Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures

Madeleine Abernot, Aida Todri-Sanial

Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures

To cite this article: Madeleine Abernot and Todri-Sanial Aida 2023 *Neuromorph. Comput. Eng.* **3** 014006

IOP Publishing Neuromorph. Comput. Eng. 3 (2023) 014006 https://doi.org/10.1088/2634-4386/acb2ef

NEUROMORPHIC
Computing and Engineering

OPEN ACCESS

RECEIVED
7 July 2022

REVISED
29 September 2022

ACCEPTED FOR PUBLICATION
13 January 2023

PUBLISHED
6 February 2023

PAPER

Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures

Madeleine Abernot[1] and **Todri-Sanial Aida**[1,2,*]

[1] LIRMM, University of Montpellier, CNRS, Montpellier, France
[2] Electrical Engineering Department, Eindhoven Technical University, Eindhoven, The Netherlands
[*] Author to whom any correspondence should be addressed.

E-mail: a.todri.sanial@tue.nl

Keywords: oscillatory neural networks, hetero-association, feedforward, image edge detection

Abstract

The growing number of edge devices in everyday life generates a considerable amount of data that current AI algorithms, like artificial neural networks, cannot handle inside edge devices with limited bandwidth, memory, and energy available. Neuromorphic computing, with low-power oscillatory neural networks (ONNs), is an alternative and attractive solution to solve complex problems at the edge. However, ONN is currently limited with its fully-connected recurrent architecture to solve auto-associative memory problems. In this work, we use an alternative two-layer bidirectional ONN architecture. We introduce a two-layer feedforward ONN architecture to perform image edge detection, using the ONN to replace convolutional filters to scan the image. Using an HNN Matlab emulator and digital ONN design simulations, we report efficient image edge detection from both architectures using various size filters (3×3, 5×5, and 7×7) on black and white images. In contrast, the feedforward architectures can also perform image edge detection on gray scale images. With the digital ONN design, we also assess latency performances and obtain that the bidirectional architecture with a 3×3 filter size can perform image edge detection in real-time (camera flow from 25 to 30 images per second) on images with up to 128×128 pixels while the feedforward architecture with same 3×3 filter size can deal with 170×170 pixels, due to its faster computation.

1. Introduction

The number of edge devices being used in everyday life is quickly growing. For healthcare, security, robotics, or even home automation, there are edge computation in every domain, inducing a large amount of data to treat. Classical computing algorithms along with classical hardware systems based on von-Neuman architecture cannot efficiently handle this large amount of data at the edge [1, 2]. Artificial intelligence (AI) field has introduced many efficient algorithms, such as artificial neural networks (ANNs), to treat big data problems. However, edge devices with limited bandwidth, memory, and energy consumption constraints simply cannot manage such complex algorithms in real-time. Thus, recently novel hardware-aware computing paradigms inspired by the brain, so-called neuromorphic computing have emerged [3, 4].

Neuromorphic computing takes inspiration from the brain architecture, and the brain representation of information to allow fast, and low power computation. For example, spiking neural networks (SNNs) [5–7] are neural networks inspired by spikes transmitted through neurons in human brain, which encode information in the latency between two spikes. Using time-domain data representation instead of amplitude-domain helps reducing the mean voltage amplitude and so the energy consumption. Implementation of large-scale SNNs in neuromorphic chips demonstrates high efficiency in solving various complex problems [8, 9]. In this work, we explore a novel neuromorphic paradigm with Oscillatory Neural Networks (ONNs).

ONNs are brain-inspired neural networks that emulate brain oscillations using networks of coupled oscillators [10–13]. ONN computation uses the natural synchronization behaviour of coupled oscillators. In this work, we use phase-computing ONN where the information is encoded in the phase relationship between oscillators [14]. It allows to decrease the signal voltage amplitude and to reduce the power consumption, such as with SNNs [15–17]. In recent years, ONNs have shown good performances to be deployed for edge applications with low power consumption. In particular, ONN is often used with fully-coupled bidirectional connections to perform auto-associative memory (AAM) tasks [18, 19], like Hopfield neural networks (HNNs) [20]. However, to solve edge AI type of problems and to benchmark ONN with other computing paradigms, there is a need to explore other applications (beyond associative memory), learning algorithms and architectures. In this work, we explore two ONN layered architectures, first with bidirectional connections introduced by [21], and then with feedforward connections, introducing a novel feedforward ONN architecture. We use both architectures to perform image edge detection. We believe performing image edge detection with two-layer ONN architecture is a first step toward exploring ONNs beyond associative memory tasks and start benchmarking ONN with other existing solutions.

Edge detection is an image processing task that detects brightness and color variations in images. Many edge devices are equipped with smart cameras, where image processing is performed at the edge in real-time [22]. Edge detection is often a first step of a more complex image processing operation, like for feature extraction, object detection, image segmentation, etc [23, 24]. Classical edge detection solutions, such as Sobel [25] and Canny [26], use small (from 3×3 to 7×7) convolutional filters to scan the image and detect contrasts in small part of the image.

In [21], authors have shown a 2-layer bidirectional ONN can perform image edge detection. Authors use coefficients from Sobel convolutional filters to compute ONN synaptic weights using the Hebbian learning rules. They achieve good results with some missing diagonal edges. In this paper, we propose a novel method to configure a two-layer bidirectional ONN to perform edge detection which rectifies the missing diagonal edges. Moreover, we show that the novel method can be adapted to a two-layer feedforward ONN architecture. We first test and validate our methods on Matlab with an HNN emulator, and then we simulate them on a digital ONN design. This work introduces the feedforward ONN and its digital implementation. Our contributions can be summarized as follows: we (a) develop a novel method to perform image edge detection with a two-layer bidirectional ONN, (b) introduce a two-layer feedforward ONN architecture and its configuration to image edge detection, and (c) validate and evaluate real-time performances of both our solutions on black and white and gray scale images in Matlab HNN emulator and digital ONN design.

The rest of the paper is organized as follows. Section 2 presents the different ONN architectures, starting from state-of-the-art AAM, to the novel proposed two-layer feedforward ONN. Also, we describe the implementations of the architectures in Matlab, and the digital ONN design. Then, section 3 details how we configure each ONN architecture to perform image edge detection using results from the HNN Matlab emulator. Next, in section 4, we show results of simulations of our bidirectional and feedforward digital ONNs on black and white and gray scale images. We also provide performances of our solutions in terms of latency and resource utilization. Finally, in section 5, we discuss our results compared to other edge detection field programmable gate array (FPGA) implementations and the perspectives of our work.

2. ONNs

ONNs are networks of coupled oscillators [27] that compute with the natural dynamics of coupled oscillators [28, 29]. In ONNs, each neuron is an oscillator and each synapse is a digital or analog component that couples two oscillators. The coupling strength varies depending on the component type and value [30]. In this work, we consider phase-computing ONNs that encode information in the phase relationship between oscillators. For example, considering binary information, we set an oscillator with 0° phase to encode a logic '0', and an oscillator with 180° phase to encode a logic '1'. Computing with phases allows to reduce the voltage amplitude and consequently the power consumption.

In this section, we present the different ONN architectures, starting from state-of-the-art of single-layer fully connected ONN architecture in section 2.1. Then, section 2.2 gives details on the 2-layer bidirectional architecture introduced in [21] for image edge detection. Section 2.3 introduces the novel 2-layer feedforward ONN architecture. Finally, section 2.4 describes the implementations of the ONN architectures in Matlab and with the digital design.

2.1. Single-layer fully-connected ONN

ONN state-of-the-art is configured with a recurrent fully-connected architecture, see figure 1(A), like in HNNs. HNNs are fully-connected recurrent networks where each neuron is a perceptron with bipolar states evolving in time such as:

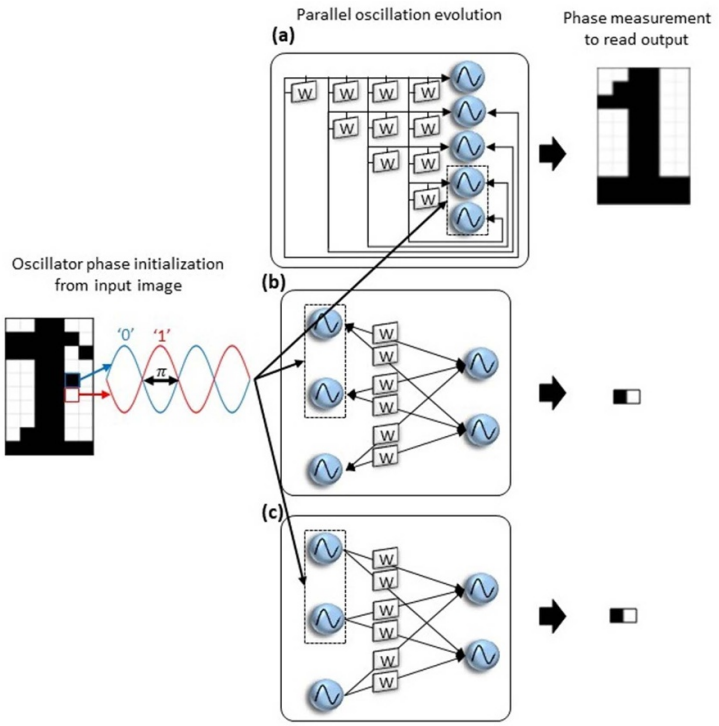

Figure 1. ONN architectures with (a) fully-connected connections for Auto-Associative Memory task, (b) bidirectional layered connections for Hetero-Associative Memory task, and (c) feedforward connections for classification task.

$$\sigma_i^t = sign\left(\sum_j w_{ij}\sigma_j^{t-1}\right) \tag{1}$$

where σ_i^t is the state of neuron i at time t, w_{ij} is the weight value between neuron i and neuron j, and σ_j^{t-1}, is state of neuron j at time $t-1$. Note, if $\sum_j W_{ij}\sigma_j^{t-1}$ is equal to 0, $\sigma_i^t = \sigma_i^{t-1}$. Configuring weight values with specific learning rules, HNNs can perform AAM tasks. AAM networks can learn patterns or images and retrieve them from corrupted information such as noisy images. In HNN, if neuron states are initialized with a corrupted information, then the states evolve in time to stabilize to a memorized pattern. Memorized patterns are embedded in the weights of the network configured during learning. The main learning algorithm for AAM is the unsupervised Hebbian learning rule which works as 'neurons that fire together, wire together' [31]. It means if two connected neurons have equal values, the coupling strength between them is reinforced. The mathematical formulation to compute weights w_{ij} between neuron i and neuron j corresponding to k learning patterns X^k is:

$$w_{ij} = \frac{1}{k}\sum_k X_i^k X_j^k \tag{2}$$

with $w_{ij} = 0 \forall i = j$. Note that with Hebbian rules, HNN learns a pattern and also the opposite one.

