Parameter Embedding for Efficient FPGA Implementation of Binarized Neural Networks

Reina SUGIMOTO Nagisa ISHIURA

School of Science and Technology, Kwansei Gakuin University
2-1 Gakuen, Sanda, Hyogo, 669–1337, Japan

Abstract—A binarized neural network (BNN), a restricted type of neural network where weights and activations are binary, enables compact hardware implementation. While the existing architectures for BNN assume that weights and biases are stored in on-chip RAMs, this paper presents an attempt to embed those parameters into processing elements by utilizing LUTs in FPGAs as ROMs. This eliminates the bandwidth limitation between memories and neuron PEs and allows higher parallelism, as well as it reduces the hardware cost of the neuron PEs. This paper also proposes a map-shift scheme to efficiently supply the neuron PEs with feature map data for convolution. As a case study, LeNet5 has been implemented based on this method targeting Xilinx FPGA Artix-7, which can process a frame in 1,386 cycles at 21.1MHz.

I. Introduction

Recent advances in neural network technology have brought great leaps in various areas such as computer vision, data classification, natural language processing, etc. Following the success in these areas, deep neural networks are now widening the scope of applications.

One of the key trends in the neural net technology is the shift from cloud to edge computing. With smaller delay, reduced communication cost, and better data security, inference on edge devices would further expand application areas of machine learning. However, expensive computation cost for deep neural networks is a burden for edge devices with limited computing resources and low power budget. Hardware acceleration of neural net inference is thus an important research topic.

A binarized neural network (BNN) is a restricted type of neural network where weights and activations are binary [1]. It drastically reduces the computation cost, hardware size, and power consumption. Many efficient architectures to implement BNNs in hardware have been proposed so far [2, 3].

One of the major bottlenecks in neural network computation is access to memories that store huge number of parameters. Although binarization substantially reduces the data amount and enables the use of on-chip RAMs, the number of the memory banks limits the parallelism of computation at neurons.

A natural idea to counter this problem is to migrate the parameters from memories to neuron processors. Chi proposed to implement a fully connected layer of neural network as a combinational logic circuit [4]. Since the parameters turn to constants, the resulting circuit is simplified. The circuit size is further reduced by sharing partial results. However, this method is effective only for fully connected or dense layers, not for convolution layers. Moreover, the resulting combinational circuit is still too large.

This paper presents a method for implementing hardware BNNs without memory access which is applicable to both convolution and fully connected layers. It is based on parameter embedding in which the weight and bias parameters are embedded in neuron processor elements [5]. This enables highly parallel computation because the parallelism is no more limited by the number of memory banks. As a method of supplying feature map data at a rate that meet the parallelism, a map shifting scheme is also proposed in this paper. As a case study, whole layers of LeNet5 are implemented targeting Xilinx FPGA Artix-7 which can process a frame in 1,386 cycles at 21.1MHz.

II. Binarized Neural Network (BNN)

For a neuron i in a neural network, let I_i be a set of neurons feeding i, b_i be the bias of i, $w_{i,j}$ be the weight associated with the synapse connecting neurons j and i. The activation value a_i of i is expressed as $a_i = f(b_i + \sum_{j \in I_i} w_{i,j} \cdot a_j)$. In the binarized neural network, a_i and $w_{i,j}$ are in $\{-1, +1\}$, and $f(x) = 1$ if $0 \leq x$ and $f(x) = -1$ otherwise. If we represent -1 and $+1$ by 0 and 1, respectively, the multiplication in the above formula is reduced to an exclusive-nor operation.

Since a neural network consists of a huge number of neurons, it is impractical to accommodate all the neurons in a single chip. Thus, processing elements (PEs) for neuron are usually repeatedly utilized and the parameters (the weights and the biases) are stored in RAMs. Binarization substantially reduces the amount of the parameters so that all the parameters for a BNN may be accommodated in on-chip RAMs, which drastically enhances the memory access speed. Even so, the parallelism of computation of a BNN is limited, because it is limited by the number of the banks of the on-chip RAMs.

III. Parameter Embedding and Map Shifting

A. Parameter Embedding

Instead of storing the weight and bias parameters in RAMs, we propose to embed them in neuron PEs.

