Digital oscillatory neural network implementation on FPGA for edge artificial intelligence applications and learning

Madeleine Abernot

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITE DE MONTPELLIER

En Systèmes Automatiques et Microélectroniques

École doctorale : Information, Structures, Systèmes

Unité de recherche UMR5506

Digital oscillatory neural network implementation on FPGA for edge applications and learning

Présentée par Madeleine Abernot

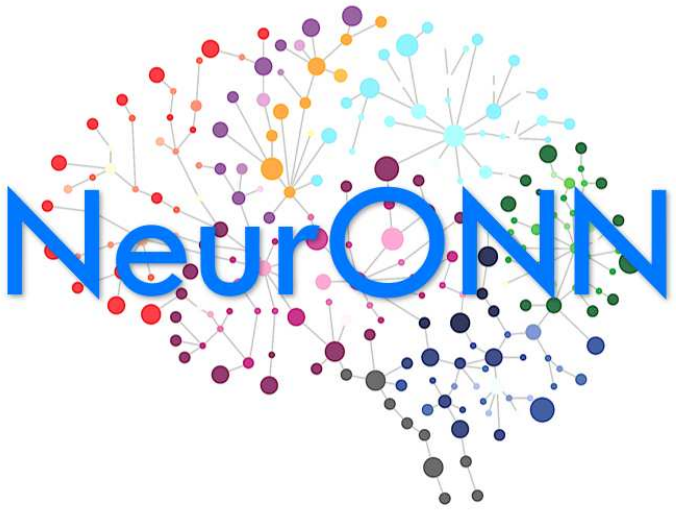
Le 18 Décembre 2023

Sous la direction de Aida Todri-Sanial et Nadine Azemard-Crestani

Devant le jury composé de

Haralampos Stratigopoulos, Directeur de recherche, LIP6, CNRS	Président du jury
Lorena Anghel, Professeur, Grenoble INP	Examinateur
Maria José Avedillo, Professeur, Institut de microélectronique de Séville	Examinateur
Gyorgy Csaba, Professeur, Université catholique Pazmany Peter	Examinateur
Nadine Azemard-Crestani, Chargé de recherche, LIRMM, CNRS	Co-directrice de thèse
Sylvain Saighi, Professeur, IMS, Université de Bordeaux	Rapporteur
Catherine D. Schuman, Maître de conférence, Université de Tennessee	Rapporteur
Aida Todri-Sanial, Professeur, Université technologique de Eindhoven	Directrice de thèse
Théophile Gonos, Président, A.I.Mergence	Invité

UNIVERSITÉ DE MONTPELLIER



Digital oscillatory neural network implementation on FPGA for edge applications and learning

Madeleine Abernot

Supervized by:

Aida Todri-Sanial
Nadine Azemard-Crestani

A thesis submitted in fulfillment of the requirements
for the degree of Doctor of Philosophy in the

SYAM - I2S

ACKNOWLEDGEMENTS

This thesis would never have been completed without the contribution of many entities.

I would like to first thank my advisors, Aida Todri-Sanial and Nadine Azémard-Crestani for your support and encouragement along the journey. A particular thanks to Aida, who followed me patiently from the beginning and always motivated me to become a better researcher, teaching me scientific, management, and communication skills.

I would like to express my gratitude to my thesis committee, Pr. Sylvain Saïghi, Dr. Catherine D. Schuman, Pr. Lorena Anghel, Pr. María José Avedillo, Dr. Haralampos Stratigopoulos, Pr. Gyorgy Csaba, and Dr. Théophile Gonos for your engagement, time, and insightful comments.

A special thanks to my colleagues from the NeurONN project, in particular Manuel Jimenez, Dr. Juan Nuñez, Pr. María José Avedillo, Sylvain Gauthier, Dr. Théophile Gonos, and Dr. Siegfried Karg. It has been a pleasure to learn and collaborate with you.

I am sincerely grateful to my excellent colleagues from the LIRMM. A major thanks to Thierry Gil who motivated me from the beginning of my Ph.D. journey, helped me build real demonstrators, and taught me rigor and method.

I thank my colleagues and friends in and around the corridor who have brightened my days for three years. Thank you Corentin Delacour for sharing this 3-year experience with me. I enjoyed working with you and I learned a lot from you, you helped me get started on many neuromorphic aspects. More than that, you have been a great emotional support. Thank you Stefania Carapezzi, you have been a really supportive office-mate, sharing all your knowledge and bringing incredible food support. Thank you Gabriele Boschetto for sharing your research experience and your gastronomic passion. Thank you Eirini Karachristou for your help in managing and organizing the NeurONN project, and for your willingness to create a cohesive and friendly team. Thank you Camille Couralet for your everyday support and for your unfailing energy for laughing, running, dancing, etc. Thank you Cécile Romane for sharing your tips on thesis and post-thesis management. Thank you Adrien Suau for your kind welcoming and advice during this thesis. Thank you Siyuan Niu for transmitting your Ph.D. experience and for being a great Tuesday's partner for the African food truck. Thank you Mr. Lee for taking care of all my Tuesday lunches.

I also would like to thank Ana Tacuri for her help in understanding the laboratory and organizing missions around the world. Thank you Virginie Feche for your positivity, and for your nutritional support in organizing breakfasts. Thank you Elena Demchenko, Paul Leloup, and Jérémie Salles for bringing me back to running, so I could take the lead of the breakfasts. Thank you as well to the entire team RA, Thomas, Julien, Loic, Pierre, Sarah, Quentin, Ismael, and others, who helped me get through the thesis writing. A major thanks to Geneviève Carrière for taking care of me and my health every day during those three years.

Now, I would like to show my gratitude to my family, starting with my parents Yves and Florence Abernot. Thank you for always helping and supporting me in achieving my ambitions. Thank you as well for your moral support every time I needed it and for your sportive motivation with windsurfing and yoga courses. I also thank my brother, my sister, and their partners, Jeanne, Samuel, Damien, and Appoline for the example you set by doing plainly what you like. Thank you Sam for keeping me up to date on musical trends, and thank you Jeanne for your traditional dance training. I also thank Réglisse. After 20 years, you are still sharp and alive to comfort the family. I finally express my gratitude to my grandparents for sharing their life experiences and for all the happy moments we had and will have.

I also would like to thank my friends, who accompanied me during the thesis. First, I would like to thank my childhood friends from Saint-Brieuc. Thank you Séléna and Julie. Meeting you for a meal, a drink, a swim, or whatever when I come back to Brittany brightens my days. Thank you, Louisa and Lucas, it is always a pleasure to share a *fondue* with a glass of wine with you in Montpellier or in Lyon. Thank you, Clotilde, Théo, François, Chloé, and Alice, it is always a pleasure to catch up. Thank you, Marie, I am really grateful that we managed to stay in contact despite the distance.

Then, I would like to thank the friends I met during my studies in Rennes and Bordeaux. Thank you, Lenny and Gautier, I am really happy to see that after Rennes we took different paths which are in the end similar and converging. Thank you, Matthieu, Sylvain, Christian, Baptiste, Clément, and Emmanuelle. We had a great time in Bordeaux, and I am really sad that we can not stay in touch more often. Thank you, Thomas, Clotilde, Etienne, Amandine, Matthieu, Lucie, and Antoine. Thank you for kindly welcoming me into your friendly group. It is always a pleasure to share moments with you everywhere in France and Europe. Thank you Félix and Axelle. I enjoy climbing, hiking, or simply conversing with you.

Now, I would like to thank all the people I met lately in Montpellier. Thank you Nizar, Marian, and Antonin for the nice dinners and board-game parties. Thank you Alexia and Benoit for teaching me (or trying to teach me) the science of bread, beer, knitting, windsurfing, and more. Thank you Aurélien, Camille, Mara, Yentl, Hyppolite, Hervé, Ioannis, Juliette, and Thomas for the nice climbing days and party nights. Thank you Alizée. We met first in Saint-Brieuc but re-discovering you in Montpellier has strengthened our friendship. I thank you for always being here to listen when I need to share my frustrations, to go hiking or biking when I need some fresh air, and to go out drinking and dancing when I need to let off steam. Also, I am sincerely thankful to the (extended) family of Thibaud. Thank you, Gilles and Caroline, for your warm welcomes and hospitality during weekends and holidays. Thank you, Julien, Clara, Cécile, and Bastien, for the nice and funny moments we had and still have. Thank you, Catherine, Jean, and Raymonde for sharing your life experiences and I wish you *bien des choses*.

Finally, a special thanks to Thibaud. You have supported my joys, my frustrations, and my clumsiness every day for more than six years. I would like to especially thank you for your help in maintaining a healthy lifestyle, taking care of most cleaning, cooking, gardening, and washing machine household tasks, and encouraging me to keep up with social, cultural, and sporting activities. I would have never achieved this 3-year experience without your sense of humor, your talents for sewing and decorating, and of course your emotional support.

This thesis work has been supported by the European Union's Horizon 2020 research and innovation program, EUH2020 NEURONN (www.neuronn.eu) project under Grant No. 871501.

