Associative Memory based on Clustered Neural Networks: Improved Model and Architecture for Oriented Edge Detection

Robin Danilo*, Hugues Wouafo*, Cyrille Chavet*, Vincent Gripon†, Laura Conde-Canencia*, Philippe Coussy*
* Lab-STICC, Université de Bretagne-Sud, Morbihan, France
† Electronics Department, Télécom Bretagne, Brest, France

***Abstract*—Associative Memories (AM) are storage devices that allow addressing content from part of it, in opposition of classical index-based memories. This property makes them promising candidates for various search challenges including pattern detection in images. Clustered based Neural Networks (CbNN) allow efficient design of AM by providing fast pattern retrieval, especially when implemented in hardware. In particular, they can be used to store and next quickly identify oriented edges in images. However, current models of CbNN only provide good performances when facing erasures in the inputs. This paper introduces several improvements to the CbNN model in order to cope with intrusion and additive noises. Namely, we change the initialization of neurons to account for precise information depending on Euclidean distance. We also update the activation rules accordingly, resulting in an efficient handling of various types of input noise. To complete this paper, associated hardware architectures are presented along with the proposed computation models and those are compared with the existing CbNN implementation. Synthesis results show that among them, several divide the cost of that implementation by 3 while increasing the maximal frequency by 25%.**

I. Introduction

Oriented Edge Detection (OED) is often used as the first step of image processing applications such as image classification or image segmentation [1]. Traditional methods use a bank of filters in which each filter is used to detect a type of oriented edges [2]. These filters, stored in an indexed memory, are applied separately on patches extracted from the input image.

Recently, the authors in [3] proposed to use an associative memory to detect oriented edges of several intensities. They also present a fully-parallel hardware implementation for that model. Unlike classical indexed memories where an explicit address is required to access a data, associative memories are capable to efficiently retrieve those data from part of them, even with noisy inputs [4]. This capability makes associative memories suitable for pattern recognition tasks like OED. In OED, a set of patterns representing the oriented edges that have to be detected in the image (equivalent to a bank of filters) is stored in an associative memory. During the detection step, patches are extracted from the input image and provided to the associative memory which is responsible for detecting the closest pattern. In this context, a patch is considered as a possible noisy version of one of the patterns stored in the associative memory. The associative memory is based on the Clustered based Neural Network (CbNN) [5], [6], [7] and it was shown that CbNN have a higher capacity compared to classical models like Hopfield neural networks and Restricted Boltzmann Machines.

In that same paper, the OED operation considers a particular type of noise named *erasure* in which some pixels of the original stored patterns might be missing from the patches. In this paper, we enrich the decoding algorithm of CbNN allowing to handle more complex types of noise. In particular, we add more precise information to each neuron that takes into account the euclidean distance.

An architecture was also proposed in that paper. However, that architecture is not optimized in terms of available resources utilization. Therefore, we also propose more optimized hardware implementations for both the original and the modified models. We study the impact of the proposed modifications on those architectures.

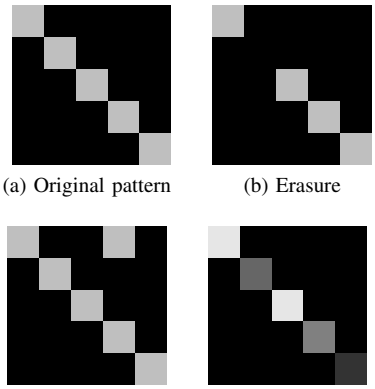
The paper is organized as follows. In the second section, the original model and its associated architecture are introduced. The third section presents the improved models to handle the new types of noise. The fourth section details the architectures proposed for the modified models and shows the results of their evaluations. The fifth section evaluates the different models against the different types of noise and performs a comparison with the classical euclidean distance. Finally, the conclusion ends this paper.

II. Oriented Edge Detection Using CbNN-based Associative Memory

In this section, we introduce oriented edge detection using associative memories, Sparse-CbNN and its associated hardware architecture.

A. Oriented Edge Detection

Let us consider an example where input image pixels can take 256 levels of gray. Before the use of an associative memory for a OED, an input image is preprocessed. During that step, a Laplacian of Gaussian (LoG) filter is applied on the input image to obtain an image in contrast level [8]. Then, sub-quantization and sub-sampling are used to reduce the intensity and the position variations respectively. After that preprocessing step, patches of size $N \times N$ are provided to the associative memory module.



(a) Original pattern (b) Erasure

(c) Intrusion (d) Additive

Fig. 1: Example of an original pattern and noisy versions of it.

In order to perform the OED, the patterns are stored in the associative memory module. Those patterns represent gray edges surrounded by black pixels as depicted in Fig. 1a. These patterns are highly sparse since they are mostly composed of zero values (black pixels). Due to this particular configuration of patterns, classical CbNN [5], [6] cannot be used for OED (CbNN retrieval performance is only optimal for uniform independently distributed patterns). A variant of CbNN, named Sparse-CbNN, has been proposed in [3] in order to cope with these limitations. This variant is described hereafter.