It is well-known that ONN with HNN architecture can also perform AAM tasks, [18, 19]. The synaptic configuration of ONN for AAM is set with the unsupervised Hebbian rule to learn patterns. Weights are computed with Hebbian and bipolar HNN states, and then values are translated into ONN coupling elements [30]. Note that Hebbian can only learn bipolar patterns as it is configured for HNN, however, ONN can encode more than two states with its phase encoding, so it can learn and stabilize to additional phase states between 0° and 180°.

After learning, ONN inference starts by initializing each oscillator in the network with its initial phase state. Then, oscillators' phases evolve, thanks to the coupling among oscillators, until they stabilize to a final phase state. If couplings among oscillators are correctly configured, the stable phase state, corresponding to the minimal energy of the system [32], represents one of the memorized patterns. In the case of image processing, each neuron is associated with a pixel, and the phase of each oscillator represents the color of the pixel. Using bipolar patterns with HNN limits the image processing to black and white, while ONN can also initialize and stabilize to gray scale images.

2.2. Two-layer bidirectional ONN

Fully-connected recurrent ONNs can perform pattern recognition by associating a corrupted pattern with a learnt pattern. However, to enlarge the scope of possible ONN tasks, other architectures are explored [33]. Thus, in [21], authors introduced a novel ONN architecture dividing neurons into two different layers, an input layer, and an output layer, with bidirectional connections between input and output layers, see figure 1(B).

This architecture resembles the bidirectional associative memory (BAM) networks [34], in which neurons are equivalent to Hopfield neurons with bipolar states, and connections are bidirectional. However, update of BAM input and output neuron states is sequential from input layer, to output layer, and reversely until stabilization. BAM networks can perform hetero-associative memory (HAM) tasks, meaning that it can associate pairs of input/output patterns with two different dimensions representing two different information. For example, it can associate an image to a class, becoming a classification task, or a sensory input data to an action in the robotics domain.

BAM networks [34] use an adapted version of the unsupervised Hebbian learning rule to compute weights between input and output. To learn k input/output pairs of patterns with i neurons in the input layer X, and j neurons in the output layer Y, weight w_{ij} between input neuron i and output neuron j is computed following:

$$w_{ij} = \sum_{k} X_i^k Y_j^k \tag{3}$$

The two-layer bidirectional ONN differs from BAM as the bidirectional connection between neurons induces a parallel exchange from input to output layer and from output to input layer. Thus, if only neurons from the input layer are initialized, neurons from the output layer will also impact the final computation result. So, the impartial initialization of neurons in output layer is important for an efficient computation. As with Hebbian learning rule, we limit the pattern output state to two phase states $\{0°, 180°\}$, same as authors in [21] proposed to double each output neuron to initialize one with one phase state $\{0°\}$, and the other with the opposite phase state $\{180°\}$. In this case, the output has no influence on the input information. Note that each doubled output neurons are connected as they represent the same output information. During the learning step, in order to also compute ONN weights for connections in-between outputs, authors in [21] translate each input/output pair patterns into a unique vector to compute the classical Hebbian coefficients as in equation (2), then removing connections among input neurons afterwards.

During inference, input neurons are initialized with input information, and output neurons with neutral information. Then, using coupling between input/output layers, and coupling among output neurons, ONN phases evolve and stabilize to a final phase state. Phases of the output neurons contain the computation output.

2.3. Two-layer feedforward ONN

In this paper, we introduce a novel ONN architecture with feedforward connections, to remove the impact of the output over the input layer, and only consider the impact of the input layer over the output layer, see figure 1(C). Also, we expect the two-layer feedforward architecture to compute faster than the two-layer bidirectional architecture. The bidirectional architecture computes with interactions between input and output until both stabilize. However, in the feedforward architecture, only the input influences the output and the input does not evolve in time, so we expect that output stabilizes faster.

Considering HNN neurons instead of oscillators, stabilization can even become a 'one-shot' computation as:

$$\sigma_j^t = sign\left(\sum_i w_{ij}\sigma_i^{t-1}\right) \tag{4}$$

with σ_j^t, the state of output neuron j at time t, σ_i^{t-1} the state of input neuron i at time $t-1$, and w_{ij} weight between input neuron i and output neuron j. Note that if the weighted sum $\sum_i W_{ij}\sigma_i^{t-1}$ is equal to zero, the new state of neuron j, σ_j^t takes the previous value σ_j^{t-1}. This is true for both HNNs and ONNs. Thus, it is still necessary to initialize neurons in the output layer to prevent this behavior. Note, the initialization is application-dependent.

In the two-layer feedforward ONN, oscillators in the input layer are phase-controlled oscillators taking phases as input and sending oscillating signals to oscillators in the output layer. Oscillators in the output layers take as input the sum of oscillating signals sent from input layer, equivalent to the weighted sum. Output oscillators evolve in phase depending on this weighted sum. Training a two-layer feedforward ONN

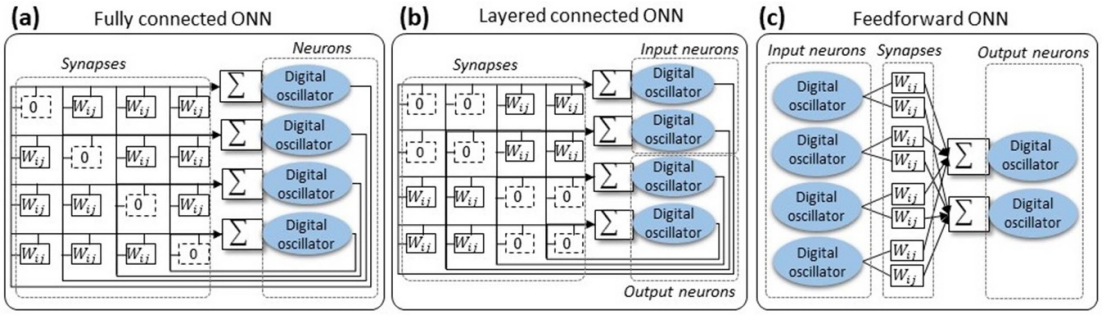

Figure 2. ONN digital implementation on FPGA with (a) Fully-connected architecture, (b) Bidirectional layered architecture, and (c) Feedforward architecture.

is not trivial and needs further exploration. However, theoretically, it can use unsupervised learning rules used for HAM tasks, as well as supervised learning rules used for classification tasks. In this paper, we use the two-layer feedforward ONN for a specific image edge detection application and compute custom weights dedicated to image edge detection.

Note that hardware implementation of such feedforward ONN is not possible with every oscillator design. Some analog designs cannot differentiate between input and output coupling directions. In this case, only bidirectional connections are possible. However, using some digital design, it is possible to differ oscillators' input and output as in [35, 36] which can allow to implement a feedforward ONN architecture.

2.4. ONN validation and simulation

In this work, we use two ONN designs to simulate and validate our architectures. First, we use an ONN Matlab emulator based on HNN described in section 2.4.1 to validate our methods. Then, we simulate a digital ONN design from [36] presented in section 2.4.2, to assess ONN efficiency and evaluate resource and timing performances.

2.4.1. HNN-based Matlab emulator

In order to easily validate our architectures, and their configuration for image edge detection, we first test our methods using an HNN emulator developed on Matlab for each architecture. HNN emulators consists of modeling ONN architectures using bipolar spins as neurons, like in HNN. For the fully-connected architecture, the Matlab emulator reproduces identically an HNN, but it is not used for the image edge detection application. For the two-layer bidirectional architecture, we consider a fully-connected HNN with zero weights between input neurons, such that we emulate the parallel behavior of the bidirectional architecture. Finally, for the two-layer feedforward architecture, we create a Matlab code which emulates the two-layer feedforward network with bipolar spins as neurons. The input layer takes directly the input information, and the output layer is updated by the weighted sum value given by the input information as in equation (4). Note that the output layer initialization is application-dependent.