RÉSUMÉ

Au cours des dernières décennies, la multiplication des objets embarqués dans de nombreux domaines a considérablement augmenté la quantité de données à traiter et la complexité des tâches à résoudre, motivant l'émergence d'algorithmes probabilistes d'apprentissage tels que l'intelligence artificielle (IA) et les réseaux de neurones artificiels (ANN). Cependant, les systèmes matériels pour le calcul embarqué basés sur l'architecture von Neuman ne sont pas efficace pour traiter cette quantité de données. C'est pourquoi des paradigmes neuromorphiques dotés d'une mémoire distribuée sont étudiés, s'inspirant de la structure et de la représentation de l'information des réseaux de neurones biologiques. Dernièrement, la plupart de la recherche autour des paradigmes neuromorphiques ont exploré les réseaux de neurones à impulsion ou *spiking neural networks* (SNNs), qui s'inspirent des impulsions utilisées pour transmettre l'information dans les réseaux biologiques. Les SNNs encodent l'information temporellement à l'aide d'impulsions pour assurer un calcul de données continues naturel et à faible énergie. Récemment, les réseaux de neurones oscillatoires (ONN) sont apparu comme un paradigme neuromorphique alternatif pour du calcul temporel, rapide et efficace, à basse consommation. Les ONNs sont des réseaux d'oscillateurs couplés qui émulent les propriétés de calcul collectif des zones du cerveau par le biais d'oscillations. Les récentes implémentations d'ONN combinées à l'émergence de composants compacts à faible consommation d'énergie encouragent le développemment des ONNs pour le calcul embarqué. L'état de l'art de l'ONN le configure comme un réseau de Hopfield oscillatoire (OHN) avec une architecture d'oscillateurs entièrement couplés pour effectuer de la reconnaissance de formes avec une précision limitée. Cependant, le grand nombre de synapses de l'architecture limite l'implémentation de larges ONNs et le champs des applications de l'ONN. Cette thèse se concentre pour étudier si et comment l'ONN peut résoudre des applications significatives d'IA embarquée à l'aide d'une preuve de concept de l'ONN implémenté en digital sur FPGA. Tout d'abord, ce travail explore de nouveaux algorithmes d'apprentissages pour OHN, non supervisé et supervisé, pour améliorer la précision et pour intégrer de l'apprentissage continu sur puce. Ensuite, cette thèse étudie de nouvelles architectures pour l'ONN en s'inspirant des architectures en couches des ANNs pour créer dans un premier temps des couches d'OHN en cascade puis des réseaux ONN multi-couche. Les nouveaux algorithmes d'apprentissage et les nouvelles architectures sont démontrées avec l'ONN digital pour des applications d'IA embarquée, telles que pour la robotique avec de l'évitement d'obstacles et pour le traitement d'images avec de la reconnaissance de formes, de la détection de contour, de l'extraction d'amers, ou de la classification.

Abstract

In the last decades, the multiplication of edge devices in many industry domains drastically increased the amount of data to treat and the complexity of tasks to solve, motivating the emergence of probabilistic machine learning algorithms with artificial intelligence (AI) and artificial neural networks (ANNs). However, classical edge hardware systems based on von Neuman architecture cannot efficiently handle this large amount of data. Thus, novel neuromorphic computing paradigms with distributed memory are explored, mimicking the structure and data representation of biological neural networks. Lately, most of the neuromorphic paradigm research has focused on spiking neural networks (SNNs), taking inspiration from signal transmission through spikes in biological networks. In SNNs, information is transmitted through spikes using the time domain to provide a natural and low-energy continuous data computation. Recently, oscillatory neural networks (ONNs) appeared as an alternative neuromorphic paradigm for low-power, fast, and efficient time-domain computation. ONNs are networks of coupled oscillators emulating the collective computational properties of brain areas through oscillations. The recent ONN implementations combined with the emergence of low-power compact devices for ONN encourage novel attention over ONN for edge computing. State-of-the-art ONN is configured as an oscillatory Hopfield network (OHN) with fully coupled recurrent connections to perform pattern recognition with limited accuracy. However, the large number of OHN synapses limits the scalability of ONN implementation and the ONN application scope. The focus of this thesis is to study if and how ONN can solve meaningful AI edge applications using a proof-of-concept of the ONN paradigm with a digital implementation on FPGA. First, it explores novel learning algorithms for OHN, unsupervised and supervised, to improve accuracy performances and to provide continual on-chip learning. Then, it studies novel ONN architectures, taking inspiration from state-of-the-art layered ANN models, to create cascaded OHNs and multi-layer ONNs. Novel learning algorithms and architectures are demonstrated with the digital design performing edge AI applications, from image processing with pattern recognition, image edge detection, feature extraction, or image classification, to robotics applications with obstacle avoidance.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of acronyms

AAM	auto-associative memory.
AI	artificial intelligence.
ANN	artificial neural network.
BAM	bidirectional associative memory.
BNN	binary neural network.
BPTT	back-propagation through time.
CHR	contrastive Hebbian rule.
CNN	convolutional neural network.
COP	combinatorial optimization problem.
CPU	central processing unit.
DLP	deep learning processor.
DNN	deep neural network.
DoG	difference of Gaussian.
DOI	Diederich-Opper rule I.
DOII	Diederich-Opper rule II.
EP	equilibrium propagation.
FF	feed-forward.
FPGA	field-programmable gate array.
FPS	frames per second.
FSM	finite state machine.
GPU	graphics processing unit.
GT	ground truth.
HAM	heterogeneous associative memory.
HD	Hamming distance.
HNN	Hopfield neural network.
I^2C	inter-integrated circuit.
JSC	Jaccard similarity coefficient.
LEGION	locally excitatory globally inhibitory oscillator network.
LIF	leaky integrate and fire.
LUT	look-up table.
MAC	multiply and accumulate.
MG	majority gate.
ML	machine learning.
MLP	multi-layer perceptron.
$NMAC_{OP}$	number of multiply and accumulate operations.
NPU	neural processing unit.
OAV	obstacle avoidance.
OHN	oscillatory Hopfield network.
OIM	oscillatory Ising machine.
ONN	oscillatory neural network.
ORB	oriented FAST and rotated BRIEF.

PL	programmable logic.
PLL	phase-locked loop.
PP	post-processing.
PS	processing system.
ReLU	rectified linear unit.
RNN	recurrent neural network.
SIFT	scale-invariant feature transform.
SLAM	simultaneous localization and mapping.
SNN	spiking neural network.
SoC	system on chip.
STDP	spike timing dependent plasticity.
STO	spin-torque oscillator.
SURF	speeded up robust features.
ToF	time of flight.
TPU	tensor processing unit.
UART	universal asynchronous receiver transmitter.

CHAPTER 1

INTRODUCTION

Contents

In the last decades, data growth has led to the emergence of increasingly powerful artificial intelligence (AI) algorithms. However, the state-of-the-art edge computing systems based on von Neuman architecture can not efficiently handle this large amount of data and calculations. Thus, novel brain-inspired neuromorphic computing paradigms emerged to propose low-power fast computing systems for edge AI. In particular, the oscillatory neural network (ONN) neuromorphic paradigm takes inspiration from brain waves to compute using the natural synchronization of coupled oscillators. ONNs have mainly been used to perform pattern-recognition tasks which is limiting for edge AI. In this thesis, we study how to perform meaningful AI edge applications with the ONN computing paradigm. In particular, we use a proof-of-concept of the ONN computing paradigm with a digital design implemented on field-programmable gate array (FPGA) to explore various learning algorithms, architectures, and edge applications compatible with ONN. This chapter first provides a description of state-of-the-art AI algorithms and hardware computing platforms before presenting neuromorphic computing paradigms, especially ONN to state the main motivation of this Ph.D. thesis. Finally, this chapter details the main contributions and outline of this Ph.D. thesis.

1.1 Artificial Intelligence and Artificial Neural Networks

The development of large-scale micro-electronic processors, following Moore's law prediction [1], has led to the development of novel computing algorithms to bring human capabilities to machines. We define AI as the ensemble of algorithms and models solved by machines that try to either overcome human capabilities to solve a specific task or to replace humans on more general-purpose tasks. With the increasing complexity of AI tasks to solve, deterministic computing algorithms were discontinued for the emergence of probabilistic machine learning (ML) models. ML models are a subset of AI algorithms that are capable of learning tasks from data using an objective function, and a learning algorithm capable of optimizing the objective

function [2]. ML models include broad types of algorithms, however, they are more and more associated with the subset of artificial neural network (ANN) models.

ANNs are computing models taking inspiration from neuroscience and brain neural network structure to learn and compute [3]. The human brain is known to compute hard tasks while consuming a low amount of power [4], making it attractive for efficient computing. It is composed of a biological neural network that treats and transmits information efficiently. Concisely, a biological neural network is made of neurons interconnected with synapses by axons and dendrites. The human neural network is made of around 86 billion neurons interconnected by even more synapses, around 10000 billion in 1 cm^3. The efficiency of the human brain mainly comes from its scalability and its plasticity, making it highly reconfigurable to adapt its knowledge through time.