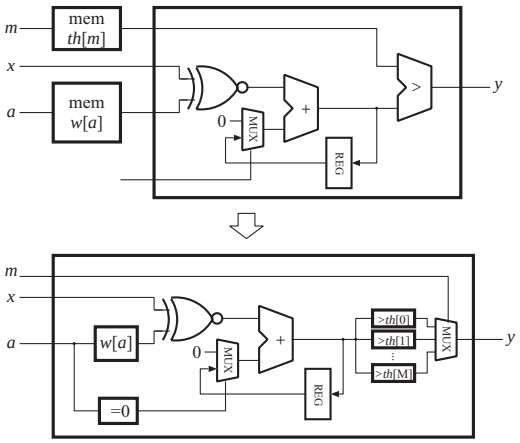


Fig. 1. Parameter embedding

Fig. 1 illustrates the basic idea. We assume that the neuron PE processes inputs sequentially; the input data come through input x and multiplied by the corresponding weights $w[a]$, and their sums are compared by threshold $th[m]$. a and m are addresses for the parameters and when the neuron has k inputs, $m = a/k$. Instead of supplying the parameters from memories, we embed them into the PEs. For a k input neuron, w parameters takes only k bits. In the case of FPGA with memory type LUTs, up to 32 parameters may be accommodated in a 5-input LUT. Moreover, parameter embedding may induce logic simplification. For example, comparison with variable thresholds are reduced to comparison with constants, which may decrease the hardware cost.

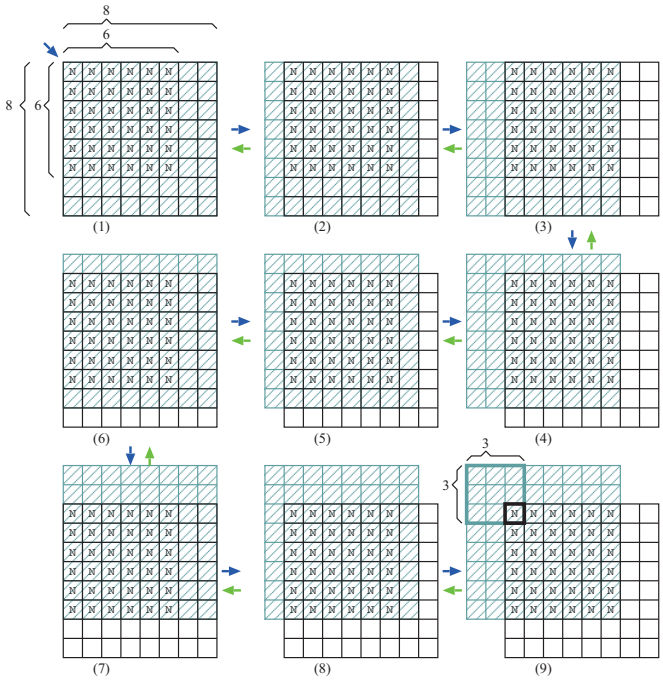
The biggest impact of parameter embedding is on freedom for the overall architecture of BNNs. The number of the PEs working in parallel are not limited by the number of the memory banks. There is no worry about routing many parameters between memories and PEs.

B. *Map Shifting*

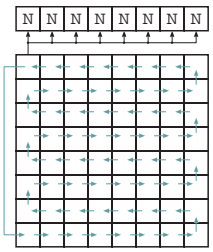
As an idea to implement a highly parallel convolution neural network circuit, a map shifting scheme is proposed in this paper.

Fig. 2 (a) illustrates the idea for 2-dimensional convolution with a 8×8 feature map with kernel size 3×3 which yields a 6×6 new map. In our scheme, 6×6 PEs (indicated by 'N' in the figure) computes the output (convolution) in parallel. At first ((1) in the figure), the PE at position (i, j) reads from the cell (i, j) of the map. Then, the map data are left shifted (2), so that the PE (i, j) reads the cell $(i, j+1)$. After another left-shift (3), the map is shifted upwards (4), then the map is right shifted twice (5–6). This zigzag shifts are repeated so that the PE (i, j) scans the cells (i, j) through $(i+2, j+2)$. When multiple maps have to be processed, the sequence of shifts is carried out in the reverse way (as illustrated by the green arrows). The data shifted out of the map may either be rotated to the vacated cells or accommodated in extra cells. The same scheme may be applied to one dimensional or more than two dimensional convolution.