B. Sparse-CbNN

A Sparse-CbNN is a neural network in which the neurons are split into c clusters of ℓ neurons. Each cluster is associated with an element of the input and each neuron is associated with a possible value of an element except the value 0. A neuron in a cluster can only be connected to neurons being in the other clusters. Those connections are binary: either they exist ($'1'$) or not ($'0'$). The set of binary connections are stored in an adjacency matrix $\mathcal{W}$ where $\mathcal{W}_{(i,j)(i',j')}$ is the connection between $n_{i,j}$ and $n_{i',j'}$, the i^{th} neuron in the cluster j and the i'^{th} neuron in the cluster j' respectively.

To store a pattern in a Sparse-CbNN, the neurons corresponding to the non-zero elements are activated and the connections between them are set to $'1'$. Fig. 2 depicts an example of a sparse-CbNN with $c = 6$ clusters of $\ell = 9$ neurons. Three patterns with 4 non-zero elements are stored in this network.

To retrieve a pattern from a noisy version of it, Algorithm 1 is used. Traditionally, the initialization rule used during the decoding process is the same as the one used during the storing process, the neurons corresponding to the non-zero elements are set to $'1'$. The dynamic rule is named *Sum-of-Max* (SoM) [9], [7] and computes a score for all the neurons of the network as follows:

$$s_{i,j} = e_{i,j} + \sum_{j' \neq j} \max_{i'} \mathcal{W}_{(i,j)(i',j')} e_{i',j'} \qquad (1)$$

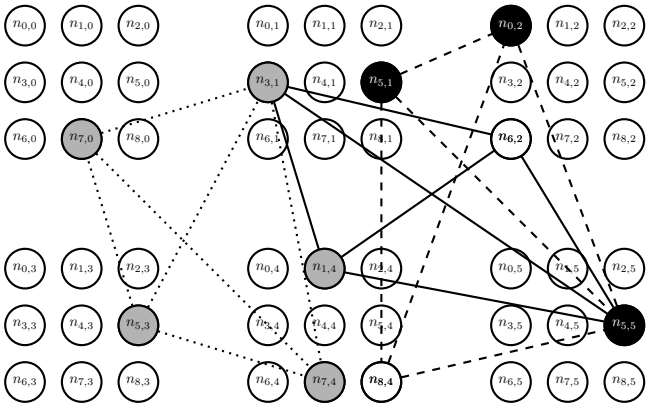


Fig. 2: Sparse-CbNN ($c = 6, \ell = 9$), 3 patterns having 4 non-zero elements are stored, lines represent the connections set to one in the network (solid, dashed and dot lines for pattern $P0 = \{n_{7,0}, n_{5,3}, n_{3,1}, n_{7,4}\}$, $P1 = \{n_{3,1}, n_{1,4}, n_{6,2}, n_{5,5}\}$ and $P2 = \{n_{5,1}, n_{0,2}, n_{5,5}, n_{8,4}\}$ respectively), the filled nodes represent two distinct activation with different noises (black and gray nodes represent erasure and intrusion).

where $e_{i,j} \in \{0, 1\}$ is the state of the neuron $n_{i,j}$. The activation rule can be changed depending of the type of noise and the desired performance in terms of error rate and speed of the algorithm [10].

In [3], we only considered the *erasure* noise. This type of noise leads to a noisy pattern in which one or several non-zero elements of the original pattern have been set to 0. Fig. 1b depicts this type of error in the context of OED. In this example, the missing pixel leads to the non-activation of a neuron in the corresponding pattern during initialization. The black nodes of Fig. 2 represent this situation. The original pattern $P2$ is composed of neurons $\{n_{5,1}, n_{0,2}, n_{5,5}, n_{8,4}\}$ and the noisy pattern is composed of neurons $\{n_{5,1}, n_{0,2}, n_{5,5}\}$. In this example, neuron $n_{8,4}$ has been erased.

To retrieve the erased symbols, the activation rule consists of triggering the neurons that have the maximum score over all the network after the application of the dynamic rule (1). This activation rule is named *Global-Winner-Takes-All* (G-WtA), [11]. In the example shown in Fig. 2, when SoM is used as the dynamic rule, neurons $n_{5,1}$, $n_{0,2}$, $n_{5,5}$ and $n_{8,4}$ have a score of 3, neurons $n_{3,1}$, $n_{1,4}$ and $n_{6,2}$ have a score of 1 and the others have a score of 0. The maximum score over all the network being 3, the neurons that belong to the original pattern are activated. We also have shown that in the case of erasure noise, the process SoM + G-WtA can be replaced by a friendly hardware boolean equation. To sum up this equation: to be activated, a neuron must be connected to at least one active neuron in each cluster that contains at least one active neuron. As a consequence, clusters with no active neuron are ignored.

C. Associated Hardware Architecture

A hardware architecture for the Sparse-CbNN was also proposed in [3]. That architecture is fully parallel, i.e. the states of all the neurons in network are computed at the same time and within one cycle. This architecture is mainly composed of interconnected identical modules, each one dedicated to

a cluster in the network. Each module contains two sub-modules: a storing sub-module and a decoding sub-module.

While the storing sub-module stores and updates the connections of the neurons in that cluster during storing phase, the decoding sub-module associated with that cluster computes their state during the decoding phase.