ANNs take inspiration from biological neural networks by considering two main elements, artificial neurons interconnected by artificial synapses. The first mathematical model of a biological neuron was proposed by McCulloch and Pitts in 1943 [5] before being used for computation, as a perceptron, by Rosenblatt in 1960 [6]. It is defined by two functions:

1. an **integration function** that integrates the synaptic information. The first integration function proposed by Rosenblatt [6] computes a weighted sum of the pre-synaptic neuron states and synaptic weight factors and is still widely used nowadays, see Figure 1.1a,
2. an **activation function** that computes the integrated information to activate or not the neuron output. The first proposition of an activation function was a $sign$ function, see Figure 1.1a.

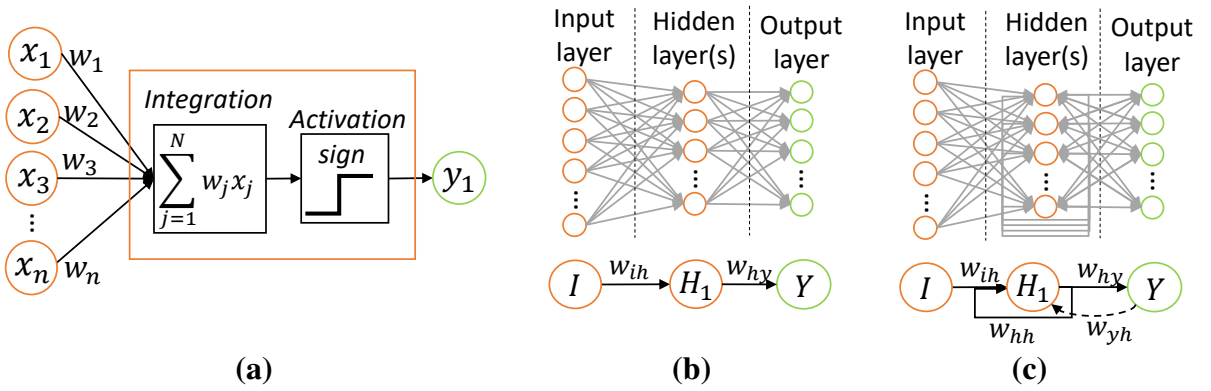


FIGURE 1.1 – *(a) Perceptron neuron. (b) Feed-forward DNN architecture. (c) Recurrent DNN architecture.*

A single perceptron neuron with a *sign* activation function does not allow non-linear transformation between input and output data, limiting its application to linear tasks [7]. Thus, since the introduction of the perceptron, novel ANN models have been proposed, modifying neuron functions and proposing novel architectures. First, novel non-linear activation functions offering a wider possibility of output states were developed to improve ANN performances. Particularly, the sigmoid function and the rectified linear unit (ReLU) function are nowadays typical activations [8]. Then, novel ANN topologies or architectures were introduced creating layers of neurons in which there is no synaptic connection within layers, but there are synaptic connections between layers. Synapses can be unidirectional, transmitting information from input to output layers, and producing feed-forward models, see Figure 1.1b. Otherwise, synapses can be multi-directional creating recurrences between layers or between neurons from the same

layer, producing recurrent models, see Figure 1.1c. If a network architecture is made of more than one layer (feed-forward or recurrent), it is called a deep neural network (DNN). A feed-forward DNN of perceptron neurons is called a multi-layer perceptron (MLP) and produces a non-linear relation between input and output, enlarging the complexity of tasks to solve with ANNs.

Image classification is a typical non-linear AI task used in a wide variety of domains, such as for smart city security, healthcare, or autonomous driving. However, even if MLPs offer distinctive performances on small-scale image classification [9], they can not easily handle large-scale images. In image classification, the input data is represented by the intensity of each pixel in the image. For example, for a 28x28 gray-scale image, the network requires an input layer of 784 neurons, which is multiplied by three for RGB images. The large input layer increases computation and memory requirements making it challenging to perform large-scale image classification with MLPs. Convolutional Neural Networks (CNNs) [10] were introduced to solve the scaling issue of MLP models on image classification, taking inspiration from the visual nervous system of the vertebrate by creating a hierarchical model [11, 12]. The first solution to reproduce the visual nervous system was the Neocognitron proposed by Fukushima in 1980 [13] before the emergence of CNNs developed by LeCun in the 1990s [9, 10]. A CNN is made of two main components, convolutional layers which are sparse layers extracting features from the original image, and a final fully connected MLP layer performing classification from the extracted features, thus limiting input layer size. Additional operations, like Maxpooling and Normalization, can also be used to analyze and select important features in order to reduce the input dimension of the MLP classifier. The architecture of the CNN, the number of convolutional layers, Maxpooling, and Normalization can vary depending on the task to solve, and finding the best architecture is still an open question [14]. In 2009, the ImageNet dataset was proposed to benchmark ANN models on image classification [15] and various CNN-based models achieved high precision [16, 17, 18]. However, CNN models are efficient for static input data like images, but not optimized for dynamical data.

Recurrent Neural Networks (RNNs) [19] are networks that include some recurrences in their topology by connecting neurons among the same layer or allowing connections from neurons of upper layers to neurons of previous layers, such that the information is propagating in different directions, see Figure 1.1c. Recurrences allow remembering previous information of sequential data, creating dependencies. Thus, RNNs are often used for dynamical data processing, but also for clustering tasks. For example, the first RNN model was the Hopfield neural network (HNN) [20] proposed by Hopfield in 1982 with a single-layer architecture to perform auto-associative memory (AAM). However, classical RNNs are slow to train and suffer from vanishing gradient problems, forgetting old dependencies for long input sequences [21]. Recently RNNs gained interest in natural language processing [22] with the introduction of Long Short Term Memories (LSTMs) [23] and the emergence of the attention mechanism [24], reducing the vanishing gradient problem. For example, the attention mechanism is behind the powerful Generative Pre-trained Transformer (GPT) models capable of generating human-like text or images [25, 26].

Novel ANN architectures are still being investigated in order to improve performances and adapt to various data types. In parallel, ANN learning algorithms are also explored to correctly configure the ANN models by adapting the synaptic parameters, mainly the weight factors, to the assigned tasks. To do so, ANN learning algorithms use a dataset, an objective function, and an optimization algorithm capable of optimizing the objective function. Learning algorithms are divided into two main categories:

1. **Supervised learning** considers datasets containing data samples with correct associated

outputs to perform classification or regression tasks, associating a prediction or a label to input data.

2. **Unsupervised learning** considers datasets containing input samples without the associated prediction and needs to find correlations and structure to organize data without additional feedback indicating if the organization is correct or not. It is often used for clustering.

The state-of-the-art **supervised learning** is the gradient back-propagation algorithm introduced in the 1980s for feed-forward ANNs [27, 9]. It calculates the error between the ANN prediction and the label to propagate the gradient to the previous layers and update the synaptic parameters. The introduction of back-propagation with the parallel developments of ANN architectures revolutionized the use of DNNs and CNNs, introducing deep learning. However, deep learning usually compels a large learning dataset, resulting in an important number of computations. For example, image classification of handwritten digits with MNIST achieves more than 99% of precision but requires training over 60000 samples [9]. Back-propagation has also been adapted to RNN with the back-propagation through time (BPTT), computing gradient back-propagation for multiple time steps, but it is even more computation-demanding than the usual back-propagation algorithm. The first **unsupervised learning** algorithm is the Hebbian learning introduced by Hebb in 1949 [28]. It uses learning data to reinforce synaptic parameters locally between two neurons if they have equal activation values, following "*neurons that fire together, wire together*". Unsupervised learning algorithms usually require fewer learning samples and fewer computations while it also achieves lower precision than supervised algorithms. Recently, unsupervised learning algorithms gained interest with the emergence of self-supervised learning [29, 30] used for limited or corrupted datasets to label additional or missing data using unsupervised learning before configuring the ANN model with supervised learning.

Supervised and unsupervised learning algorithms are commonly computed only once with the entire learning dataset before using the configured model for an infinite inference loop. However, in the case of evolving environments or data, it is necessary to learn redundantly or continuously through time to keep the ANN configuration optimized. Continual learning is another area of research which integrates supervised and unsupervised algorithms [31, 32, 33] to learn novel data while avoiding catastrophic forgetting of previously learned data [34, 35].

Using ANNs, specifically CNNs and DNNs, has successfully been used to teach smart systems to recognize or detect objects [36], [16], [37], [38], [39], read texts [9], and analyze speeches [40, 41]. However, with the growing complexity of applications to solve and the ever-increasing amount of data to treat, ANNs are often implemented on expensive hardware, and the impact of AI on global warming is becoming non-negligible [42, 43]. Thus, while the need for more powerful AI algorithms is not decreasing, it is necessary to propose novel hardware computing solutions to reduce the impact of AI.