Fig. 2 (b) shows a map shifting scheme for a fully connected



(a) For convolution



(b) For full connection

Fig. 2. Map shifting scheme

layer, which is simpler than that for convolution. All the map data are chained and shifted at each cycle. Then, all the PEs can read all the cell data sequentially at some cell (the left-top cell in this figure).

IV. Case Study: Implementation of LeNet5

As a case study, whole layers of LeNet5 [6] are designed, based on the idea presented in the previous section. As shown in Fig. 3, the net takes a 32×32 image map of a handwritten digit and outputs the one-hot code of the recognition result. We assume all the feature map data including those of the input image are binary.

Note that our implementation is specialized for a given neural network, so the design must be autogenerated every time a neural network configuration and a set of parameters are given.

A. *Layer 0 and 1*

Fig. 4 shows structures of the circuits for the layer-0 and layer-1 modules and the connection among them. The layer-0

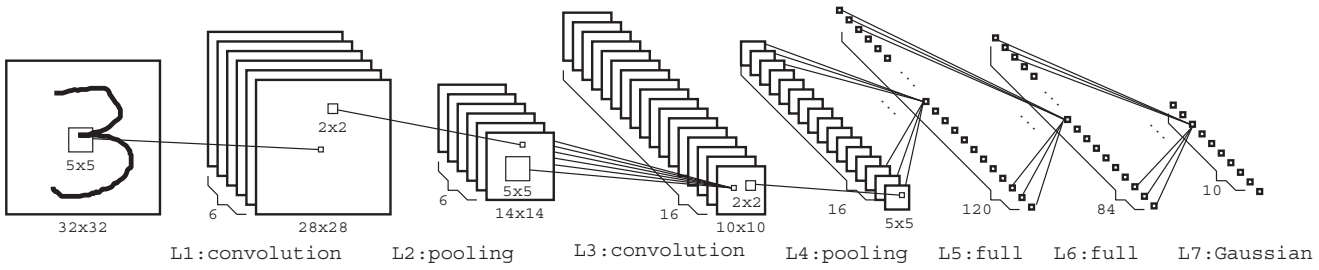


Fig. 3. LeNet5 [6]