Data: Input Patch
Result: Closest pattern
Apply initialization rule;
while *Stopping criterion* **do**
  Apply a dynamic rule;
  Apply an activation rule;
end
Return the value of activated neurons;
Algorithm 1: Decoding algorithm of a CbNN for OED

III. More Complex Scenarios

In this section, we present the required modifications to handle additive and intrusion noise. Those noises lead to more complex scenarios during decoding phase in which the algorithm and the architecture presented in [3] cannot be used. To handle these types of noise, we propose a new initialization scheme along with corresponding dynamic and activation rules.

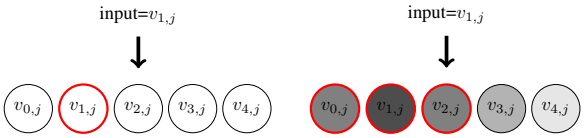
A. Intrusion noise

The first noise we consider is called *intrusion noise*. An intrusion occurs when at least one element at zero in the stored pattern becomes non-zero. This is depicted in Fig. 1c where there is only one intrusion. The consequence in the network is the activation of a spurious neuron. Two cases can arise from this example:

- **Case 1** : the spurious neuron is not connected to a neuron being part of the original pattern,
- **Case 2** : the spurious neuron is connected to one neuron being part of the original pattern. This case is represented by the gray neurons of Fig. 2. Neuron $n_{1,4}$ is activated and connected to neuron $n_{3,1}$ which is part of the original pattern $P0$.

For these two cases, the boolean equation proposed in [3] fails to retrieve the original pattern. In the first case, any neuron is connected to all the active neurons. In the second case, only one neuron is connected to all the other active neurons. Using SoM + G-WtA allows to handle the case 1 but not the case 2. If the dynamic rule SoM 1 is applied on the example represented by the gray neurons of Fig. 2, the following scores are produced: neuron $n_{3,1}$ has a score of 5, neurons $n_{7,0}$, $n_{5,3}$ and $n_{7,4}$ have a score of 4, neurons $n_{1,4}$, $n_{6,2}$ and $n_{5,5}$ have a score of 2 and all the others have a score of 0. By applying the activation rule G-WtA, only the neuron $n_{3,1}$ will be activated after the first iteration and the others will be shut down.

To handle intrusion noise, we propose to use a different activation rule named k-G-WtA. Instead of only activating the neurons having the highest score, this rule activates the neurons having one of the k highest scores. In [10], the authors show that k-G-WtA perform better than G-WtA when erasure noise is considered. However, we found that k-G-WtA is also efficient for intrusion noise. In the previous example, if $k = 3$, the 3 highest scores are $\{5, 4, 4\}$ and so neurons $n_{3,1}$, $n_{7,4}$, $n_{6,2}$ and $n_{5,3}$ will be activated at the next iteration. The parameter k is set to a high value, during the initialization phase and is decreased at each iteration.



(a) Hamming-like activation (b) Euclidean-like activation

Fig. 3: Initialization of the neurons of cluster j having 5 neurons.

B. Additive noise

The second type of noise considered in this paper is the *additive* noise. For this type of noise, a random value is added to each non-zero element of the original pattern (see Fig. 1d). This leads to a much more complex scenario in which the neurons being part of the original pattern still have to be activated.

Previous works [5], [11], [7] on CbNN are mainly interested in, given some inputs, finding the closest stored pattern according to an inner metric that resembles the Hamming distance. One of the reasons is that binary activations are particularly suitable for this kind of metrics. Fig. 3a depicts the Hamming-like activation inside a cluster j having 5 neurons, each value $v_{i,j}$ is associated with neuron $n_{i,j}$. The input value is equal to $v_{1,j}$ leading to the activation of neuron $n_{1,j}$. The activation of a neuron is pictured by a red circle.

However, the additive noise modifies the values of the symbols and therefore, it is necessary to take into account the distances between the input values and the values associated to the neurons. If we consider in the example of Fig. 3a that for $i \in \{0, ..., 4\}$ the values $v_{i,j}$ are in increasing order, the neuron $n_{0,j}$ is more probable than the neuron $n_{4,j}$ to belong to the original pattern.

In order to handle additive noise, we propose to add an action potential $p_{i,j}$ to each neuron i in each cluster j. It is obtained from the euclidean distance between the input value and the value associated with a neuron. For each neuron $n_{i,j}$, $p_{i,j}$ is given by:

$$p_{i,j} = (\max_i v_{i,j})^2 - (v_{in,j} - v_{i,j})^2, \tag{2}$$

where $\max_i v_{i,j}$ is the maximal input that may receive the network and $v_{in,j}$ the input value of cluster j. This action potential is similar to the Parabolic kernel used in [12] to initialize another type of neural network using binary connections. After the computation of action potentials in each cluster, we use them to activate the neurons in the network and to compute the score of the neurons during the dynamic rule of Algorithm 1.

During initialization, the $s \leq c$ neurons with the highest action potentials are activated. Fig. 3b shows euclidean-like activation on the same example as Fig. 3a for $s = 3$. The gray levels that color the neurons represent the distance between its associated value and the input value. Dark gray means that the distance is small while light gray means that the distance is high. The neurons associated with the 3 nearest values are activated.