1.2 Hardware architectures for edge computing

The development and generalization of computing hardware started with the introduction of the von-Neuman architecture, together with Moore's law prediction. A classical central processing unit (CPU), based on von-Neuman architecture, disconnects the memory unit from the computing unit, inducing memory reading and writing processes during computation, see Figure 1.2. CPUs are nowadays spread everywhere thanks to their efficiency in treating general-purpose sequential tasks. However, for large-scale memory-intensive computations, the com-

munication bus between memory and computing units is saturated, generating computing latency, and high energy consumption, this is called the von Neuman bottleneck [44]. Graphics processing units (GPU) were developed to enable parallel processing, combining multiple CPU cores, to speed up data-intensive processing, such as graphical processing. In ANNs, the core inference operation is the multiply and accumulate (MAC) operation that is computed thousands or millions of times, distributed sequentially or in parallel depending on the architecture. Moreover, ANN learning is in general more computation-intensive than inference, as it uses large datasets to correctly represent all possible input data and maximize the accuracy during inference.

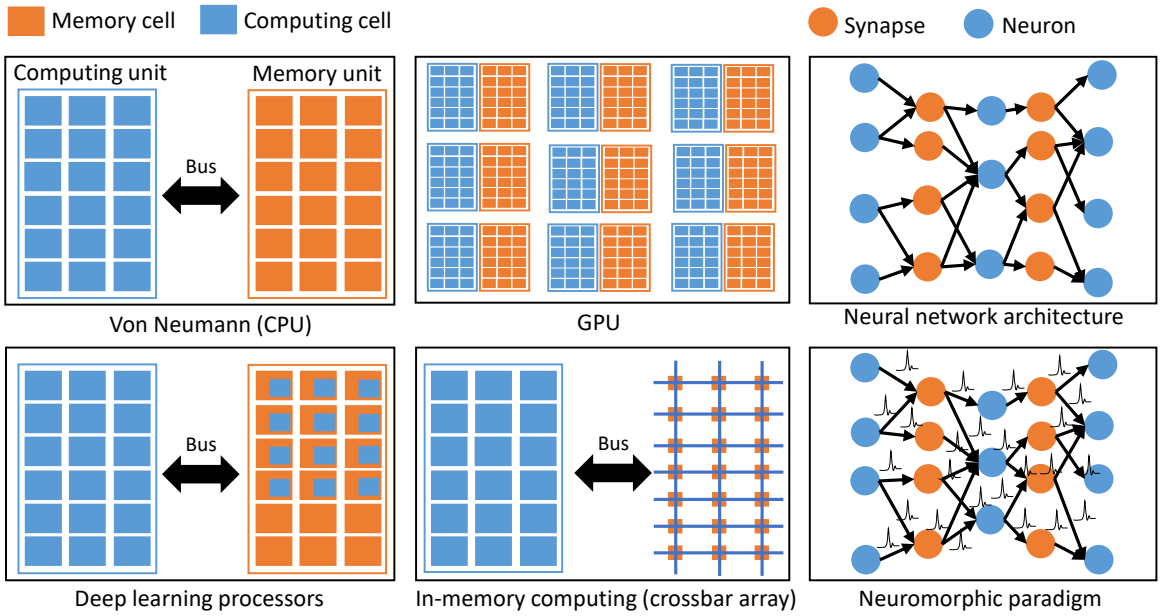


FIGURE 1.2 – *Hardware computing architectures, from von Neuman to in-memory computing, up to neuromorphic computing.*

State-of-the-art ANN computes inference and learning in the cloud using large-scale CPUs and GPUs. With the increasing complexity of ANN models and learning datasets, the important data transfer between processing and memory units slows down ANN computation and consumes more energy [45]. For example, training a CNN model sometimes needs to use multiple GPUs for hours or days to obtain satisfactory precision [46]. Various strategies are explored to improve the performances of cloud data center hardware resources [47]. In particular, researchers study ANN-based hardware accelerators to optimize data and computation distribution to reduce resource utilization, data transfer, latency, and energy consumption of ANN inference and learning in the cloud [48, 49, 50]. Most ANN-based accelerators investigate algorithm-aware hardware processing unit architectures that are made of operation-specific cells with distributed memory for ANN models [51]. For example, deep learning processor (DLP) such as neural processing unit (NPU) and tensor processing unit (TPU) are made of multiple small-scale cells containing both memory units and processing units capable of simple or vectored MAC operations to improve parallelization and accelerate the ANN inferences. Recently, analog computing also gained interest in taking advantage of the physical computing properties of analog circuits for low-cost computation. Analog computing also provides natural continuous values encoding while digital processing still relies on discrete binary values, even though multi-valued logic is also explored [52]. Similarly to analog computing, in-memory computing uses physical attributes of memory devices to compute MAC operations. It allows fast computation with low energy consumption compared to classical von-Neuman architectures [53]. One interesting solution creates crossbar arrays of memory devices to per-

form vectored MAC operations [54, 55], see Figure 1.2. Nowadays, there are digital, analog, or hybrid DLP accelerators. However, even if DLP drastically reduces the energy and latency of DNN inference and learning, it can be challenging to implement it in highly constrained edge devices [51].

Especially, in recent years, we have witnessed a proliferation of smart edge devices adopted by all industry sectors. From healthcare to security, robotics, and even home automation, there are edge computations in every domain [56, 57]. Currently, many edge devices still use cloud computing, sending data over the cloud for computation. However, data transfer requires large bandwidth meanwhile generating important computation latency, energy consumption, and privacy concerns. Thus, there is a need to develop fast and low-power hardware architectures to compute ANN inference and learning at the edge.

Learning at the edge is really challenging due to the large datasets and intensive learning algorithms necessary to achieve high inference accuracy. Yet, one solution comes with continual learning, performing learning iterations through time depending on the environment. Continual learning can hardly be implemented in the cloud due to the recurrent transmission of information between cloud and edge devices. However, implementing learning at the edge is also challenging as it requires additional resources for learning and re-programmable synaptic elements. Thus, at the same time, some study novel low-cost continual learning algorithms [58] and others explore compact and low-power devices and circuits for reconfigurable synapses [59].

For edge inference, a straightforward solution is to consider existing tiny micro-controllers with tiny ANN models [60], applying additional constraints to their architectures. For example, limiting the synaptic parameter precision, scaling down the network size, increasing the sparsity of synaptic connections, or reducing the possible states of the activation functions can optimize ANN models to reduce resource utilization and energy consumption while saving correct precision [61]. Recently, binary neural network (BNN) were investigated as low-cost solutions for image classification tasks at the edge [62, 63, 64, 65]. Alternatively, in the same spirit as DLP for cloud computing, brain-inspired neural network architectures were introduced, distributing memory to reduce latency and energy consumption induced by on-chip data transfer. Neural network architectures use analog or digital circuits to mimic ANN neuron and synapse functionalities, see Figure 1.2. Neurons and synapses are often implemented with multiple analog or digital circuits depending on the type of ANN and the task to solve. For example, in-memory computing with crossbar arrays can also be configured in neural network architectures if combined with additional neural activation circuits [66, 67]. Implementing tiny ANN models in neural network architectures can importantly reduce ANN inference latency and energy consumption. However, using tiny ANN models, with either micro-controllers or neural network architectures, can also impact the accuracy performances of the model.

In the human brain, data is encoded in the time domain through spikes. Using the time domain can encode continuous data in a more natural manner than in the amplitude domain. Also, edge devices often process dynamic data from sensors, and using the time to encode data creates natural dependencies between the following data, helping for dynamic data processing. Temporal encoding, together with the biological plausibility motivated the recent development of brain-inspired neuromorphic computing paradigms [68, 69, 70], mimicking not only the biological neural network architecture but also the representation of information.

1.3 Neuromorphic computing paradigms

The definition of a neuromorphic computing paradigm is still debated today [69] but in this thesis, we define a neuromorphic computing paradigm as an electronic system mimicking the brain with its neural network architecture and representation of information. Taking inspiration from biological representations of information, various neuromorphic paradigms were proposed to achieve low-power computation. On one side, researchers explore novel materials, devices, and circuits to build low-power neuromorphic systems capable of learning and inference [71, 72, 69]. In parallel, novel algorithms and applications are explored to demonstrate the effectiveness and added value of neuromorphic paradigms for edge computing [53, 73].