2.4.2. ONN digital design

After validating architectures and image edge detection methods with HNN Matlab emulator, we check and evaluate our solutions by simulating a digital ONN design. The ONN digital design was introduced in [36] with a fully-connected architecture. In the design from [36], each oscillator combines a phase calculator as well as a phase-controlled digital oscillator. Synapses are five-bits signed registers, see figures 2(A) and 3. In [36], Oscillators are initialized sequentially before starting their parallel computation. Authors from [36] used their design to perform real-time image recognition. Then, in [21] authors simulated their two-layer bidirectional architecture, using the fully-connected digital ONN design from [36], and applying zero weights in synaptic couplings between input oscillators, see figure 2(B). In this work, we use the same design for the bidirectional layered architecture, however we modify the design to perform a parallel initialization instead of a sequential one to speed up the process. Finally, we develop a two-layer feedforward design inspired by the previous ONN design. The new feedforward ONN architecture implementation is similar to the bidirectional ONN. Oscillators from the output layer are equivalent to oscillators from the bidirectional architecture, containing a phase calculator followed with a phase-controlled oscillator. However, oscillators from the input layer are only phase-controlled oscillators which take the input phase value and generates an oscillating signal, see figure 3. In this new design, there is no more recurrence between input and output layers. Synaptic implementation also uses five-bits signed representation for each weight, and initialization of

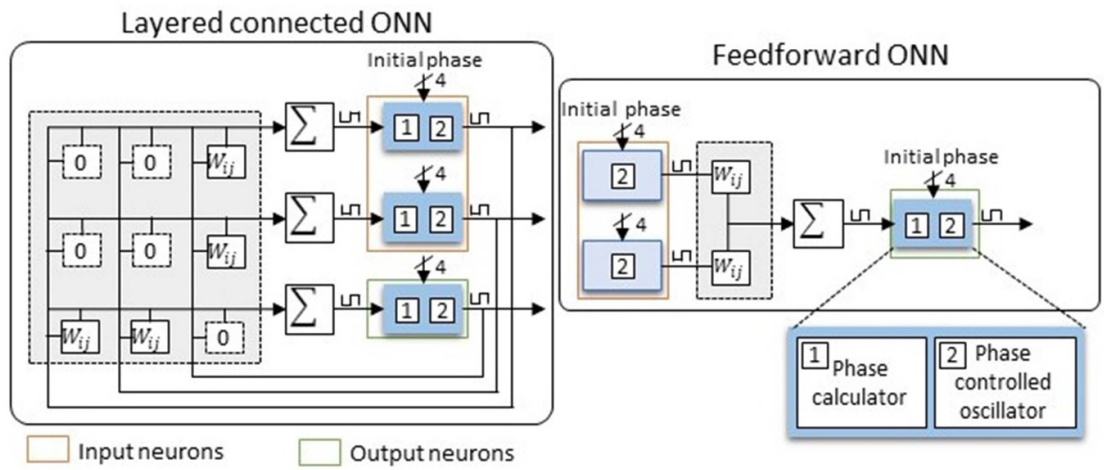

Figure 3. Digital implementation of the two-layer bidirectional ONN, and the two-layer feedforward ONN with details of neuron components.

the input layer is also performed in parallel. We set up the two-layer ONN architectures to perform image edge detection. Note, in this paper, we focus on simulation tests. We use the Vivado software to simulate the digital ONN designs with the XC7Z020-1CLG400C FPGA as target device.

Next, we present methods to perform image edge detection using the different ONN architectures.

3. ONN for image edge detection

Edge detection task consists on extracting contrast differences between various regions of an image. Typically, it uses small-size filters to scan images and extract contrasts in the different image parts. Here, we present first the existing edge detection algorithms. Then, we show how a two-layer ONN with bidirectional architecture can perform image edge detection using a 3×3 input layer. After, we extend the method to the two-layer feedforward ONN architecture. Also, we show that the 3×3 input layer can be extended to larger filters with 5×5 and 7×7 pixels for both bidirectional and feedforward ONN. Finally, we describe the method used to evaluate our ONN-based edge detection solutions.

3.1. Edge detection algorithms

Various edge detection algorithms exist to detect brightness and color differences between two neighboring regions of an image. State-of-the-art Sobel [25], and Canny [26] algorithms use small convolutional kernels (commonly 3×3, 5×5 and 7×7) to scan the image and detect edges in the different area of the image. For example, Sobel commonly uses two 3×3 convolutional kernels associated to vertical and horizontal edges. Scanning the image consists on applying the convolutional kernels' parameters on small parts of the image, 3×3 if the kernel is 3×3, and moving the kernel all around the image with one pixel stride. The results of the convolutional kernels are then used to calculate a global gradient for the central pixel of the 3×3 image window which indicates if an edge is detected or not in the given pixel, and how strong the edge is. An optional final state binarize the output by using a threshold to select only strong edges in the image. Both Sobel and Canny use at least two kernels to detect horizontal and vertical edges. In addition, Canny includes a previous step with a gaussian filter to remove noise in the image.

3.2. 2-layer bidirectional ONN with 3×3 input size

A recent paper introduced ONN as a solution to perform image edge detection, using a two-layer bidirectional architecture [21]. The method proposed in [21] consists on using the bidirectional ONN as a filter with a 3×3 input format and a two-neuron output to represent if an edge is detected or not. The network is configured using the Sobel kernels coefficients associated to output combinations as training patterns for the Hebbian algorithm, see figure 4. One pattern associates the horizontal Sobel kernel with white pixels output corresponding to $\{-1, -1\}$ for HNN, or $\{0°, 0°\}$ for ONN, and the second pattern associates the vertical Sobel kernel with black pixels output corresponding to $\{+1, +1\}$ for HNN, or $\{180°, 180°\}$ for ONN. No-edge cases are detected when the ONN stabilizes to none of the trained output patterns. Note, weights connected from one input neuron to the two output neurons are equal because the two output neurons need to represent the same edge information.

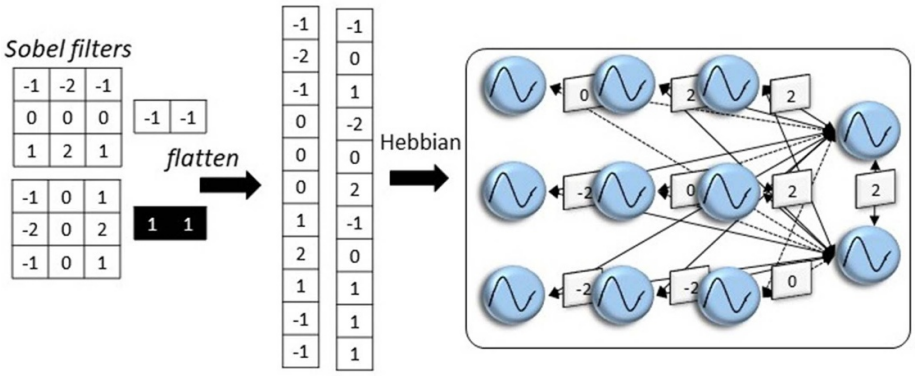

Figure 4. Bidirectional ONN configuration for edge detection proposed by [21], and resulted connections using 3 × 3 input filter.

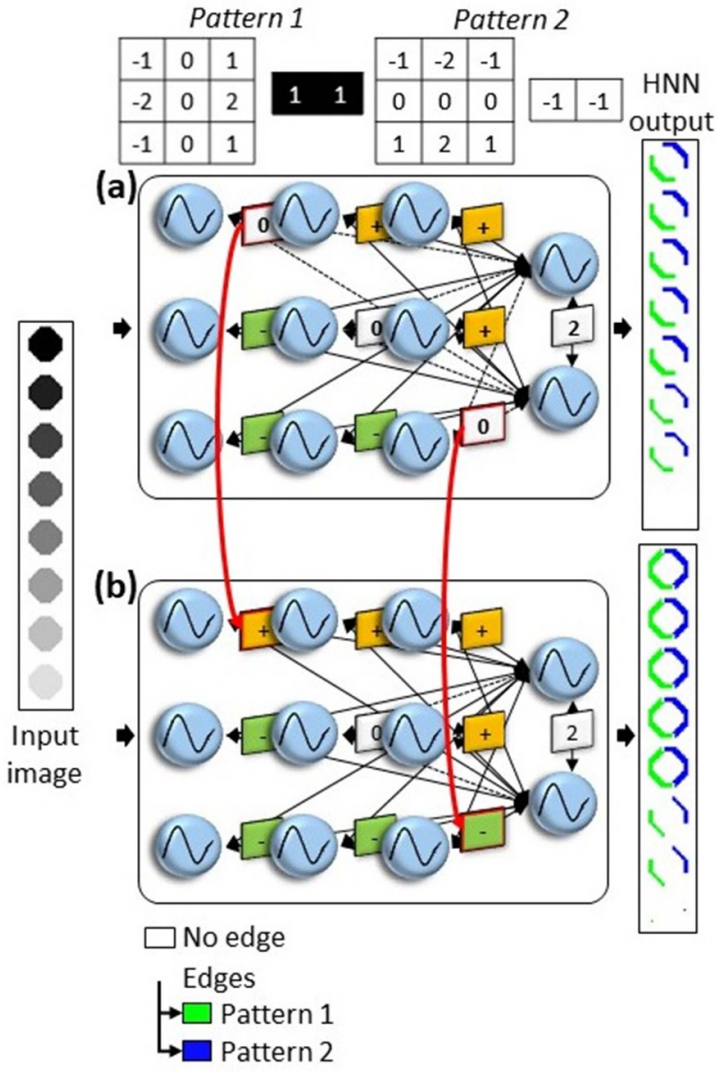

Figure 5. (B) Customization of the bidirectional ONN configuration for edge detection solving missing edges from (A) previous work [21], and results of the HNN Matlab emulator for both configurations on the gray-scale octagon map.

Figure 5(A) shows results obtained on a gray-scale octagon map with methods from [21] using the HNN Matlab emulator. It is visible that some diagonal edges are missing. Thus, in this paper we apply a novel method to customize weights and solve the missing edges. When looking at the weights of the corresponding network, we detected that weights applied on opposite edge sides of the filter have opposite values. For example, weights from top right input neuron to output neurons are −2, while weights from bottom left

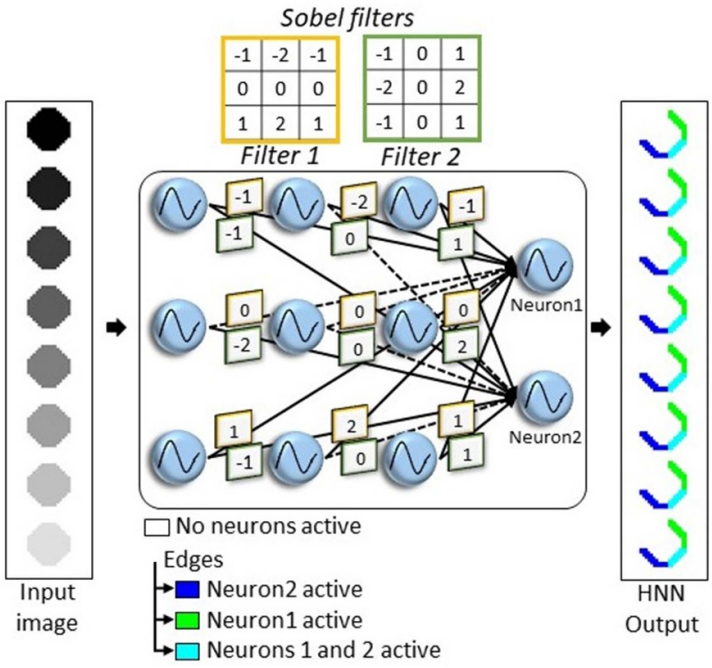

Figure 6. Configuration of the feedforward ONN with 3 × 3 input size, using Sobel kernel coefficients as weights, and results of the HNN Matlab emulator on the gray-scale octagon map.

input neuron to output neurons are +2, corresponding to the right-oriented diagonal. Vertical and horizontal edges have the same values, respectively. Those three edges are correctly retrieved. However, for input neurons corresponding to the missing left-oriented diagonal edges, weights connected to the output neurons have zero values. Thus, in order to solve the missing edges, we apply the same rule to weights connected to those input neurons. We customize the weights to have opposite weight values between weights connected to the top-left input neuron, and weights connected to the bottom-right input neuron. Figure 5(B) shows the customization of the weights as well as the results obtained on a simple gray scale octagon map with the HNN Matlab emulator. It shows that our customization method improves the image edge detection proposed by [21]. Results and precision on larger scale images are reported in section 4, as well as resource and timing performances of the digital ONN design.