To take into account the action potential in the dynamic rule we propose a new dynamic rule named Integer-SoM (I-SoM) that computes the score of each neuron as follows:

$$s_{i,j} = e_{i,j} p_{i',j'} + \sum_{j'=j} \max_{i'} \mathcal{W}_{(i,j)(i',j')} e_{i',j'} p_{i',j'}. \quad (3)$$

In other words, the score of a neuron is equal to the sum over all the clusters of the maximum action potential of the neurons for which it is connected. Once the scores are computed for all the neurons, we use the k-G-WtA to activate the neurons for the next iteration.

IV. Hardware Architectures

This section presents the architecture we propose for the enhanced model of associative memory. While the proposed architecture in [3] for the Sparse-CbNN is parallel and fast, it is costly in terms of hardware resources. In addition, all the connections are accessed at the same time and therefore, they are mapped onto independently-accessed registers. More optimized architectures for the classical CbNN (i.e. not sparse CbNN) have been proposed in the state of the art. The best architecture in [13] computes the value of each neuron within a constant latency (depending on the number of neurons in each cluster or the number of clusters). It also takes advantage of the symmetry of the adjacency matrix, i.e. it is only required to store once the connection between two neurons from different clusters. The best architecture proposed in [14] and implemented in [15] computes the value of the neurons in a relative short time by only focusing on the active neurons. That architecture also maps connections onto block RAMs instead of registers unlike the other architectures. However, the connection between two distant neurons has to be stored twice (for each cluster).

A. Baseline of the proposed architectures

Since [14] offers the best trade-off between speed and resources, we propose architectures based on the same scheme. These architectures contain four main modules (see Fig. 4):

1) The storing modules (one per cluster) contain $c - 1$ blocks RAM that store the connections between the neurons of a cluster and the neurons of the others.
2) The scoring modules (one per cluster) that compute and store the scores of the neurons.
3) The activation module (one for all the clusters) that computes the next state of each neuron in the network.
4) The Serial Pass Modules (SPMs) that output the active neurons in the clusters (one per cluster).

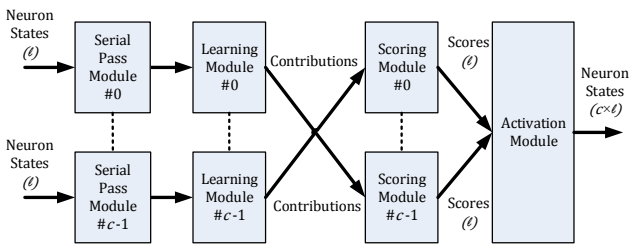


Fig. 4: Overview of the proposed architecture

The storing modules and the SPMs are similar for all the architectures while the scoring modules implement the different dynamic rules. The boolean equation proposed in [3] can be implemented as a dynamic rule in the scoring module. When this boolean equation is used, no activation module is needed since the boolean equation performs the neuron's activations. The activation module implements the k-G-WtA, since the G-WtA is equivalent to the k-G-WtA with k = 1.

B. Serial Pass Module

A SPM retrieves the indexes of the active neurons in a cluster. It takes as inputs the states of the neurons in that cluster and outputs the index of its active neurons. The states of the neurons in a cluster are stored in a binary vector of size ℓ. At each cycle, the address of a bit at $'1'$ is produced and that bit is set to $'0'$ for the next cycles. Therefore, the number of cycles needed to output all the active neurons of a cluster j is equal to a_j, the number of active neurons in that cluster, with $0 \leq a_j < \ell$. The number of cycles needed to output all the active neurons in the network is equal to $a_{max} = \max_j a_j$, the maximal number of active neurons in a cluster over the entire network.

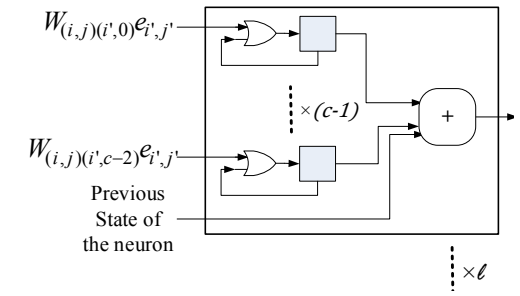
C. Storing Module

Each index provided by the SPM of a cluster is sent to the storing module associated to that same cluster. During the storing, there are at most one active neuron for each cluster. In a storing module having received an active neuron, the storing is performed in two steps. Firstly, the connections associated to that active neuron are read from the RAMs and secondly, these connections are updated by setting to '1' the connections with the active neurons in the other clusters.

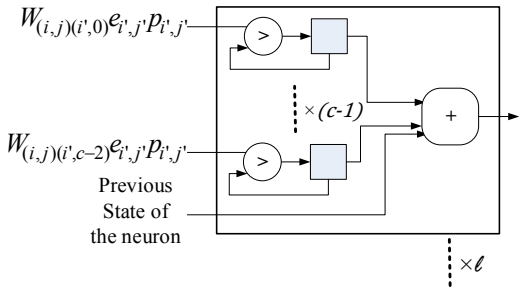
During the decoding, there is a_j active neurons in a cluster j. At each cycle and in each storing module, the set of connections corresponding to one of these active neurons is distributed to the scoring modules of the other clusters. We name the connections sent by an active neuron to a distant cluster as the *contribution* of that neuron to that cluster.