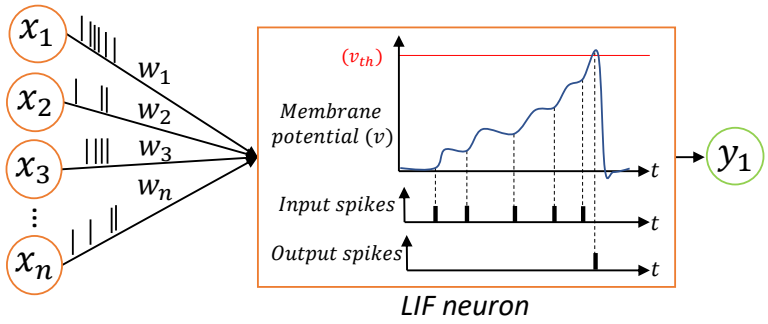


FIGURE 1.3 – *Leaky integrate and fire (LIF) spiking neuron model.*

The state-of-the-art neuromorphic paradigm is called spiking neural network (SNN) [74]. SNNs were introduced in the 1990s taking inspiration from the human brain where information is transmitted between neurons through spikes [75, 74, 76]. The interest of SNN is its data representation with spikes in the time domain, providing natural and low-energy continuous data encoding [77]. Many different spiking neuron models were developed over time, being more biologically plausible or providing better accuracy [75, 76, 78]. Nowadays, the leaky integrate and fire (LIF) neuron model [79] is widely used for its simplicity and efficiency while being less biologically plausible than others. The LIF neuron integrates the post-synaptic spikes by increasing the membrane potential of the neuron until it reaches a threshold that triggers an output spike, see Figure 1.3. Using numerical data, SNN necessitates data encoding through time, using mainly rate or temporal encoding, to be used straightforwardly on event-based data and sensors [80]. For example, SNN has been efficiently used to classify images from dynamic vision sensors (DVS) or event-based cameras [81, 82, 83]. To do so, various supervised and unsupervised learning strategies have been proposed [73, 84]. The supervised back-propagation algorithm can not directly be employed to train SNN because of the non-differentiable activation function [85]. A simple solution is to employ back-propagation on an ANN with an approximated differentiable activation function before transferring the weights to a real SNN. However, back-propagation is not biologically plausible, and the precision obtained is not always competitive with equivalent ANNs [86]. Thus, alternative spike-based supervised learning algorithms are also investigated to obtain better precision and to be more biologically plausible [87, 88, 89, 90, 91]. In parallel, always with the aim of being closer to the brain, spiking synaptic plasticity was introduced with the spike timing dependent plasticity (STDP) [92, 93], close to the Hebbian rule [28]. STDP has efficiently been used with many SNN architectures, feed-forward or recurrent, to perform image classification, clustering, or dynamic data treatment [94, 95]. Furthermore, beyond-ML applications have been explored, solving for example combinatorial optimization problem (COP) [96] or random walks [97]. Even if SNN showcased good accuracy for edge applications, it still falls behind ANN performances. Current research interests concentrate first on novel beyond-ML algorithms and applications for which SNN

overcomes ANN accuracy. Meanwhile the exploration of SNN learning algorithms and applications, other efforts are given to create efficient SNN testing methods [98, 99] and to build reliable SNN-based chips [100]. Multiple neuromorphic chips were developed providing large-scale SNNs for efficient edge inference, for example SpiNNaker [101], Truenorth [102], Loihi [103], Brainscale [104], among others. However, those neuromorphic chips do not provide learning at the edge, and continual online or on-chip learning solutions for SNNs are still being explored. Complementary to these large-scale CMOS-based neuromorphic chips, important efforts are being made to create novel materials, devices, and circuits for low-power compact SNN hardware [105, 69].

SNN has been widely explored in the last decades. However, novel promising neuromorphic computing paradigms emerged recently to substitute SNN for edge computing, i.e. hyperdimensional computing takes inspiration from the hyperdimensionality, the holographic representation, and the randomness of the biological neural network to compute [106, 107]. Alternatively, another neuromorphic paradigm emerged recently, taking inspiration from the collective computational properties of the brain oscillations with oscillatory neural network (ONN) [108].

1.4 Oscillatory Neural Networks (ONNs)

ONN is a promising alternative neuromorphic computing paradigm taking inspiration from the collective computational properties of brain areas through oscillations [109]. ONN computes using the physical synchronization of coupled oscillators [110, 111, 112]. To make the parallel with ANNs and SNNs, each neuron is an oscillator and each synapse is the coupling element, or the connectivity, between oscillators. In particular, ONN can be considered a specific case of SNN where neurons have constant frequency [113]. Synchronization of coupled oscillators was first observed by Huygens with Pendulum [114], before being employed for solving boolean functions in the 1950s [115, 116]. However, the development of transistor-based logic became more competitive than coupled oscillators. In the 1980s, Hopfield [20, 117] combined the neural network principle with physics and the Ising principle [118] to create energy-based spin-glass models. Later, Hoppenstead [119] and Aoyagi [120] highlighted the intrinsic Ising-energy minimization of ONN, which brought ONN back into the spotlight combined with its remarkable dynamic from Kuramoto [121]. It motivated on one side, the research for compact, fast, and low-power ONN implementations, and on the other side novel ONN architectures, learning algorithms, and edge applications.

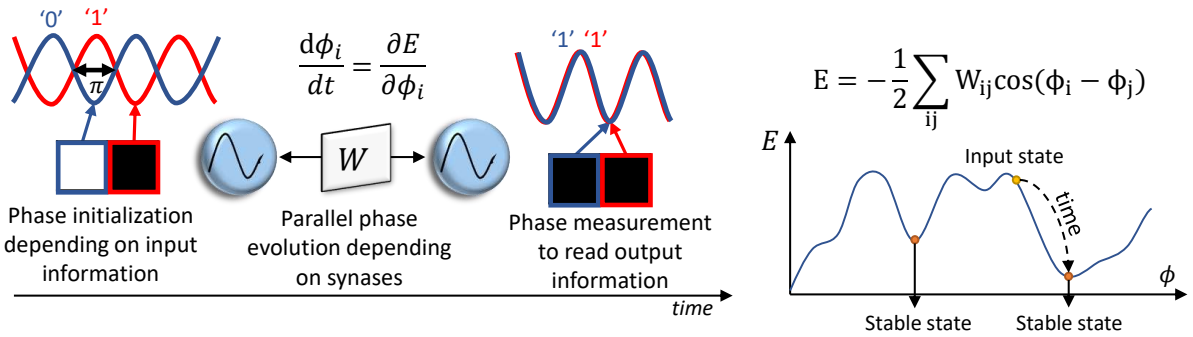


FIGURE 1.4 – *Phase-based ONN computing principle.*

The non-linear dynamics of coupled oscillators can be harnessed in many different ways to perform intelligent tasks. However, most ONN developments fall into two classes depen-

ding on the type of input/output encoding, **frequency-based ONN** where inputs are oscillator frequencies and outputs are the synchronization levels between oscillators, and **phase-based ONN** where oscillators have equal frequency and input/output is encoded in phase relationship among oscillators. Additionally, in phase-computing ONN, the phase evolution during computation can be associated with the minimization of an energy function, like in HNNs or attractor networks [122], see Figure 1.4. In comparison with classical ANNs, ONNs like SNNs do not use a deterministic activation function but compute depending on physical parameters with phase and/or frequency dynamic models, such as the Kuramoto model [121]. Also, SNNs represent information in binary spikes while ONNs represent information in continuous phase or frequency. Figure 1.5 presents the main differences of ANN, SNN, and ONN computing.

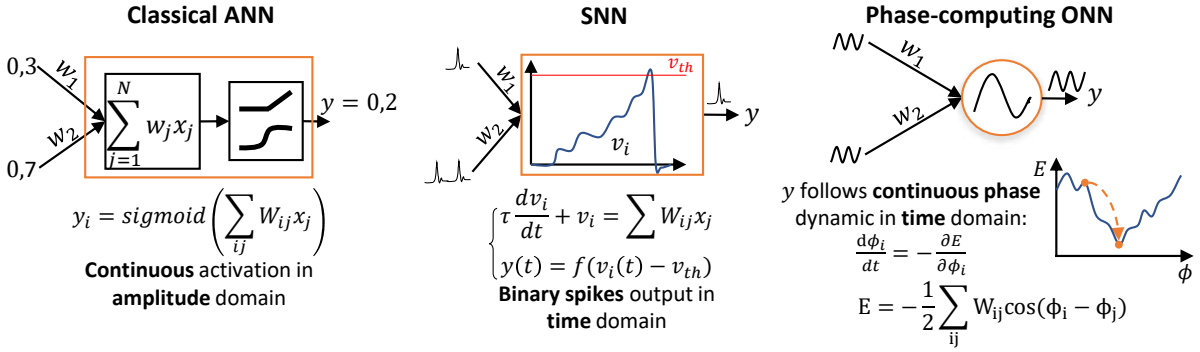


FIGURE 1.5 – *ANN vs. SNN vs. ONN computing.*

From the theory of computing with coupled oscillators, researchers investigated how to build ONN demonstrators for meaningful applications, see Figure 1.6. Starting with Wang and Terman in 1994, who introduced the locally excitatory globally inhibitory oscillator network (LEGION) ONN array for image segmentation [123], just before Hoppensteadt and Izhikevich linked the theory of coupled oscillators to energy-based HNNs [20] for associative memory applications [124, 119]. LEGION was introduced in 1994 as a solution to perform image segmentation using an array topology of locally excitatory oscillators computing in phase [123, 125, 126, 127] with an additional global inhibitory neuron, all trained with unsupervised Hebbian learning. The first analog implementation was proposed in 1999 [128] and further improved with a neuromorphic analog image segmentation system in 2006 [129]. Meanwhile, other works focused on adapted LEGION architectures with digital implementations [130, 131, 132]. Lately, LEGION motivated the development of phase-based and frequency-based ONNs for clustering and vision tasks [133, 134, 135, 136] using oscillators' array topology. In parallel, Hopfield's work [20, 117] propelled novel energy-based models of neural networks using analog phase dynamics such as phasor neural networks [137] and oscillatory Hopfield network (OHN) [108, 124]. The fully connected architecture allows recurrent signal propagation with oscillating neurons capable of performing auto-associative memory (AAM) tasks or pattern recognition using unsupervised Hebbian learning. Although Hoppensteadt presented the first hardware solution to implement OHN using phase-locked loop (PLL) in 2000 [119], many challenges were still limiting the large-scale implementation of such networks. In 2011, it was reported the first implemented OHN with 8 analog van der Pol oscillators performing phase-based pattern recognition [138]. Later, building OHNs with spin-torque oscillator (STO) was suggested to solve pattern recognition with frequency-based computing [139]. However, OHN raised more interest when Jackson[140, 141] and Shi [142] proposed larger-scale OHN, going up to 100 oscillators. Similarly, the development of novel compact and low-power devices for neuromorphic computing offered novel solutions for efficient phase-computing ONN [143, 144, 145].