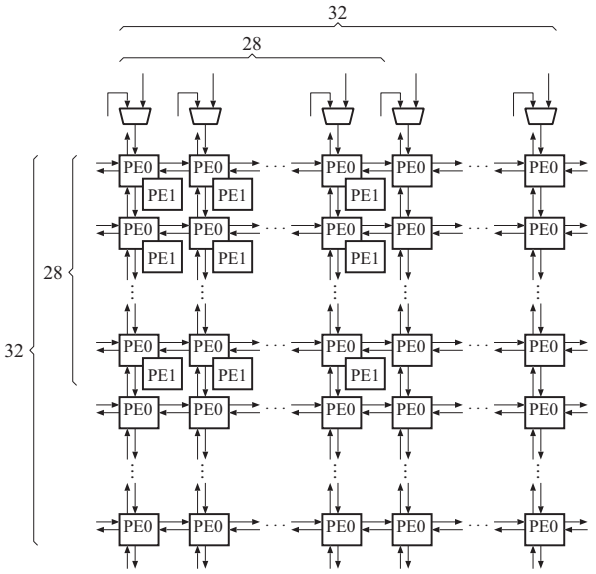


Fig. 4. Layer0 and Layer1

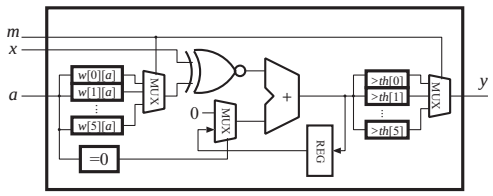


Fig. 5. PE1

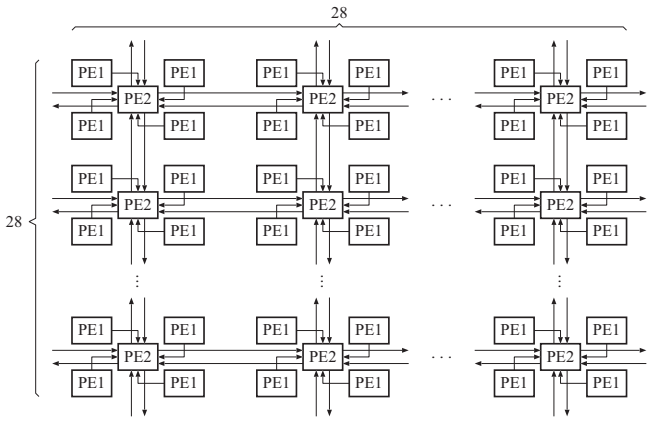


Fig. 6. Layer1 and Layer2

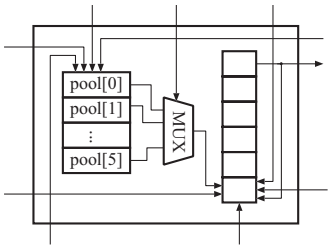


Fig. 7. PE2

module loads the input image map and feeds them to the layer-1 module which computes convolution.

The layer-0 module is a 32×32 two-dimensional rotate-shift register. The image map is loaded from the top side, 32 bits at a time taking 32 cycles. Then the map is shifted as was illustrated in Fig. 2 (a) to feed the layer-1 module.

The layer-1 module consists 28×28 neuron PEs (PE1s) to compute convolution of kernel size 5×5. The structure of PE1 is as illustrated in Fig. 5. Since the layer-1 computes convolution 6 times, PE1 has 6 sets of weight parameters ($w[0][a]$ through w[5][a]) and threshold modules (for $th[0]$ through $th[5]$), where $m = 0$ through 5, and $a = 0$ through 5×5.

After the layer-0 is loaded in 32 cycles, it takes $5 \times 5 \times 6 = 150$ cycles to produce 6 maps.

B. Layer 2

The layer-2 module computes pooling (4 to 1 sub sampling) of the output from the layer-1 module. The connection between the layer-1 and layer-2 modules are as shown in Fig. 6. The layer-2 module consists of 14×14 PE2s, each of which is connected with 4 adjacent PE1s. The layer-2 module also keeps the results of pooling and feeds them to the next convolution layer, so it constitutes a two dimensional rotate shift register.

The structure of PE2 is as shown in Fig. 7. It has six function modules for pooling and six 1-bit registers to store the results. Note that in parameter embedding scheme any pooling computation is represented as a 4-input 1-output Boolean function, which is implemented with a single 4-input LUT. Every time pooling is computed, the result is shifted into the registers.

It takes 1×6 cycles to compute pooling in the layer-2 module. However, the computation can be overlapped with the convolution in the layer-1 module, so virtually it takes no cycle.

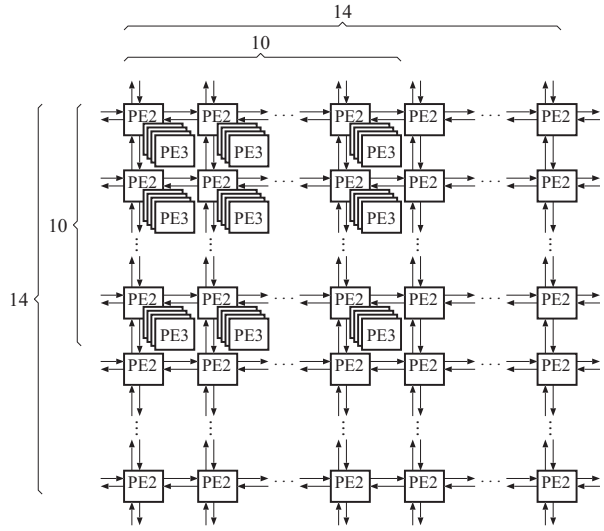


Fig. 8. Layer2 and Layer3

Fig. 9. Shift in Layer2

C. Layer 3

The Layer-3 module computes convolution for the six 14×14 maps of kernel size 5×5 to produce 16 output maps of size 10×10. 16 maps are computed by 4 sets of processors; namely, 4 results are computed at a time which is repeated for 4 times. Thus, the layer-3 module consists of $10 \times 10 \times 4$ PE3s. The structure of PE3 is the same as PE1 except that it needs $5 \times 5 \times 6$ weight parameters for each iteration.