D. Scoring module

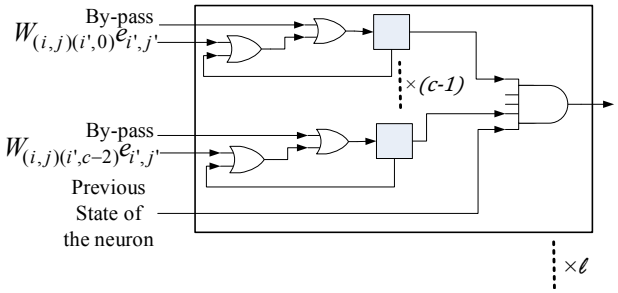
Since each storing module sends a contribution to the distant clusters at each cycle, each scoring module receives one contribution coming from each distant cluster. Using those values, each one computes the score of each neuron in its



(a) Sum-of-Max



(b) Integer-Sum-of-Max



(c) Boolean

Fig. 5: Architectures of the different scoring modules

associated cluster depending on the chosen dynamic rule i.e. the SoM (see (1)), the boolean equation used in [3] or the proposed I-SoM (see (3)). In terms of complexity, the number of connections between the storing modules and the scoring modules is in the order of $c^2 * l^2$.

1) Sum-Of-Max (SoM): For a cluster, the scoring module implements (1) for each of its neurons. For each neuron i of a cluster j and for a distant cluster j', the max operation ($\max_{i'} \mathcal{W}_{(ij)(i'j')}e_{i'j'}$) is performed sequentially (since one contribution from a cluster is sent at each cycle). That operation handles a set of bits ($W_{(ij)(i'j')}e_{i'j'}$). Therefore, it can be implemented by using a logical OR (See Fig. 5a). Each neuron is associated with $c-1$ intermediate values for each distant cluster. These values are stored in registers. When all the contributions of every cluster have been sent, an additional cycle is needed to compute the score of every neuron. The score of a neuron is computed by performing the sum over all the intermediate values associated with that neuron. This is done in parallel thanks to a tree adder.

2) Boolean equation: Using this rule, the max operation is performed using the same scheme as in the SoM implementation but the final sum (used in SoM) is replaced by a logical AND (See Fig. 5c). In a Sparse-CbNN, some clusters do not have any active neuron and have to be ignored. This is performed by using a by-pass mechanism. To sum up this mechanism, the intermediate register used to store the value of a logical OR related to a distant cluster j' is set to '1' if that cluster does not have any active neuron.

3) Integer-Sum-of-Max (I-SoM): The implementation of the I-SoM is nearly similar to the SoM implementation. It takes into account the action-potential of the neuron and the logical OR allowing to compute $\max_{i'} \mathcal{W}_{(ij)(i'j')}e_{i'j'}$ is replaced by a comparator allowing to compute $\max_{i'} \mathcal{W}_{(ij)(i'j')}e_{i'j'}p_{i'j'}$ (See Fig. 5b. Therefore, the registers used to store intermediate values are larger. One-bit registers are replaced by b-bit registers, b being the number of bits used to encode the action-potentials.

E. Activation module (k-G-WtA)

The k-G-WtA operation may easily require a lot of resources (comparators to compute the maximal scores, multiplexers to retrieve the winners, etc.). However, efficient architectures have been proposed in the literature to perform such task within a short time. We use the algorithm proposed in [16]. Starting from the most significant bits, this algorithm analyses the bits of a given power of two to find the k-winners. The k-winners are retrieved in m cycles, m being the number of bits required to store the score of a neuron.

F. Architectural Results

The different exposed models were implemented by using the scheme presented in the previous sections. Four architectures were designed and the five architectures that are to be compared are summed up in Table I:

- [3] is inspired by the original Sparse-CbNN model and uses the Boolean equation.
- V1 is implements the same model but uses a different scheme.
- V2 is inspired by the same model but uses the SoM + G-WtA.
- V3 implements the enhanced model able to handle the intrusion noise. It uses the SoM + k-G-WtA.
- V4 is based on the enhanced model able to handle the additive and the intrusion noises. It uses the I-SoM + k-G-WtA.

Those architectures were evaluated in terms of hardware resources and maximal frequency. The number of bits used in the block RAMs is not shown since it is the same for all the architectures. It is equal to the total number of connections in the network. The architectures have being designed by using VHDL. The simulation was performed by using ModelSim from Altera while the architecture was synthesized by using Altera Quartus and targeted the platform Stratix V 5SGXMABN3F45C2 FPGA, as in [3]. Results show that for

	Initialization	Dynamic	Activation
[3]	Hamming	Boolean equation	X
Arch. V1	Hamming	Boolean equation	X
Arch. V2	Hamming	SoM	G-WtA
Arch. V3	Hamming	SoM	k-G-WtA
Arch. V4	Euclidean	I-SoM	k-G-WtA

TABLE I: Rules used for the proposed architectures

every resource (ALMs and registers) and every network size, V1 always gets the lowest resource usage (see Fig.6a and Fig. 6b) and the best frequency (see Fig.6c) thanks to its less complex computing operators (no adders or comparators). [3] and V4 ALMs and registers occupation reaches the limit of the figures which means they do not fit on the platform.