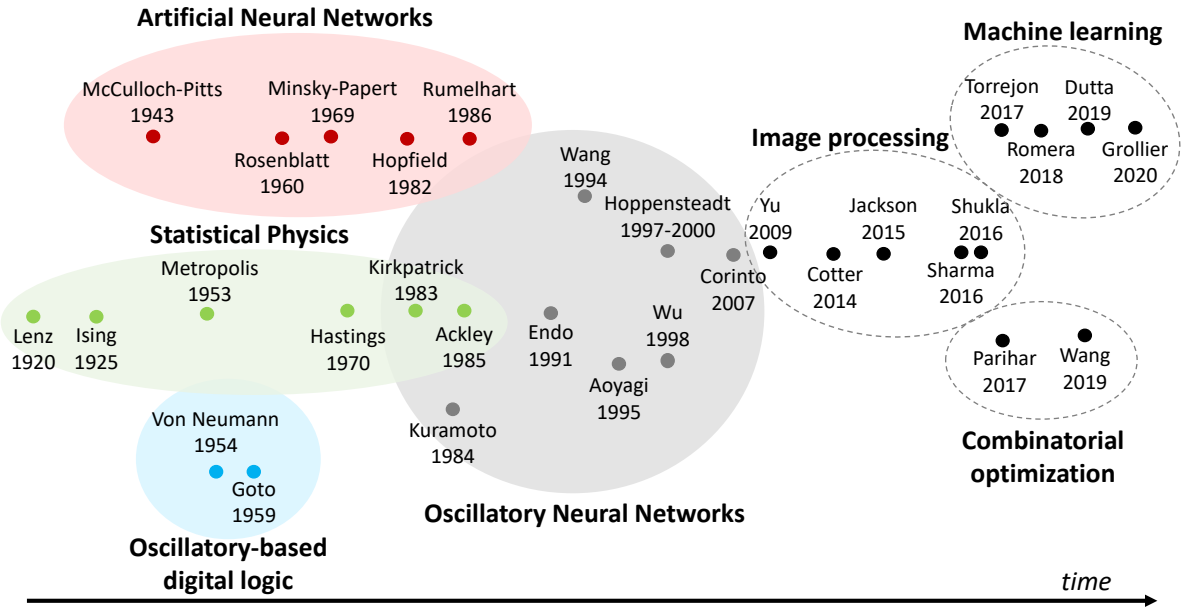


FIGURE 1.6 – *Timeline of ONN historical development.*

In the last decades, researchers also proposed alternative architectures to LEGION and OHN. For example, the star coupling topology was introduced to perform phase-based static or dynamic pattern recognition [146, 147]. The star coupling was then derived in frequency-based computing to also perform image processing, like pattern recognition [148], image segmentation [149] and convolution operations [150]. Alternatively, convolution operations can also be solved using layered networks of frequency-computing oscillatory neurons [151]. The layered topology, often used in neural networks, was also applied for classification tasks using oscillatory neurons, however being different from ONN computing [152, 153, 154, 72]. Additionally, for classification, the random and sparse topology of reservoir computing combined with the high non-linearity of coupled oscillators highlighted low-power and low-density properties [155, 156, 157]. Finally, another promising area recently proposed to build oscillatory Ising machine (OIM) to solve COPs creating graph-type architectures [158, 159, 160].

In terms of learning, Hebbian plasticity is the state-of-the-art for many applications and architectures, for example performing image segmentation [123, 125, 149], or pattern recognition [138, 141, 147]. Alternatively, we found in the literature a few works using simulated annealing to define circuit parameters [151], as well as gradient back-propagation on ANNs with weight transfer [152]. Finally, others perform custom learning, reproducing convolution filters [150, 143], or adapting a computational graph representation with coupled oscillators for COP. Up to our knowledge, all existing ONN demonstrators perform learning offline and synaptic weights are configured before inference.

1.5 Challenges and motivations

The theory around coupled oscillators encourages research on ONN-based systems for low-power edge AI accelerators. In particular, in this thesis, we focus on phase-computing ONN accelerators, motivated by:

1. the **physical dynamic** that can facilitate the ONN design with a wide variety of oscillators, including low-power compact analog or digital devices,

2. the **parallel synchronization** of coupled oscillators providing fast inference,

3. the **data representation** in the time domain with oscillators' phases limiting voltage amplitude and power consumption.

In the literature, different architectures, implementations, learning algorithms, and applications were introduced for ONN, however, there appear to be a few isolated works. More generally, the state-of-the-art for ONN is to build a network of fully-coupled oscillators, configured with unsupervised Hebbian learning to solve AAM tasks, creating an OHN.

The fully-coupled architecture limits the scalability of ONN implementation. Considering a network of N neurons, there are $N(N-1)$ synapses, making it difficult to implement at a large scale. For example, the largest OHN implemented in hardware integrates 100 neurons and is designed mainly with digital technologies [141]. Meanwhile, for analog design, a recent work built the largest 30-neuron fully connected ONN [161]. In comparison, [162] proposed an ONN-based digital hardware integrating 1968 oscillators interconnected in an array topology to perform COP. Even if a large part of the ONN community explores novel low-power compact materials, devices, and circuits [163, 164, 72, 159], we believe there is a need to investigate alternative ONN architectures to go beyond OHN.

Furthermore, OHN, such as HNN, trained with unsupervised Hebbian learning is not competitive with alternative models executing AAM tasks [165]. Thus, it is also necessary to study innovative learning solutions, first to try increasing OHN performances on AAM tasks, then to provide on-chip or online learning ability to OHN designs, and finally to enlarge the ONN scope of architectures and applications. The ONN can not be shortened to OHN to answer edge AI requirements, and we believe there are alternative applications where ONN can be competitive against other ANN and neuromorphic models.

Finally, to study and demonstrate diverse ONN architectures, learning algorithms, and applications, we need an easily reconfigurable ONN implementation. Current state-of-the-art ONN implementations are hardly reconfigurable, for example, synaptic weights can be modified but the core structure of the network can not be changed. Here, we focus on a digital ONN implemented into a field-programmable gate array (FPGA) to be able to reconfigure the ONN synaptic weights and structure and to allow deploying demonstrators easily inside edge systems. Thus, in this Ph.D. thesis, I explore novel architectures, learning algorithms, and edge applications to go beyond OHN based on the digital ONN on FPGA.

1.6 Outline

My Ph.D. thesis is organized into four chapters. **Chapter 2** details the phase-based ONN computing paradigm, presents the digital ONN implementation, and characterizes the design compared with previous ONN implementations, considering the state-of-the-art fully connected recurrent OHN configured for pattern recognition.

Chapter 3 studies various learning algorithms to improve OHN performances for pattern recognition. In particular, it first studies the adaptation of unsupervised learning algorithms introduced for HNN to the digital OHN with binary patterns, considering offline and on-chip

learning. Then, it introduces a supervised learning algorithm and showcases the performance improvement to solve a digit recognition application.

Chapter 4 introduces a novel ONN architecture with cascaded OHNs implemented in digital which allows applying ONN to a robotic obstacle avoidance application. Furthermore, it proposes a solution to implement the cascaded OHN architecture in analog to perform an image edge detection application.

Chapter 5 presents another innovative ONN architecture with multi-layer ONNs for hetero-association or classification. It studies inference and learning with multi-layer ONNs implemented digitally considering bidirectional or feed-forward connections. Also, it showcases the application of two-layer ONNs to image edge detection and feature extraction.

Finally,**Chapter 6** provides a discussion along with a general conclusion to resume the main contributions of this Ph.D. thesis and open novel perspectives for the ONN computing paradigm. Table 1.1 and Figure 1.7 summarize the main contributions of each chapter.