3.3. 2-layer feedforward ONN with 3 × 3 input size

One main drawback in [21] is the latency of the image edge detection algorithm to compute a full image, due to the sequential scanning process. In order to explore reducing the computation latency of ONN, we introduce a feedforward ONN architecture, where we expect stabilization to be faster than with bidirectional interaction. Also, we expect a reduction in the computation latency as a result of the new parallel initialization from the digital design.

In the mathematical concept, the weighted sum computed before the neuron activation function is equivalent to a convolutional operation, as used in classical edge detection algorithms. Thus, our first approach to use the feedforward ONN for edge detection consisted on applying the Sobel kernels coefficients as weights between the 3 × 3 input neurons and two output neurons, one for each Sobel kernel, as shown in figure 6. In this case, output neurons are initialized with −1 for HNN or 0° phase for ONN that we associate to a no-edge information. Then, we expect the feedforward ONN to change the output information to +1 for HNN, or 180° phase for ONN if an edge is detected. However, figure 6 highlights that not all edges are correctly retrieved by our HNN Matlab emulator. It can be explained by the difference between the Sobel gradient calculation and the HNN or ONN activation functions. In Sobel, a large negative intensity and a large positive intensity of gradient will correspond to an edge. However, with HNN or ONN, if the weighted sum is negative, the states will evolve to −1 for HNN, and 0° phase for ONN, corresponding to a no-edge information. A solution to counter this effect could be to use two output neurons per kernel, initializing both with opposite values, and checking if at least one changed during the computation. However, this solution uses a double amount of output neurons, inducing twice the number of synapses, and so increasing ONN resource utilization.

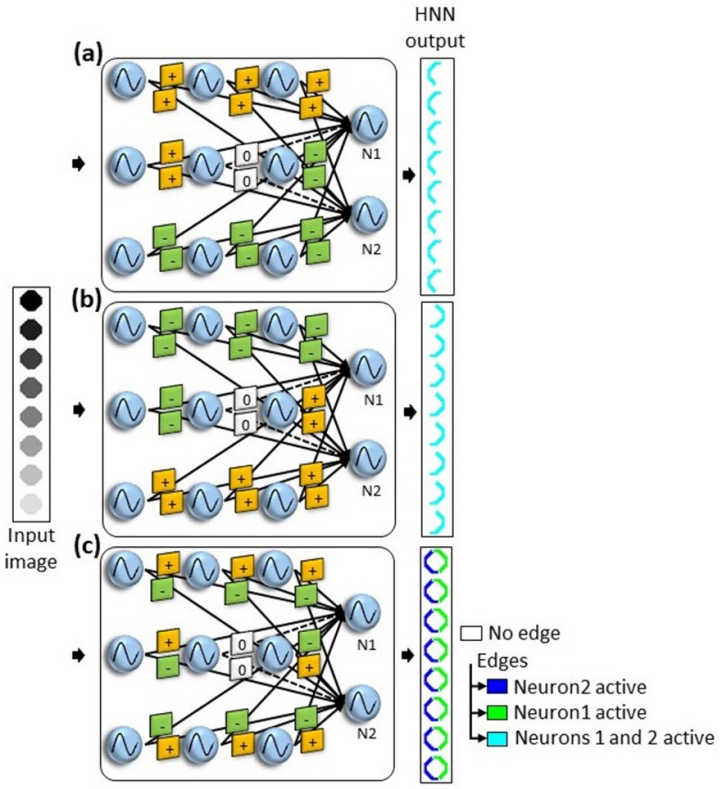

Figure 7. Feedforward ONN configuration with 3 × 3 input size, with (A) first option tried from the bidirectional ONN configuration, (B) second option by reversing weights of the first option, and (C) final option which combines (A) and (B). Also results of the HNN Matlab emulator on the gray-scale octagon map are presented for each configuration.

Thus, we explored alternative solutions to use our feedforward ONN for edge detection. First, we tried to reproduce similar configuration than with the bidirectional architecture. We do not differ the two output neurons and consider equal weights from input neurons to the two output neurons, and we initialize output neurons to $\{-1,-1\}$ HNN states or $\{0°,0°\}$ ONN phase states. We set weights from input neurons corresponding to opposite side of each edge with opposite weight values, as displayed in figure 7(A). It highlights that with this method, only half of the edges from the gray-scale octagon map are detected. Next, we exchange each weight values from previous configuration, so positive weights become negative, and negative weights become positive, see figure 7(B). In this case, we detect edges that were missing in the previous configuration. Thus, we deduce that by using one of the described configurations to connect input neurons to one output neuron, and the other configuration to connect input neurons to the second output neuron, we are able to retrieve all edges of the gray scale octagon map. Figure 7(C) shows results of the HNN Matlab emulator on the gray scale map image for this last optimal configuration. Precisions obtained with the digital ONN design on larger scale images, and resource and latency performances of the solution are described in section 4.

3.4. Extension to 5 × 5 and 7 × 7 ONN inputs

Convolutional filters configured for edge detection are often 3 × 3 kernels but can also be larger, with 5 × 5 kernels, or even 7 × 7 kernels. Thus, in this work, we explore if our edge detection methods with two-layer bidirectional and feedforward ONNs, can scale for larger input size. In particular, we extend our methods to perform image edge detection with ONN filters using input of 5 × 5 and 7 × 7 sizes, see figure 8. To extend our methods, we reproduce the main principle to have weights connected to central input neurons set to zero, and weights connected in neurons from the border respecting the rule: if two weights corresponds to the same edge, they have opposite values. And for the feedforward ONN, an additional rule is necessary: weights from the same input neuron to the two output neurons, need to have opposite values, see figure 8.

Note, in this paper we consider scanning the image with one-pixel stride. Thus, when we apply the 5 × 5 or 7 × 7 filters, we use the same method as with 3 × 3, we select a small part of the image (5 × 5 or 7 × 7), we apply our ONN for edge detection (bidirectional or feedforward architecture), and we apply the output information to the central pixel. Then, we move the filter to select the next part of the image, and so on. Thus, using larger filters we expected to detect more times each edges, which is confirmed in figure 8 with the

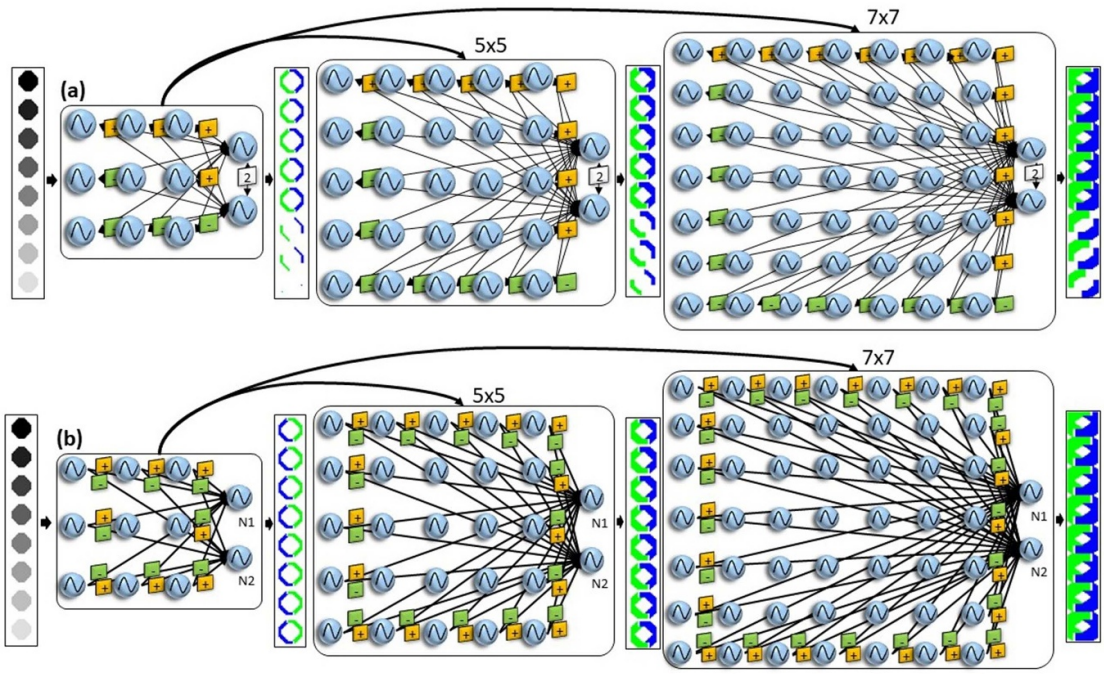

Figure 8. ONN configuration with 3 × 3, 5 × 5, and 7 × 7 input size, for (A) the bidirectional ONN architecture, and (B) the feedforward ONN architecture. Also results of the HNN Matlab emulator on the gray-scale octagon map are presented for each configuration.