Fig.6a shows that in terms of ALMs, V4 is more costly than the other architectures. This is mainly due to the increased size of adders in V4 compared to V3/V2, while they are replaced by logical AND in V1. However, the architecture proposed in [3] is at least 3 times more costly than V1 while they implement the same model. It is the same for V3/V2 while the latter are able to handle intrusion noises. Those comparisons show that using a more optimized baseline architecture to implement the modified models was really necessary.

Fig.6b shows that the cost of V4 in terms of registers is the highest one compared to the other architectures. That difference is explained by an increased number of bits required to store the scores of neurons. The number of registers in V3/V2 is slightly higher than in V1 since the latter does not store any score. The architecture proposed in [3] is at least 3 times more expensive than V1. This is due to the mapping of the connections to block RAMs in V1 instead of registers, unlike [3].

The energy consumption mainly depends on the frequency (clock) and the logic and on the routing [17]. Therefore, V1, V2 and V3 consume less energy compared to [3] since they divide the logic by 3 while having an increasing the maximal frequency only by 25%. However, V4 may consume more energy due to its much increased resource cost, even with its lower frequency.

G. Latency of the architectural variants

The overall latency of the proposed architectures is distributed among the scoring and the activation modules (except for V1 for which there are no activation modules). Each scoring module receives one contribution from an active neuron at each cycle. The latency depends on a_{max}, the maximal number of active neurons in a cluster over the entire network. Once all the contributions have been sent, an additional cycle is needed to compute the sum of the intermediate values (see Fig. 5). Thus the latency of the scoring module is equal to $\theta_{dec}(a_{max}+1)$.

Concerning the activation module, the latency θ_{act} is equal to m. For V2 and V3, the maximal score of a neuron is equal to c and so $m = log_2(c)$. For V4, the maximum score is equal to c multiplied by the maximal action-potential ℓ^2, thus $m = log_2(c.\ell^2)$. In comparison, the latency of the architecture proposed by [3] is 1 but that architecture cannot implemented large networks as shown in Fig.6a.

V. Simulation

In this section, we show the results of the evaluation of the proposed models used in the context of OED with intrusion and inversion noises. This evaluation examined the ability to retrieve patterns as well as the total latency to perform that task.

To evaluate the retrieval ability of Sparse-CbNN, we apply the following procedure:

1) Storage of the set of patterns in the network,
2) Random selection of one pattern and application of the noise on it to produce a noisy pattern,
3) Usage of each proposed model/implemented architecture to retrieve the closest pattern,
4) Checking if the retrieved pattern is the closest pattern (the retrieved pattern is compared with the result obtained by using a brute force approach based on a given distance).

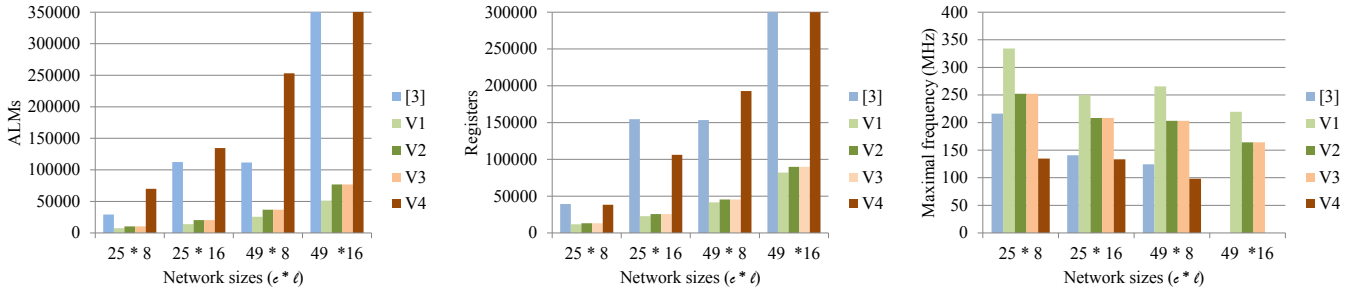
The steps 2-4 are iterated and the error rate is computed by taking the number of successes over number of trials. We use two different sets of pattern. **Set1** is composed of $8*8=64$ patterns of size 5×5 representing 8 orientations of $\ell=8$ intensities. **Set2** is composed of $16*8=128$ patterns of size 7×7 representing 16 orientations of $\ell=8$ intensities. For all the architectures and experiments, the stopping criterion is the number of iterations which is set to 4.

We began to evaluate the proposed models and associated architectures for intrusion noise, then for additive noise and finally for intrusion + additive noises.