The map data are supplied from layer-2 to layer-3 in the similar way as illustrated in Fig. 2 (a), except that data of 6 maps must be moved during zigzag shifts. This is done by shifting data within PE2 and between PE2s as shown in Fig. 9; 6 bits in each PE2 is shifted (upwards) with a bit shifted in from the adjacent PE2 and a bit shifted out to the PE2 on the opposite side.

To compute all the convolution, the layer-3 module takes $5 \times 5 \times 6 \times 4 = 600$ cycles with $10 \times 10 \times 4 = 400$ PE3s.

D. Layer 4

The layer-4 module performs pooling of data from the layer-3 and feeding of the resulting data to the layer-5. The connection between the layer-3 and layer-4 is similar to that between the layer-1 and layer-2, but 4 sets of data are passed at a time.

Fig. 10 shows the structure of PE4. It has 4×4 function units for pooling and 16 1-bit registers to store the results. Every time 4 pooling results are computed, the registers are shifted by 4 bits to store the new data.

Pooling takes 1 cycle per output map, but it is overlapped with the convolution in the layer-3 module.

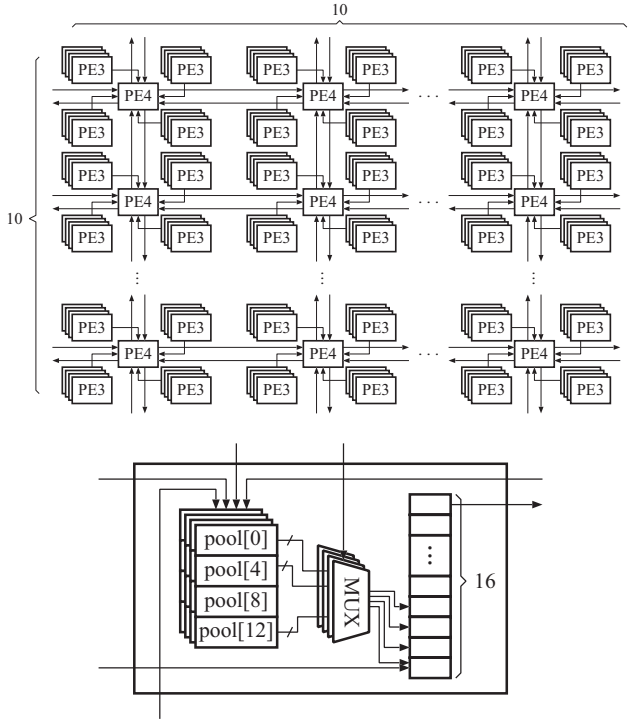


Fig. 10. PE4

E. Layer 5 throught Layer 7

The layer-5 ,-6, and -7 modules perform neural network computation of full conection, which are 5×5-input/120-output, 120-input/84-output, and 84-input/10-output, respectively.

Fig. 11 shows the connections between the layers 4 through 7. All the 5×5 registers in the layer 4 are chained and 1-bit per cycle is supplied to PE5s. Similarly, the 120 registers in the layer 5 are chained to feed 1-bit per cycle to the PEs in layer 6.

The layer-5, -6, and -7 modules take $5 \times 5 \times 16 = 400$ cycles, 120 cycles, and 84 cycles, respectively, for their computation.

V. Preliminary Result

A full-hardware LeNet5 described in the previous section has been designed in Verilog HDL, where the weights and biases were determined randomly.

The number of cycles are summarized in Table I. It takes only 1,386 cycles. Moreover, layers 0–2, 3–4, and 5–7 are computed with independent hardware, so they can form a 3-level pipeline. In this case, the iteration interval will be 604 cycles.

The result of logic synthesis targeting Xilinx FPGA Artix-7 (xc7a100tcsg324-3) by Vivado (2016.4) is summarized in Table II. The circuit may fit in a low-end FPGA chip. Since combinational part of computation in our architecture is very light, it is considered that the large critical path delay (47.3ns) is due to routing. It takes $47.3\text{ns} \times 1,386 \simeq 65.6\mu\text{s}$ to process a single frame. If executed in the pipeline manner, the iteration interval is $47.3\text{ns} \times 604 \simeq 28.6\mu\text{s}$, which translates to 35.0 kfps.