HNN Matlab emulator on the gray scale octagon map. Results of the digital ONN designs with larger input sizes and resource utilization and timing performances are presented in section 4.

3.5. Evaluation of edge detection algorithms

Evaluation of edge detection algorithm is not an easy task as, to the best of our knowledge, there are no state-of-the-art evaluation metric, as well as no ground truth (GT) available. To first visually evaluate our edge detection solutions with our HNN emulator, we use a method from literature [37] which uses hexagonal forms with various gray levels on a white background. It allows to first assess if our solution is able to retrieve the simple edges from the simple image. However, we believe a more precise assessment of the performance is necessary to evaluate our ONN edge detection solutions. In literature, similarity coefficients are often used to assess precision of object detection algorithms. Here, we propose to use the Jaccard similarity coefficient (JSC) [38] to evaluate our solutions. JSC metric takes a GT and calculates the similarity between the GT and the obtained output information. To do so, it makes a ratio of the area of overlap between the two solutions, and the area of union. To apply it to edge detection algorithms, we consider the area of overlap as the number of overlapping edges, and the are of union the number of union edges between the two output images. Note, the JSC is computed considering similarity only between detected edges, and does not consider the full image with background.

$$JSC = \frac{Overlappingedges}{Unionedges} \tag{5}$$

As already mentioned, to the best of our knowledge, there is no GT for edge detection algorithms using our dataset. Thus, to evaluate performances of our solutions, we propose to use results from state-of-the-art edge detection algorithms as GT. Canny is known as the most precise solution for image edge detection, while Sobel is a cheaper and faster solution with less precision and more sensitivity to noise. Thus, we take Canny as GT and evaluate its similarity with Sobel and our solutions using the JSC metric. We use images generated with built-in Sobel and Canny Matlab functions.

4. Results

After exploring and validating methods to perform image edge detection using our two-layer ONNs with the HNN Matlab emulator, we validate and assess performances of the methods with the digital ONN design. We test both architectures on black and white 28 × 28 images from modified national institute of standards and

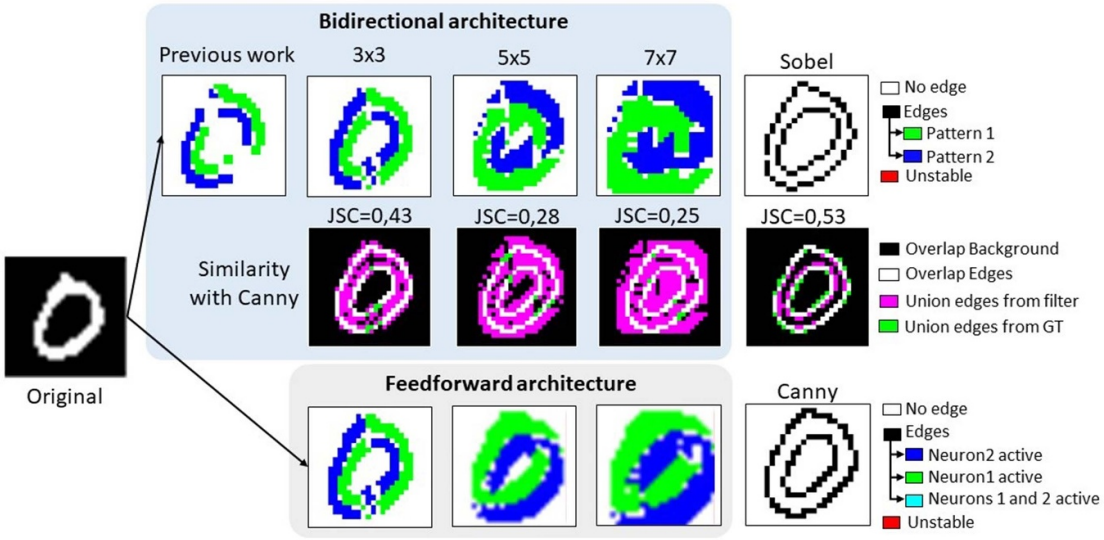

Figure 9. Edge detection results of the digital ONN designs for the bidirectional architecture with input size 3 × 3, 5 × 5, and 7 × 7, and for the feedforward architecture with input size 3 × 3, 5 × 5, and 7 × 7, compared with previous work [21] and state of the art Sobel and Canny on a black and white 28 × 28 MNIST image.

technology database (MNIST), gray-scale 28 × 28 MNIST images, as well as on large scale black and white images. For each architecture, we perform image edge detection using 3 × 3 input filters, 5 × 5 input filters, and 7 × 7 input filters, and we evaluate precision of each solution. Finally, we extract resource utilization and latency performances of each ONN architecture and size to assess real-time performances of our solutions.

4.1. Black and white 28 × 28 MNIST images

We first evaluate our two-layer ONNs for image edge detection using black and white images of handwritten digits from MNIST database [39]. The MNIST database contains 28 × 28 gray scale images, that we binarize to obtain black and white images.

Figure 9 displays an example of output images generated by the digital ONN design simulation for multi-scale bidirectional and feedforward architectures. Figure 9 also highlights outputs of previous work [21], and outputs of state-of-the-art Sobel and Canny configured with 3 × 3 kernels. Additionally, figure 9 illustrates the similarity between our ONN bidirectional filter and Sobel filter with GT Canny. It highlights that our customized weight solution for the 3 × 3 bidirectional architecture improves the number of detected edges from previous work [21]. Also, both 3 × 3 bidirectional architecture and 3 × 3 feedforward architecture have equal results. Comparing to state-of-the-art Sobel and Canny algorithms, our solution correctly detects all necessary edges. However, each edge is detected multiple times, creating larger lines. When increasing the input layer size, for both bidirectional and feedforward architectures, edges are detected even more times as we scan the image with one pixel stride. This creates large edge shapes, making it hard to visualize edges on a small 28 × 28 image. Figure 10 corroborates this visual assessment by comparing numerically our ONN feedforward and bidirectional filters and Sobel filter with Canny as GT. Figure 10 shows than for black and white MNIST images, the number of overlapping edges between our solutions and Canny are close to the overlapping edges between Sobel and Canny. However, the number of union edges is larger for our solutions, increasing with the filter size. It confirms that our solutions detect more edges than Sobel. Thus, the JSC between Sobel and Canny is higher than between our solutions and Canny as we detect more edges. Figures 9 and 10 have similar precision to state-of-the-art Sobel algorithm.

4.2. Gray scale 28 × 28 MNIST images

After investigating ONN for image edge detection on black and white MNIST images, we also analyze its efficiency on gray scale images. Figure 11 shows ONN output on a 28 × 28 gray scale MNIST image for the various architecture configurations. It also shows the similarity between our ONN feedforward filter and Sobel filter with GT Canny. First, we highlight that the bidirectional architecture, even with our customized weights does not allow to perform image edge detection on realistic gray scale images. The HNN Matlab emulator was able to detect edges on gray scale octagon map, however, the ONN digital design is not efficient on a realistic MNIST image. For various part of the image, the digital ONN system is unstable and never stabilizes to one of the patterns. However, the feedforward ONN architecture is able to correctly detect edges

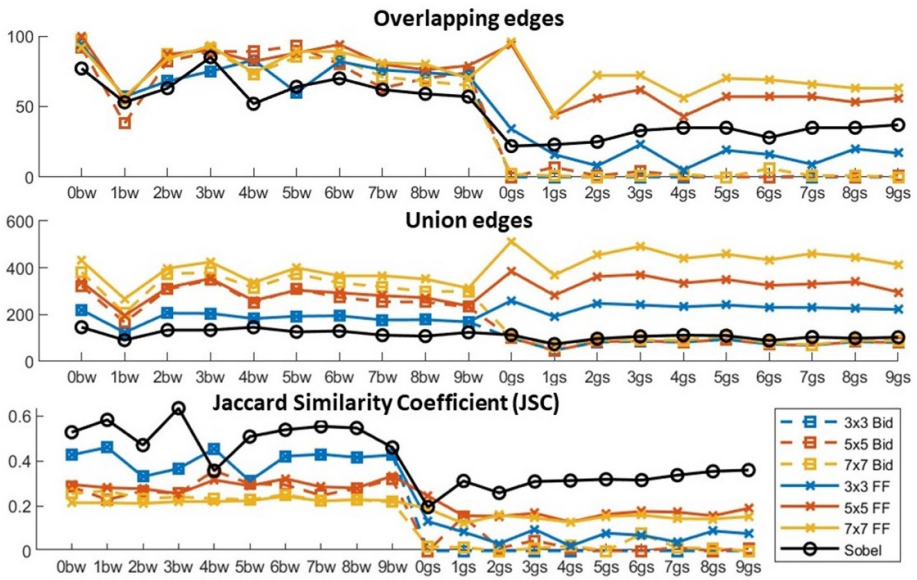

Figure 10. Number of overlapping edges, union edges, and similarity coefficient between our digital ONN solutions and Canny reference, and between Sobel state-of-the-art and Canny reference on MNIST black and white (bw) and grayscale (gs) digit images.