A. Intrusion noise

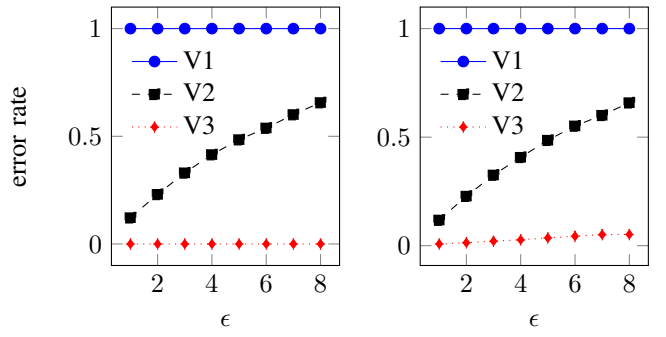
For the intrusion noise, we compare architectures V1, V2 and V3 (V4 has been proposed for additive noise). Intrusion noise is added by switching ϵ zero pixels in the original pattern to a non-zero value. The value assigned to the switched pixels is randomly selected from the ℓ possible values with the probability $\frac{1}{\ell}$. We consider that the closest pattern is the one with the smallest Hamming distance with the noisy pattern. For V3, we initialize k to 4 (number of iterations) and decrease it by 1 at each iteration.

Fig. 7 depicts the error rate as a function of ϵ for Set1 and Set2. As explained in section III-A, V1 is incapable to handle intrusion noise and the error rate is always equal to 1. For V2, the error rate grows in the same way for Set1 and Set2. In fact, for V2, an error occurs when the value of the switched pixel is equal to the value of the edge pixels (case 2 of section III-A). So the error rate is equal to the probability that there is one switched pixel equal to the value of the edge pixels. This probability is $1-\frac{(\ell-1)^\epsilon}{\ell^\epsilon}$. For V4, the error rate is equal to 0 for Set1 and slowly increases for Set2. That increase is due to the fact that more orientations are stored in the network.



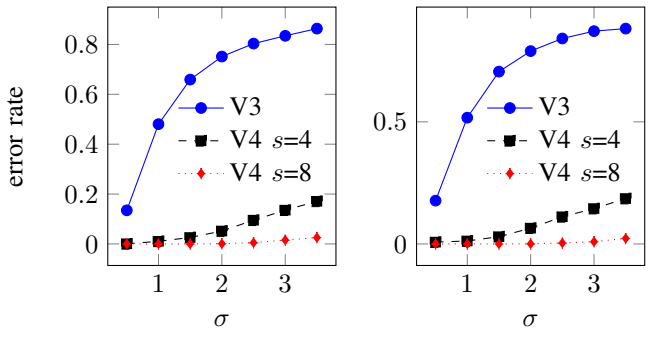
(a) Resource usage in terms of ALMs (b) Resource usage in terms of Registers (c) Maximal frequency

Fig. 6: Synthesis results for the evaluated different architectures.



(a) Set1 (b) Set2

Fig. 7: Error rate in function of intrusion noise ϵ, each point is the result of 10000 trials



(a) Set1 (b) Set2

Fig. 8: Error rate in function of additive noise $\xi =\sim \mathcal{N}(0, \sigma^2)$, each point is the result of 10000 trials

We compute $\mathcal{A} = \{\bar{a}^{(1)}_{max}, ..., \bar{a}^{(4)}_{max}\}$ the average of a_{max} for each iteration it of Algorithm 1. From $\mathcal{A}$, we derive the average overall latency of each architecture. For all the architectures (V1 to V3), for the two data set, we find $\mathcal{A} = \{1, 1, 1, 1\}$. The mean overall latency for architecture V1 is thus equal to 8 for the two sets. For architectures V2 and V3, the latency also depends on the data set (the size of the patterns leads to a particular c). For Set1 and Set2 the latency of is thus equal to 13 and 14, respectively.

B. Additive noise

Concerning the additive noise, we compare V3 and V4. To generate a noisy pattern, a random noise $\xi \sim \mathcal{N}(0, \sigma^2)$ is added to each edge pixel. We consider that the closest pattern is the one with the smallest euclidean distance with the noisy pattern. For V3, we keep the same procedure for k. As in V4 more neurons are activated during network initialization, we initialized k to $4N$ and decreased it by N at each iteration. N is equal to 5 and 7 for Set1 and Set2, respectively. We simulate V4 for 2 values of s (the number of neurons activated during initialization).

Fig. 8 depicts the error rate depending on σ^2 for Set1 and Set2. Note that they show the same behavior. For V3, the error rate grows quickly when the noise intensity increases. Using V4 leads to a lower error rate than V3 for the two values of s. However, for large value of σ, $s = 8$ leads to a lower error rate than $s = 4$. That is due to the fact that the neurons being part of the closest pattern have to be activated during the initialization to retrieve this pattern.

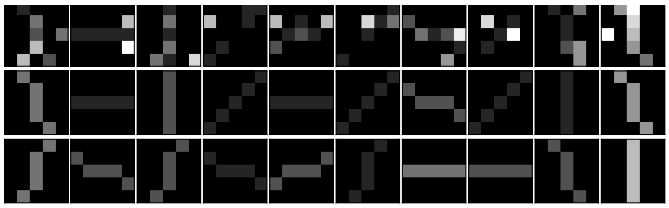


Fig. 9: Examples for Set1, of patterns and the retrieval results: the first row shows the random patterns, second row, the patterns retrieved by using V4 and $s = 8$, third row, the patterns retrieved with the euclidean distance

We also evaluated the latency of the architecture V4 for the two values of s. Like the V1, V2 and V3 architectures, we did not notice any difference in V4 when using Set1 and Set2. However, the number of active neurons is more important. For $s = 8$, $\mathcal{A} = \{8, 4, 3, 2.16\}$, leading to an average overall latency for Set1 and Set2 of 32.16 and 33.16 cycles respectively. For $s = 4$, $\mathcal{A} = \{4, 3.8, 3, 2.1\}$ leading to an average overall latency for Set1 and Set2 of 27.16 and 38.16 cycles.