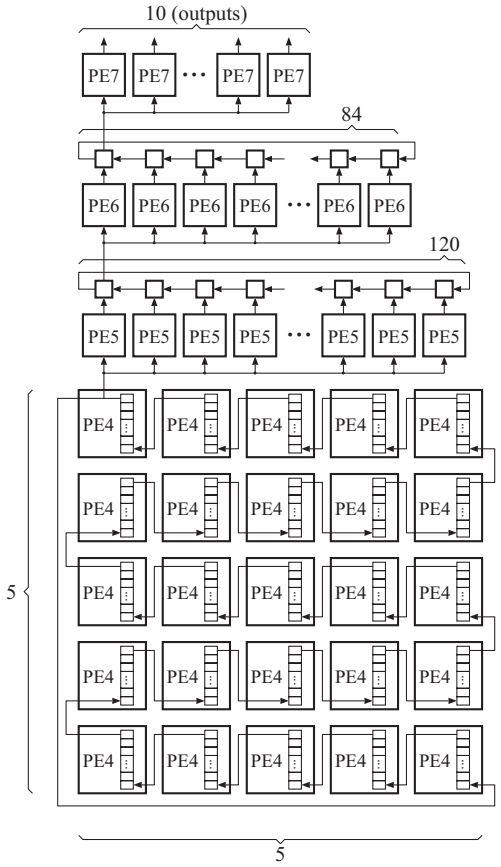


Fig. 11. Layer4 through Layer7

TABLE I
EXECUTION CYCLES

	Layer	#cycle
0	loading	32
1	convolution	150
2	pooling	(0)
3	convolution	600
4	pooling	(0)
5	full connection	400
6	full connection	120
7	gaussian connection	84
	total	1,386

VI. CONCLUSION

A new efficient scheme for FPGA implementation of BNNs is presented, which is based on parameter embedded and map shifting. With this scheme, LeNet5 has been designed which can process a frame in 1,386 cycles with reasonable amount of hardware.

At this point, we have only studied the feasibility of this scheme and a lot of issues are left. In our experiment, all the weights and biases were randomly determined. These parameters do not affect execution cycles but circuit size and the delay, so we must synthesize the circuit with actual learned parameters. We also assume all the map data including the input image data are in binary, by which a sufficient recognition rate may not be achieved. We must refine the precision for the data. Moreover, there is a lot of room for improving performance and reducing the cost of the circuit. We are now working on these issues.

TABLE II
SYNTHESIS RESULT

	#LUT	#FF	delay [ns]
overall	38,151	10,911	47.3
Ctrl	146	27	
L0	2,286	1,024	
L1	13,453	3,920	
L2	1,137	397	
L3	15,517	3,200	
L4	699	400	
L5	3,487	1,201	
L6	1,181	672	
L7	245	70	

Synthesizer: Xilinx Vivado (2016.4)
Target: Xilinx Artix-7

Acknowledgments

Authors would like to express their appreciation to Dr. Hiroyuki Kanbara of ASTEM/RI, Prof. Hiroyuki Tomiyama of Ritsumeikan University, and Mr. Takayuki Nakatani (formerly with Ritsumeikan University) for their discussion and valuable comments. We would also like to thank to the members of Ishiura Lab. of Kwansei Gakuin University. This work has been driven partly in response to support of KIOXIA Corporation (former Toshiba Memory Corporation).

REFERENCES

[1] M. Courbariaux, et al.: "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," Computer Research Repository (Mar. 2016).

[2] K. Ando, et al.: "BRein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W," *IEEE J. Solid-State Circuits*, vol. 53, no. 4, pp. 983–994 (Apr. 2018).

[3] H. Nakahara, H. Yonekawa, T. Sasao, H. Iwamoto, and M. Motomura: "A memory-based realization of a binarized deep convolutional neural network," in *Proc. FPT 2016*, pp. 273–276 (Dec. 2016).

[4] Chia-Chih Chi and Jie-Hong R. Jiang: "Logic Synthesis of Binarized Neural Networks for Efficient Circuit Implementation," in *Proc. ICCAD '18*, (Nov. 2018).

[5] R. Sugimoto and N. Ishiura: "Parameter Embedding for FPGA Implementation of Binarized Neural Networks," in *Proc. IEICE Society Conf. 2018*, A-6-1, (Sept. 2018).

[6] Y. LeCun, et al.: "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324 (Nov. 1998).