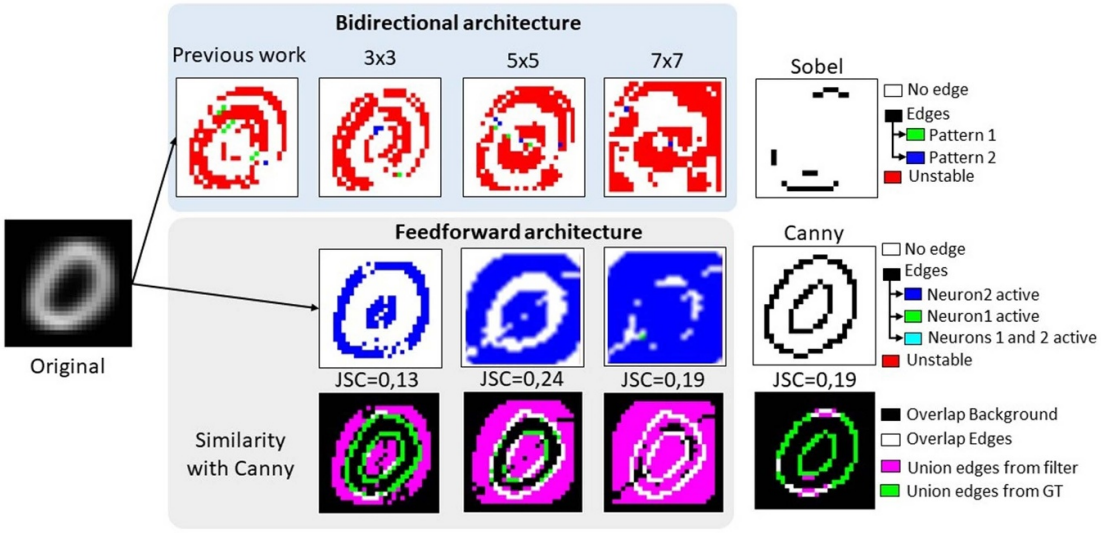

Figure 11. Edge detection results of the digital ONN designs for the bidirectional architecture with input size 3 × 3, 5 × 5, and 7 × 7, and for the feedforward architecture with input size 3 × 3, 5 × 5, and 7 × 7, compared with previous work [21] and state of the art Sobel and Canny on a gray scale 28 × 28 MNIST image.

on gray-scale images. In comparison to state-of-the-art Sobel and Canny, visually our feedforward ONN solution seems to perform better that Sobel, which misses a majority of edges from the image, but worst than Canny which still performs well on the gray-scale image. However, when computing the JSC with Canny as GT, Sobel gets better results, as shown in figure 10. It highlights that our feedforward ONN detects more edges that Sobel, however edges are not overlapping with Canny edges. Note that we observe the same behavior as before when increasing the input filter size. Each edge is detected more times, and edge lines become thicker. This behaviour increases a little bit the JSC as the number of overlapping edges increases with the number of detected edges. Using large-scale input with 5 × 5 or 7 × 7 pixels does not seem relevant for small-size images.

4.3. 512 × 512 standard black and white images

Finally, we test our solutions on 512 × 512 large scale standard black and white images. The digital ONN simulation process takes time, and to simulate 512 × 512 images, it is equivalent to simulate more than 260k times our ONNs for each part of the image. So, for the 3 × 3 ONNs, we simulated all 3 × 3 black and white

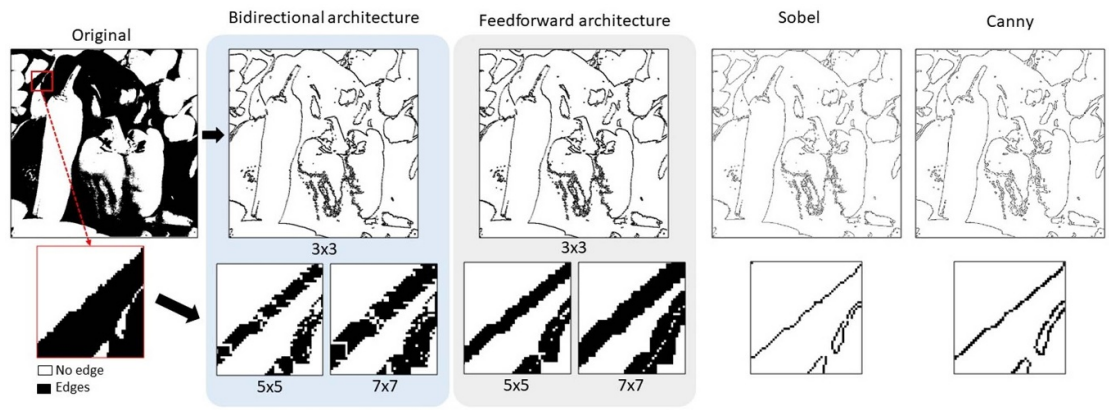

Figure 12. Edge detection results of the digital ONN designs for the bidirectional architecture with input size 3 × 3, 5 × 5, and 7 × 7, and for the feedforward architecture with input size 3 × 3, 5 × 5, and 7 × 7, compared with previous work [21] and state of the art on the black and white 512 × 512 standard 'pepper' image.

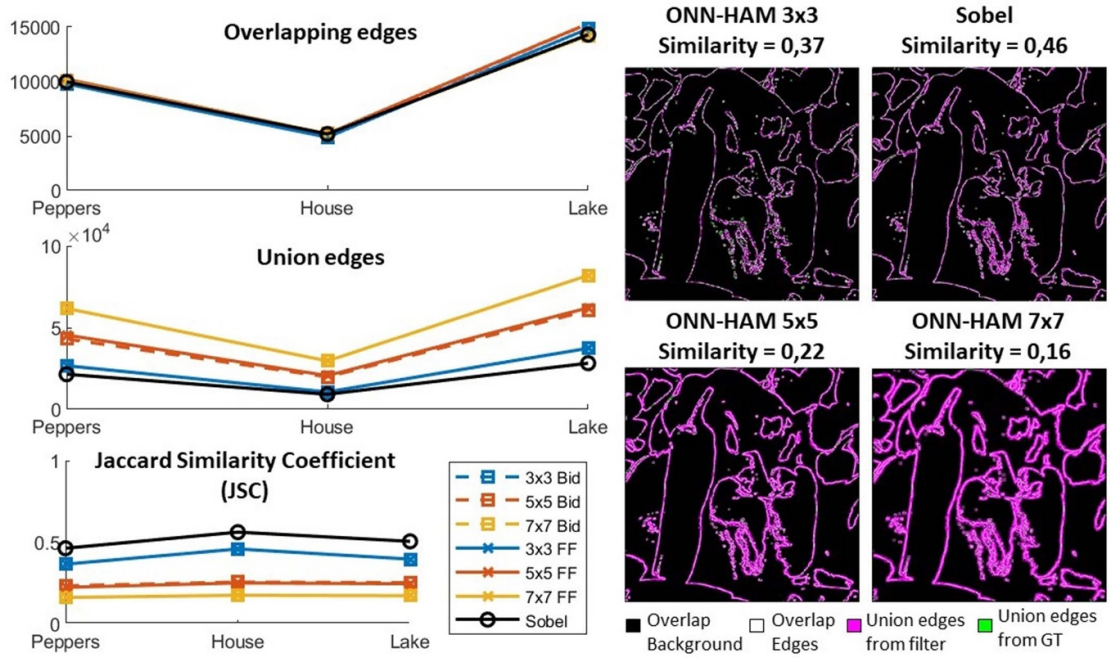

Figure 13. Number of overlapping edges, union edges, and similarity coefficient between our ONN solutions simulated on Matlab, Sobel state-of-the-art and Canny GT reference on large scale black and white images *Peppers*, *House*, and *Lake*. Similarity outputs of the bidirectional ONN solutions simulation on Matlab and Sobel solution compared to Canny on black and white *Peppers* image.

options (512 possible inputs) using our digital ONNs, and associated each part of the image with the corresponding ONN output. However, with 5 × 5 ONN input, the number of possibilities increases to more than 33 million, and for 7 × 7 it is even larger, becoming a challenge to simulate. For both the 5 × 5 and 7 × 7 options, we select a small part of the image and apply it as input of our ONNs edge detection algorithms.

Figure 12 shows the output of our ONNs on the standard image 'peppers' converted in black and white. This confirms the efficiency of our two ONN solutions with bidirectional and feedforward architecture to perform image edge detection on black and white images. It also supports the previous results showing an increase in the number of detected edges when increasing the input layer size. In addition, we use the HNN emulator to estimate the the JSC between our ONN edge detection filters on large scale black and white images, as shown in figure 13. It corroborates that our ONN edge detection filter is close to the Sobel state-of-the-art algorithm as it detects similar number of overlapping edges, but the higher number of union edges, increasing with the filter size, decreases the JSC when comparing similarity with Canny algorithm.

Table 1. ONN latency performances and resource utilization depending on the architecture and the size. The system is configured with 166 MHz frequency, allowing 2.7 MHz oscillation frequency.

	Bidirectional			Feedforward		
ONN architecture ONN size	3 × 3	5 × 5	7 × 7	3 × 3	5 × 5	7 × 7
Initialisation	24 ns	24 ns	24 ns	24 ns	24 ns	24 ns
Computation	1.42 μs	1.42 μs	1.42 μs	1.15 μs	1.15 μs	1.15 μs
LUTs (%)	484 (0.91)	1214 (2.28)	2369 (4.45)	211 (0.40)	302 (0.57)	457 (0.86)
Flip-Flops (%)	458 (0.43)	1026 (0.96)	1626 (1.53)	277 (0.26)	437 (0.41)	597 (0.56)

Table 2. Estimation of full image edge detection using our ONN architectures. The estimation multiplies the number of pixels to treat in each image by the time to initialize and compute each ONN design.

	Bidirectional			Feedforward		
ONN architecture ONN size	3 × 3	5 × 5	7 × 7	3 × 3	5 × 5	7 × 7
3 × 3 image	1.42 μs	1.42 μs	1.42 μs	1.15 μs	1.15 μs	1.15 μs
28 × 28 image	1.11 ms	1.11 ms	1.11 ms	0.78 ms	0.78 ms	0.78 ms
100 × 100 image	14.2 ms	14.2 ms	14.2 ms	11.5 ms	11.5 ms	11.5 ms
128 × 128 image	23.3 ms	23.3 ms	23.3 ms	18.8 ms	18.8 ms	18.8 ms
170 × 170 image	41 ms	41 ms	41 ms	33.2 ms	33.2 ms	33.2 ms
512 × 512 image	372 ms	372 ms	372 ms	301 ms	301 ms	301 ms

4.4. ONN performances

After validation of our methods efficiency to perform image edge detection, we extract ONN characteristics from simulations. Table 1 presents latency performances of the different ONN solutions (feedforward and bidirectional with various input sizes). Each ONN operates with a system frequency set to $F_{sys} = 166$ MHz, equivalent to an oscillation frequency of $F_o = 2.7$ MHz. It highlights that a 3 × 3 bidirectional ONN needs 1.42 μs in average to stabilize. Note, the computation time stays stable with the increase of the bidirectional ONN size, as ONN computes in parallel. The new parallel initialization implemented in both the bidirectional and feedforward ONN designs, drastically accelerates the initialization process compared to original fully-connected ONN design from [36], as it only needs one clock cycle to initialize all oscillators no matter the input size. Also, we highlight that, as expected, the computation is faster with the feedforward architecture than with the bidirectional architecture. The difference is minimal when considering only one ONN computation, but as shown in table 2 for a full image edge detection process, there is an important difference between the two architectures. For example, considering a 5 × 5 input size, the feedforward ONN can treat images up to 170 × 170 pixels in real-time (considering 25 to 30 images per second), while the bidirectional ONN can not treat such large images.