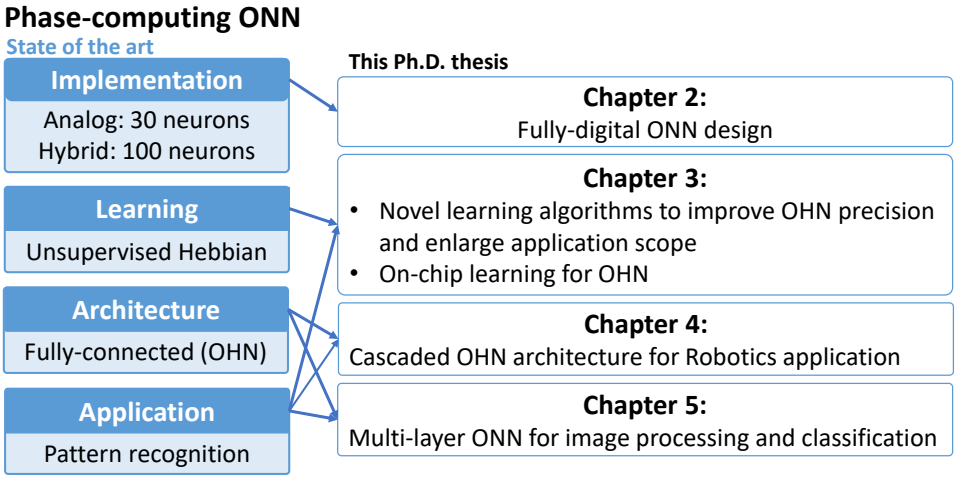


FIGURE 1.7 – *Main contributions of the Ph.D. thesis. OHN means oscillatory Hopfield network.*

TABLE 1.1 – *Main contributions of the Ph.D. thesis. OHN means Oscillatory Hopfield Network and EP means equilibrium propagation.*

Ch.	Problem	Topic	Contributions
1	Can we build a large-scale reconfigurable ONN?	Digital OHN for pattern recognition.	(1) I detail the ONN computation and the digital OHN on FPGA. (2) I validate the OHN with unsupervised learning solving digit recognition. (3) I demonstrate its real-time performances at the edge using a camera stream.
2	Can we improve OHN precision results?	OHN learning.	(1) I study and compare unsupervised learning algorithms for OHN. (2) I build an on-chip learning platform for OHN. (3) I apply supervised EP learning to OHN solving MNIST.
3	Can we go beyond-OHN with novel architectures?	Cascaded OHN architecture.	(1) I introduce a cascaded OHN architecture and implement it in FPGA. (2) I apply it on robotic obstacle avoidance using off- and on-chip learning. (3) I apply an analog cascaded OHNs to image edge detection.
4	Can we go beyond-OHN with novel architectures?	Layered ONN architecture.	(1) I introduce layered ONN architectures and implement it in digital. (2) I apply a 2-layer ONN to image edge detection. (3) I apply a 3-layer ONN to classification.

CHAPTER 2

DIGITAL OSCILLATORY HOPFIELD NETWORK (OHN)

Contents

2.1 Introduction

An oscillatory neural network (ONN) is a physical and hardware-based, neuromorphic computing approach [166, 167] which aims to provide low-power AI edge systems. Phase-computing ONNs, that are considered in this work, use coupled oscillators mimicking at the circuit level the basic structure of the brain architecture, and at the system level the collective computational synchronization of brain areas. In phase-based ONN, information is represented in the phase relationship between oscillators, and coupling between them induces phase synchronization or de-synchronization in time. For example, considering mechanical oscillators with metronomes, the random start of five homogeneous metronomes in a propagating environment will make them oscillate in parallel [168]. After several cycles, they get synchronized in frequency while their phase relations can give us meaningful information. Using phase-based computing, ONN achieves fast parallel computation with a limited oscillating voltage amplitude to perform low-power computation.

In state-of-the-art, ONN is used as an oscillatory Hopfield network (OHN) structured with a fully connected architecture, configured with unsupervised Hebbian learning for pattern recognition [119], such as in HNN. However, current implementations of OHN are limited in size due to the exponential increase of synaptic elements when increasing the number of oscillating neurons. To our best knowledge, the largest OHN fully analog design reaches 30 oscillators [161], and the largest hybrid OHN reaches 100 neurons [141]. Compared to other ANNs, built with thousands of neurons, HNN and OHN are also limited in terms of precision. The recent development of modern Hopfield networks with multi-layer architectures and continuous output states trained with gradient-based learning achieved competitive results com-

pared with other machine learning methods [169]. It motivates the exploration of novel ONN architectures and learning algorithms to go beyond OHN and showcase ONN effectiveness for edge applications compared with other ANNs.

To explore and demonstrate beyond-OHN computation with coupled oscillators, we consider a proof-of-concept of the ONN computing paradigm with a digital design implemented on field-programmable gate array (FPGA) [170]. The ONN on FPGA allows fast and easy reconfigurability of the ONN in terms of size, architecture, and applications. Also, the ONN on FPGA can easily be integrated into larger systems to demonstrate ONN for edge applications.

In this Chapter, we present the ONN computing paradigm before introducing the OHN computing principle and training process taking inspiration from HNN. Then, we describe the digital OHN design and validate it through an auto-associative memory (AAM) application. Finally, we characterize the digital OHN and compare it with previous OHN implementations to discuss its advantages and limitations.

2.2 ONN computing paradigm

ONN computing paradigm operates using the physical synchronization between coupled oscillators. The dynamic of coupled oscillators can be represented by the Kuramoto model which expresses the time derivative of oscillators' phase or frequency like:

$$\frac{d\phi_i}{dt} = \omega_i + \sum_j K_{ij} \sin\left(\phi_j - \phi_i\right) \tag{2.1}$$

Where ω_i is the oscillator free-running frequency. The sinusoidal interaction terms are responsible for the frequency adjustment and they model the adaptation of oscillating neuron i to other oscillating neurons j. Despite its simple expression, the Kuramoto model produces very complex dynamics depending on the connectivity (K_{ij}) and frequency distributions [171].

The non-linear dynamics of coupled oscillators such as (1) can be harnessed in many different ways to perform intelligent tasks. However, most ONN developments fall into two classes depending on the type of input/output encoding: **Frequency-based ONN** and **Phase-based ONN**.

In a **Frequency-based ONN**, frequency-dependent input signals are injected into the ONN that reacts to the input perturbations. During computation, groups of oscillators lock in frequency, representing synchronization. Interestingly, computing in the frequency domain can also remove some physical connections between oscillators [172]. Despite the advantageous physical scaling of the proposed ONN, generating the modulation signal which includes all pairwise oscillator interactions is not straightforward [138, 173]. This computation scheme has been used for image processing [136, 149, 150], associative memory tasks [148], or spoken vowel classification [153, 174].

In this thesis, we focus on **Phase-based ONN**, in which oscillators have equal frequencies, the coupling elements are symmetric ($K_{ij} = K_{ji}$), and the inputs and outputs are encoded in the phase relationship between oscillators. With that, Hoppensteadt and Izhikevich [119] have shown that the ONN inference computation minimizes an energy function through time, such as in HNNs, Ising models [118], and attractor networks [122]. The ONN energy follows:

$$E = -\frac{1}{2} \sum_i \sum_j K_{ij} \cos\left(\phi_i - \phi_j\right) \tag{2.2}$$

Oscillator dynamics depend on the phase initialization and the ONN energy, enabling a broad range of applications. For example, considering a fully connected recurrent architecture, ONN becomes an Oscillatory Hopfield Network (OHN) acting as an HNN to perform pattern recognition.

2.3 Oscillatory Hopfield network (OHN)

OHN was introduced in 1997 by Hoppensteadt [124], and corresponds to an ONN configured as an HNN, with a fully-connected architecture, to perform pattern recognition. Pattern recognition is defined as the ability to learn and store patterns in a system, and to retrieve one of the patterns from corrupted input information. For example, considering images as patterns, a system configured for pattern recognition can memorize images and retrieve them from corrupted images with noisy or missing pixels. Classical HNN are state-of-the-art neural networks for solving pattern recognition [20].

HNN is a particular case of RNNs considering a single layer of neurons, where each neuron is connected with the others apart from itself. Each neuron is a perceptron neuron with a weighted sum as an integration function and a $sign$ activation function, constraining the original HNN to binary bipolar states $\{-1, +1\}$, see Figure 2.1. In the case of the AAM task with images, each neuron represents a pixel, and the neuron activation value $\{-1\}$ or $\{1\}$ represents the pixel color. Thus, classical HNN can treat and learn binary patterns, like images with black and white pixels. Recently, alternative HNNs were proposed to treat and learn multi-state or continuous patterns, such as the complex HNN using complex activation functions and complex weights [175, 176], or the modern HNN considering multi-layer architectures with continuous activation functions [177, 169].