C. Intrusion plus additive noise

As a final evaluation, we used V4 to retrieve closest patterns from random inputs. To generate the random patterns we

applied additive and intrusions noises to original patterns, $\epsilon = 2$ and $\sigma^2 = 1.5$. Combining those two noises leads to a high difference between the original patterns and the generated patterns. We compared the result obtained with V4 with the closest patterns found with the euclidean distance. Fig. 9 shows some cases for which euclidean distance and V4 do not retrieve the same pattern. Only the results for Set1 are presented in this paper (similar results are obtained for Set2). We can see that V4 performs better than euclidean distance in the majority of the cases. It is not surprising since it is well known that euclidean distance is not adapted for sparse signals. This is due to the fact that it gives the same importance to any dimension.

VI. Conclusion

In this paper, improved models and their associated hardware architectures were presented. They allow to detect oriented edges when the patterns in the inputs are noisy. The proposed models are able to manage more types of noise than the state of the art and the most advanced model performs better than the euclidian distance for that task. Concerning the architectures, even if they are more efficiently designed compared to the first implementation of the Sparse-CbNN model, they remain expensive in terms of computing resources. Concerning that matter, more generic and performant architectures for the Sparse-CbNN and more generally, CbNN inspired models are under research. From a set of allocated computing and memory resources, those are able to handle networks with different sizes while taking in account the sparsity of the active connections and active neurons in the network.

Acknowledgment

References

[1] N. Senthilkumaran and R. Rajesh, "Edge detection techniques for image segmentation–a survey of soft computing approaches," *International journal of recent trends in engineering*, vol. 1, no. 2, 2009.

[2] G. Shrivakshan and C. Chandrasekar, "A comparison of various edge detection techniques used in image processing," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 5, pp. 272–276, 2012.

[3] R. Danilo, H. Jarollahi, V. Gripon, P. Coussy, L. Conde-Canencia, and W. J. Gross, "Algorithm and implementation of an associative memory for oriented edge detection using improved clustered neural networks," in *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 2501–2504.

[4] G. Palm, "On associative memory," *Biological cybernetics*, vol. 36, no. 1, pp. 19–31, 1980.

[5] V. Gripon and C. Berrou, "A simple and efficient way to store many messages using neural cliques," in *Proceedings of IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain*, Paris, France, April 2011, pp. 54–58.

[6] ——, "Sparse neural networks with large learning diversity," *IEEE Transactions on Neural Networks*, vol. 22, no. 7, pp. 1087–1096, July 2011.

[7] R. Danilo, P. Coussy, L. Conde-Canencia, V. Gripon, and W. J. Gross, "Restricted clustered neural network for storing real data," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 2015, pp. 205–210.

[8] A. Huertas and G. Medioni, "Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 5, pp. 651–664, 1986.

[9] V. Gripon and C. Berrou, "Nearly-optimal associative memories based on distributed constant weight codes," in *Information Theory and Applications Workshop (ITA), 2012*. IEEE, 2012, pp. 269–273.

[10] A. Aboudib, V. Gripon, and X. Jiang, "A study of retrieval algorithms of sparse messages in networks of neural cliques," in *Proceedings of Cognitive 2014*, May 2014.

[11] B. K. Aliabadi, C. Berrou, V. Gripon, and X. Jiang, "Storing sparse messages in networks of neural cliques," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, pp. 980–989, 2014.

[12] V. J. Hodge and J. Austin, "A binary neural k-nearest neighbour technique," *Knowledge and Information Systems*, vol. 8, no. 3, pp. 276–291, 2005.

[13] P. Coussy, C. Chavet, H. N. Wouafo, and L. Conde-Canencia, "Fully binary neural network model and optimized hardware architectures for associative memories," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 11, no. 4, p. 35, 2015.

[14] C. Chavet, P. Coussy, and N. Charpentier, "Architecture de réseau de neurone, procédé d'obtention et programmes correspondants," May 30 2014, wO Patent App. PCT/EP2013/074,518. [Online]. Available: http://www.google.com/patents/WO2014079990A1?cl=fr

[15] H. Jarollahi, N. Onizawa, and W. J. Gross, "Selective decoding in associative memories based on sparse-clustered networks," in *Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE*. IEEE, 2013, pp. 1270–1273.

[16] M. Yoon, "A parallel search algorithm and its implementation for digital k-winners-take-all circuit," *JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE*, vol. 15, no. 4, pp. 477–483, 2015.

[17] J. Lamoureux and W. Luk, "An overview of low-power techniques for field-programmable gate arrays," in *Adaptive Hardware and Systems, 2008. AHS'08. NASA/ESA Conference on*. IEEE, 2008, pp. 338–345.