Table 1 also points out that the bidirectional architecture requires more resources than the feedforward architecture, regardless of the size. Yet, in general, the ONN design for image edge detection does not require a large amount of resources, with up to 4.5% of the look up tables (LUTs), and 1.36% of the Flip-Flops for the 7 × 7 bidirectional ONN. Thus, both ONN architectures can easily be integrated in larger systems with minimal resources.

5. Discussion

Assessing precision of edge detection algorithms is challenging. A common GT is needed for comparison. However, up to our knowledge, to date there is no edge detection database providing input and GT output. A first solution to counter the lack of GT is to compare to state-of-the-art algorithms. Another solution is to integrate the image edge detection solution in a more complex algorithm to check the global outputs of the algorithm. For example, some use edge detection as part of image segmentation applications [24], or in feature extraction applications [15]. It could be interesting to set up a more complex demonstrator to better assess the quality of our edge detection solutions. In this work, we evaluate our ONN edge detection algorithms by comparing the output of our solutions with other state-of-the-art edge detection algorithms, Sobel and Canny. In particular, we consider Canny as GT, as it is known to provide higher precision than Sobel, and compute the JSC for our ONN solutions and Sobel algorithm. We show that both the bidirectional and feedforward ONNs can detect most edges detected by Sobel and Canny algorithms on black and white images, even if edges are detected multiple times depending on the size. We also show that the efficiency of the bidirectional ONN solution is limited to black and white images, like Sobel, while the

Table 3. Performances of FPGA implementation of edge detection algorithms from literature.

		Resources				
	Filter size	LUTs	Flip-Flops	Hardware	Frequency	Latency 512 × 512
Sobel [40]	3 × 3	346	289	Xilinx Spartan 3 XC3S200	50 MHz	5.25 ms
Sobel [41]	3 × 3	47	107	Xilinx Spartan 3 XC3S50-5PQ20	204 MHz	1.28 ms
Canny [43]	3 × 3	82 496	40 640	Xilinx Virtex 5 XC5VSX240T	100 MHz	0.721 ms
Sobel [42]	3 × 3	0	114	Xilinx Spartan 6 XC6SLX43TQG144	504 MHz	0.52 ms

feedforward ONN can also address gray scale images, like Canny. In addition, an important point is that our methods are scalable with different ONN input size.

The latency is a critical point in image edge detection. Main algorithms use small convolutional filters to scan each part of the image. The sequential scanning is one of the main drawbacks of these methods, as well as for ours. For example, respecting real-time camera constraints of 25 to 30 images per second, we report that the 3 × 3 feedforward ONN can handle images of around 170 × 170 pixels while the bidirectional ONN can only treat smaller images, that is limited for image processing applications.

In literature, there are various references of FPGA implementation of Sobel and Canny algorithms. Mainly, two options are considered to accelerate the image scanning process, with one more adapted to Sobel, and the other more adapted to Canny. In [40–42], authors propose to simplify the Sobel convolutional computation in order to reduce the number of operations to be able to increase the frequency and speed up the process. For example, the Sobel FPGA architecture proposed in [40] can process each pixel gradient in a single clock cycle. Using a clock at 50 MHz, they are able to process a 512 × 512 image in around 5 ms. Using faster frequency, authors in [41] achieve the process of a 512 × 512 image in 1.28 ms, and with an even faster frequency, authors in [42] process the same-size image in less than 1 ms, as shown in table 3. For Canny, additional parallelization is necessary to achieve short processing. Authors in [43] propose to parallelize the process by blocks. They divide the image in numerous 64 × 64 non-overlapping blocks and process edge detection inside the blocks in serial, but edge detection of the different blocks in parallel. In this way, with a system running at 100 MHz, they can process a 512 × 512 image in less than 1 ms. However, as shown in table 3, it requires much more resources than the previous Sobel implementations.

We can see that our ONN solution is slower than the state-of-the-art FPGA implementations of both Sobel and Canny. As explained, Sobel implementations only need one clock cycle to process the gradient for one pixel and with the low amount of necessary resources, they can work at high frequency. Meanwhile, we need to wait for a few oscillation cycles depending on the architecture and our oscillation frequency is much slower than Sobel FPGA implementation system frequencies. Thus, as Canny we should consider including some parallelization to be competitive with Sobel or Canny in terms of Latency. Additionally, in [43], they also mention the overlapping parameter which can accelerate the process by scanning less time, this is another parameter we can play with in order to accelerate the image edge detection complete process. In table 4, we estimate latency to compute a 512 × 512 image for various overlapping parameters, depending on the number of parallel ONNs implemented for different ONN filter sizes and architectures. Note, we consider only two overlapping options, either a full overlapping (Y) considering a scanning stride of one pixel, or no overlapping at all (N) considering a scanning stride of the size of the filter such that there is no overlapping but no missing pixels as well. Also note, in order to assess the impact of the overlapping parameter on the edge detection precision, we computed the JSC between our ONN solutions simulated with the Matlab emulator and Canny algorithm on three large scale images (peppers, house, lake). Table 4 features the mean JSC value between the three images for each ONN solution. Finally, in table 4, we consider the number of parallel ONNs as the maximum number of parallel ONNs that can be implemented in the XC7Z020-1CLG400C FPGA chip used in our experiments. Table 4 shows that using a single ONN to sequentially scan an image is hardly competitive compared to state-of-the-art Sobel and Canny FPGA implementations. Only using a 7 × 7 filter with no overlapping option reaches a latency between 6 ms and 7 ms, depending on the architecture, similar to some Sobel implementations. However, precision reduces with the no-overlapping option, as well as with the size of the ONN filter. However, it is interesting to note that using parallel ONNs, with overlapping, the bidirectional ONN can process the entire 512 × 512 image in less than 10 ms, in particular, using 100 3 × 3 bidirectional ONNs in parallel, we can process the image in 3.7 ms, which is in the same range as the state-of-the-art Sobel and Canny implementations. Furthermore,

Table 4. Estimation of resource utilization and latency for various parallel and overlapping parameters for the two ONN architectures and various sizes.

				Resources			
	Filter size	Overlap	Parallel ONNs	LUTs	Flip-Flops	Latency 512 × 512	JSC between Canny/ONN
Bid	3 × 3	Y	1	484	458	372 ms	0.41
Bid	5 × 5	Y	1	1214	1026	372 ms	0.25
Bid	7 × 7	Y	1	2369	1626	372 ms	0.17
Bid	3 × 3	N	1	484	458	41.5 ms	0.23
Bid	5 × 5	N	1	1214	1026	14.8 ms	0.16
Bid	7 × 7	N	1	2369	1626	7.6 ms	0.13
Bid	3 × 3	Y	100	48 525	45 577	3.7 ms	0.41
Bid	5 × 5	Y	40	49 625	38 520	9.3 ms	0.25
Bid	7 × 7	Y	20	47 980	27 778	18.6 ms	0.17
Bid	3 × 3	N	100	48 525	45 577	0.42 ms	0.23
Bid	5 × 5	N	40	49 625	38 520	0.37 ms	0.16
Bid	7 × 7	N	20	47 980	27 778	0.38 ms	0.13
FF	3 × 3	Y	1	211	277	301 ms	0.41
FF	5 × 5	Y	1	302	437	301 ms	0.24
FF	7 × 7	Y	1	457	597	301 ms	0.17
FF	3 × 3	N	1	211	277	33.6 ms	0.23
FF	5 × 5	N	1	302	437	11.9 ms	0.16
FF	7 × 7	N	1	457	597	6.1 ms	0.13
FF	3 × 3	Y	290	53 125	82 727	1.04 ms	0.41
FF	5 × 5	Y	170	51 665	74 297	1.77 ms	0.24
FF	7 × 7	Y	110	51 724	61 616	2.74 ms	0.17
FF	3 × 3	N	290	53 125	82 727	0.12 ms	0.23
FF	5 × 5	N	170	51 665	74 297	0.07 ms	0.16
FF	7 × 7	N	110	51 724	61 616	0.06 ms	0.13

the feedforward ONN is even faster and requires less resources so we can implement more feedforward ONNs. Thus, using 290 3 × 3 feedforward ONNs in parallel, we can scan the 512 × 512 image in around 1 ms which is faster than some Sobel FPGA implementations, but slightly slower than the Canny FPGA implementation. Combining both non-overlapping option with parallel ONNs provides faster latency than reported Sobel and Canny FPGA implementations, however as explained the non-overlapping parameter impacts the edge detection precision. Additional study is necessary to assess precision of the overlapping parameter.

Future work will include the implementation of the solutions to assess real-time performances, with the various latency optimization methods. Additionally, this work uses ONN as filter to replace convolutional kernels. A possible future exploration could be to try replacing other types of convolutional filters than edge detection filters with ONN [44]. Finally, this work also introduces the feedforward ONN architecture. Future exploration will also investigate general learning methods adapted to the feedforward ONN and exploration of applications with this novel ONN architecture.

