CA: Morgan Kaufmann, 1994.

[11] J. L. Blue, G. T. Candela, P. J. Grother, R. Chellappa, and C. L. Wilson, "Evaluation of pattern classifiers for fingerprint and OCR applications," Pattern Recognition, vol. 27, no. 4, pp.485-501, Apr. 1994

[12] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. Jackel, Y. Le Cun, U. Muller, E. Sackinger, P. Simard, and V. N. Vapnik, "Comparison of classifier methods: A case study in handwritten digit recognition," in Proc. Int. Conf. Pattern Recognition. Los Alamitos, CA: IEEE Comput. Soc. Press, 1994.

[13] D. K. Burton, "Text-dependent speaker verification using vector quantization source coding," IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-35, pp. 133, 1987.

[14] R. Chellappa, C. L. Wilson, and S. Sirohey, "Human and machine recognition of faces: A survey," Proc. IEEE, vol. 83, pp. 5, pp. 705–740, 1995.

[15] I. J. Cox, J. Ghosn, and P. N. Yianilos, "Feature-based face recognition using mixture-distance," in Computer Vision and Pattern Recognition. Piscataway, NJ: IEEE Press, 1996.

[16] R. Brunelli and T. Poggio, "Face recognition: Features versus templates," IEEE Trans. Pattern Anal. Machine Intell., vol. 15, pp. 1042–1052, Oct. 1993.