6. Conclusion

This paper presents two methods to perform efficient image edge detection using two-layer ONN architectures, one with bidirectional connections, and the other with feedforward connections. This is to date the first work to explore ONN with feedforward architecture. We validate and evaluate precision of our multi-scale edge detection methods with both the bidirectional and feedforward ONNs using HNN Matlab emulators, as well as digital ONN designs. We compute edge detection on black and white and gray scale images. We report a good precision, close to the state-of-the-art Sobel algorithm, of both bidirectional and feedforward methods on black and white images. However, only the feedforward ONN can perform efficient edge detection on realistic gray scale images. We also extract resource utilization and computation latency from the digital ONN simulations. We highlight that the feedforward ONN computes faster (in 1.15 μs) than the bidirectional ONN (in 1.42 μs), regardless of the size, leading to longer full image edge detection processing for the bidirectional ONN. For example, we estimate that the 3 × 3 feedforward ONN is able to treat images with up to 170 × 170 pixels while the bidirectional ONN can not treat such large images in real-time (considering 25–30 images per second). This work brings ONN as a possible image processing

solution, as it can replace convolutional filters in more complex algorithms, and perform feedforward computation that is often used in image processing tasks.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgment

This work is supported by the European Union's Horizon 2020 research and innovation program, EU H2020 NEURONN (www.neuronn.eu) project under Grant No. 871501.

ORCID iDs

Madeleine Abernot https://orcid.org/0000-0002-8267-2972
Todri-Sanial Aida https://orcid.org/0000-0001-8573-2910

References

[1] Xu D, Li T, Li Y, Su X, Tarkoma S, Jiang T, Crowcroft J and Hui P 2020 Edge intelligence: architectures, challenges, and applications (arXiv:2003.12172 [cs.NI])
[2] 2020 Beyond von Neumann *Nat. Nanotechnol.* **15** 507
[3] Christensen D V *et al* 2022 2022 Roadmap on neuromorphic computing and engineering *Neuromorph. Comput. Eng.* **2** 022501
[4] Schuman C D, Kulkarni S R, Parsa M, Mitchell J P, Date P and Kay B 2022 Opportunities for neuromorphic computing algorithms and applications *Nat. Comput. Sci.* **2** 10–19
[5] Maass W 1997 Networks of spiking neurons: the third generation of neural network models *Neural Netw.* **10** 1659–71
[6] Javanshir A, Nguyen T T, Parvez Mahmud M A and Kouzani A Z 2022 Advancements in algorithms and neuromorphic hardware for spiking neural networks *Neural Comput.* **34** 1289–328
[7] Izhikevich E M 2003 Simple model of spiking neurons *IEEE Trans. Neural Netw.* **14** 1569–72
[8] Orchard G, Frady E P, Rubin D B D, Sanborn S, Shrestha S B, Sommer F T and Davies M 2021 Efficient neuromorphic signal processing with Loihi 2 (*Coimbra, Portugal*) *2021 IEEE Workshop on Signal Processing Systems (SiPS)* pp 254–9
[9] Pehle C, Billaudelle S, Cramer B, Kaiser J, Schreiber K, Stradmann Y, Weis J, Leibfried A, Müller E and Schemmel J 2022 The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity *Front. Neurosci.* **16** 795876
[10] Linares-Barranco B, Sanchez-Sinencio E, Rodriguez-Vazquez A and Huertas J L 1992 CMOS analog neural network systems based on oscillatory neurons *IEEE Int. Symp. Circuits and Systems (ISCAS) (San Diego, CA, USA)* vol **5** pp 2236–9
[11] Jackson T, Pagliarini S and Pileggi L 2018 An oscillatory neural network with programmable resistive synapses in 28 nm CMOS *2018 IEEE Int. Conf. on Rebooting Computing (ICRC)* (McLean, VA: IEEE) pp 1–7
[12] Csaba G, Raychowdhury A, Datta S and Porod W 2018 Computing with coupled oscillators: theory, devices and applications *2018 IEEE Int. Symp. on Circuits and Systems (ISCAS)* (Florence: IEEE) pp 1–5
[13] Raychowdhury A, Parihar A, Smith G H, Narayanan V, Csaba G, Jerry M, Porod W and Datta S 2019 Computing with networks of oscillatory dynamical systems *Proc. IEEE* **107** 73–89
[14] Todri-Sanial A *et al* 2022 How frequency injection locking can train oscillatory neural networks to compute in phase *IEEE Trans. Neural Netw. Learn. Syst.* **33** 1996–2009
[15] Lowe D G 1999 Object recognition from local scale-invariant features *Proc. 7th IEEE Int. Conf. on Computer Vision* vol 2 pp 1150–7
[16] Shukla N, Tsai W-Y, Jerry M, Barth M, Narayanan V and Datta S 2016 Ultra low power coupled oscillator arrays for computer vision applications *2016 IEEE Symp. on VLSI Technology* pp 1–2
[17] Delacour C, Carapezzi S, Abernot M and Todri-Sanial A 2022 Energy-performance assessment of oscillatory neural networks based on VO2 devices for future edge AI computing TechRxiv (https://doi.org/10.36227/techrxiv.19248446.v1)
[18] Hölzel R W and Krischer K 2011 Pattern recognition with simple oscillating circuits *New J. Phys.* **13** 073031
[19] Kumar A and Mohanty P 2017 Autoassociative memory and pattern recognition in micromechanical oscillator network *Sci. Rep.* **7** 411
[20] Hopfield J J 1982 Neural networks and physical systems with emergent collective computational abilities *Proc. Natl Acad. Sci.* **79** 2554–8
[21] Abernot M, Gil T and Todri-Sanial A 2022 Oscillatory neural network as hetero-associative memory for image edge detection *Neuro-Inspired Computational Elements Conf. (NICE)* (New York: Association for Computing Machinery) pp 13–21
[22] Real F D and Berry Fçois 2010 *Smart Cameras: Technologies and Applications* (Boston, MA: Springer) (https://doi.org/10.1007/978-1-4419-0953-4_3)
[23] Dhar P, Guha S, Biswas T and Abedin M Z 2018 A system design for license plate recognition by using edge detection and convolution neural network *2018 Int. Conf. on Computer, Communication, Chemical, Material and Electronic Engineering (IC4ME2)* pp 1–4
[24] De A and Guo C 2014 An image segmentation method based on the fusion of vector quantization and edge detection with applications to medical image processing *Int. J. Mach. Learn. Cyber.* **5** 543–51
[25] Sobel I and Feldman G 1968 *A 3 × 3 isotropic gradient operator for image processing.* A Talk at the Stanford Artificial Intelligence Project
[26] Canny J 1986 A computational approach to edge detection *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-8** 679–98
[27] Schwemmer M A and Lewis T J 2012 The theory of weakly coupled oscillators *Phase Response Curves in Neuroscience* ed N W Schultheiss, A A Prinz and R J Butera (New York: Springer) pp 3–31
[28] Winfree A T 1967 Biological rhythms and the behavior of populations of coupled oscillators *J. Theor. Biol.* **16** 15–42
[29] Fell J and Axmacher N 2011 The role of phase synchronization in memory processes *Nat. Rev. Neurosci.* **12** 105–18

[30] Delacour C and Todri-Sanial A 2021 Mapping hebbian learning rules to coupling resistances for oscillatory neural networks *Front. Neurosci.* **15** 694549
[31] Morris R G M Hebb: D O 1999 The organization of behavior, Wiley: New York; 1949 *Brain Res. Bull.* **50** 437
[32] Delacour C, Carapezzi S, Abernot M, Boschetto G, Azemard N, Salles J, Gil T and Todri-Sanial A 2021 Oscillatory neural networks for edge AI computing *2021 IEEE Computer Society Annual Symp. on VLSI (ISVLSI)* pp 326–31
[33] Wang D and Terman D 1994 Locally excitatory globally inhibitory oscillator networks: theory and application to pattern segmentation *Proc. IEEE Workshop on Neural Networks for Signal Processing* (*Ermioni*: IEEE) pp 136–45
[34] Kosko B 1988 Bidirectional associative memories *IEEE Trans. Syst. Man Cybern.* **18** 49–60
[35] Jackson T C, Sharma A A, Bain J A, Weldon J A and Pileggi L 2015 Oscillatory neural networks based on TMO nano-oscillators and multi-level RRAM cells *IEEE J. Emerg. Sel. Top. Circuits Syst.* **5** 230–41
[36] Abernot M, Gil T, Jiménez M, Núñez J, Avellido M J, Linares-Barranco B, Gonos T, Hardelin T and Todri-Sanial A 2021 Digital implementation of oscillatory neural network for image recognition applications *Front. Neurosci.* **15** 713054
[37] Ramalho M A and Curtis K M 1996 Neural network arbitration for edge detection *Proc. 3rd Int. Conf. on Electronics, Circuits and Systems* vol 2 pp 1112–15
[38] Chung N, Miasojedow B, Startek M and Gambin A 2019 Jaccard/Tanimoto similarity test and estimation methods for biological presence-absence data *BMC Bioinform.* **20** 644
[39] LeCun Y, Boser B, Denker J S, Henderson D, Howard R E, Hubbard W and Jackel L D 1989 Backpropagation applied to handwritten zip code recognition *Neural Comput.* **1** 541–51
[40] Guo Z, Xu W and Chai Z 2010 Image edge detection based on FPGA *2010 9th Int. Symp. on Distributed Computing and Applications to Business, Engineering and Science* pp 169–71
[41] Halder S, Bhattacharjee D, Nasipuri M and Basu D K 2012 A fast FPGA based architecture for sobel edge detection *Progress in VLSI Design and Test* (Berlin: Springer) pp 300–6
[42] Nausheen N, Seal A, Khanna P and Halder S 2018 A FPGA based implementation of sobel edge detection *Microprocess. Microsyst.* **56** 84–91
[43] Xu Q, Varadarajan S, Chakrabarti C and Karam L J 2014 A distributed canny edge detector: algorithm and FPGA implementation *IEEE Trans. Image Process.* **23** 2944–60
[44] Nikonov D E, Kurahashi P, Ayers J S, Li H, Kamgaing T, Dogiamis G C, Lee H-J, Fan Y and Young I A 2020 Convolution inference via synchronization of a coupled CMOS oscillator array *IEEE J. Explor. Solid-State Comput. Devices Circuits* **6** 170–6