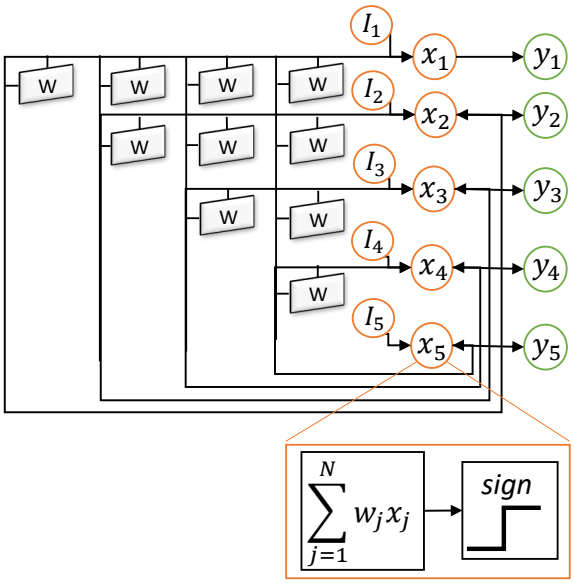
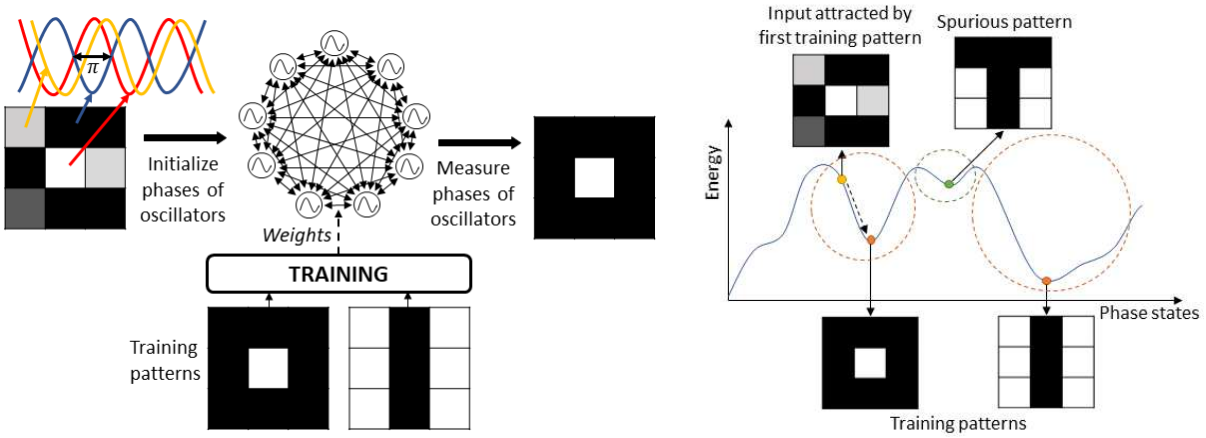


FIGURE 2.1 – *HNN building blocks.*

For ONNs, each neuron activation can take various phase values depending on the ONN design permitting multi-state or continuous information, like gray-scale images. For pattern recognition, the couplings among neurons are configured during learning and represent the memory of the network. In particular, during the learning process, the training algorithm defines the coupling weight values such that learning patterns become minima on the ONN energy

landscape, and ONN acts as a clustering problem by creating clusters of images attracted by training patterns, see Figure 2.2b. Learning does not ensure that all local minima are training patterns, and in some cases, local minima become stable phase states while it does not correspond to any learning pattern, which is labeled as a spurious pattern, see Figure 2.2b. During the ONN inference process, one input pattern is applied to the network by initializing the oscillators' phases with the corresponding input information. Then, phases evolve thanks to the inherent phase interaction between coupled oscillators until they stabilize and the final phase state represents the OHN output pattern, see Figure 2.2a. In practice, the measurement of the OHN output state is done either regularly to detect stability, or at a predefined time after which it is considered stable.



(a) *ONN configured as OHN and trained for pattern recognition.*

(b) *Simplified representation of an energy landscape in the case of pattern recognition task.*

FIGURE 2.2 – *OHN for pattern recognition, with energy landscape.*

Existing learning algorithms to train an OHN for pattern recognition are mainly unsupervised learning rules which were first introduced for HNNs [178]. The Hebbian learning rule [28] is one of the most popular learning algorithms to calculate synaptic weights for bipolar-valued stored patterns on HNNs. So, to validate the digital OHN implementation, we first apply the Hebbian learning rule, considering bipolar learning pattern values. We transform each stored pattern with index k into a vector ξ^k of length N, with N, the number of neurons inside the network. Each vector element is bipolar $\{-1; +1\}$. The synaptic weight w_{ij} between neuron i and neuron j is calculated as:

$$w_{ij} = \frac{1}{N} \sum_{k} \xi_i^k \xi_j^{k^T} \tag{2.3}$$

with $w_{ij} = 0 \, \forall \, i = j$. Note, for ONN inference, we translate the bipolar values used for training to opposite phase values, such that a $\{-1\}$ is encoded to a $\{0^o\}$ phase while a $\{+1\}$ is encoded to a $\{180^o\}$ phase.

2.4 Digital OHN implementation

We develop a digital ONN to be implemented on FPGA as a proof-of-concept of the phase computing paradigm to explore ONN architectures, learning algorithms, and AI edge applications. The digital ONN design is first proposed and validated considering the state-of-the-art OHN architecture [170] trained with the state-of-the-art Hebbian learning rule.

We present an OHN digital design inspired by hybrid analog-digital work from [141] but without analog components. In [141], synapses are implemented by a resistor network, and a critical analog comparator is required at the input of each digital neuron. In contrast, in the digital ONN implementation, an arithmetic circuit is used for each synapse and there is no analog comparator in neurons. Figure 2.3 and 2.4 illustrate the digital OHN architecture for two and four neurons. Note, compared to analog OHN architectures, in the digital OHN we implement both synaptic weights from neuron i to neuron j, and from neuron j to neuron i separately, so we can differentiate both weights. Additionally, we can also implement self-coupling, with weights connected from neuron i to neuron i.

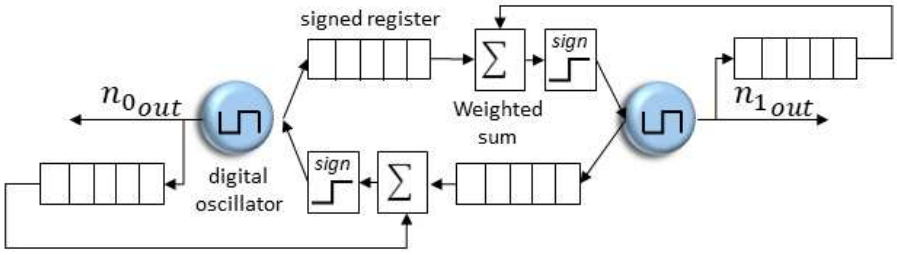


FIGURE 2.3 – *Digital 2-neuron OHN with separate weights between neuron i and neuron j and additional self-coupling.*

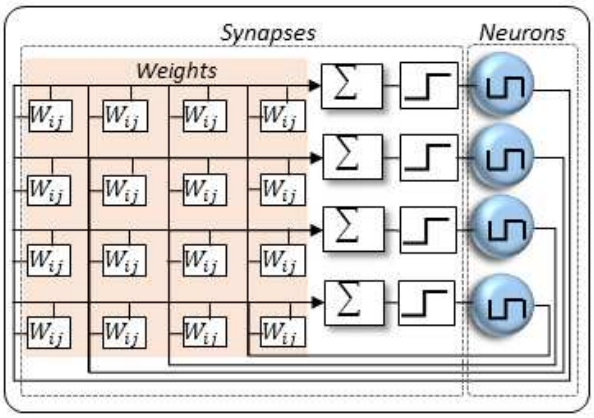


FIGURE 2.4 – *Digital 4-neuron OHN with separate weights between neuron i and neuron j and additional self-coupling.*

We implement an OHN, where each neuron inside the FPGA oscillates in parallel. In addition, the synapse block is combinatorial, which means all synaptic operations are computed in parallel. Finally, the architecture needs extra blocks to control synapses and neuron signals. Figure 2.5 presents the digital OHN design composed of neurons, synapses, and control blocks. In the following sections, we detail the implementation of each block.

2.4.1 Digital oscillators

In this design, neurons are phase-changed oscillators made of a state register, a phase calculator, and a phase-controlled oscillator, see Figure 2.6. Each neuron i computes the phase difference between the present input and output oscillations to align the output in phase with the input. A neuron has one 1-bit input, n_{in}, and one 1-bit output, n_{out}, that are oscillating square signals, in addition to synchronization, initialization, and control signals. The synapse block generates the n_{in} signal, which determines the evolution of the neuron phase. The ϕ_{out} and $state\_changed$ signals give information on the neuron phase and its evolution. We use

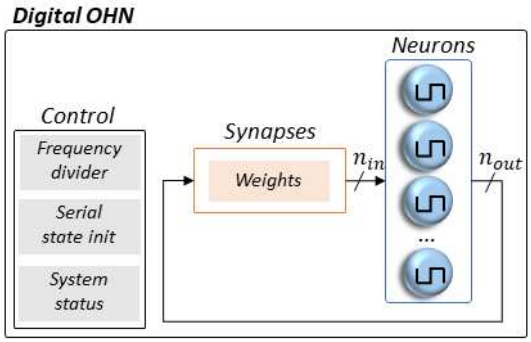


FIGURE 2.5 – *Simplified representation of the digital OHN building blocks.*

$full\_tick$, ser_state_in, and ser_state_out signals to initialize the neuron phase, and $reset$, clk, and $slowclock$ signals to ensure synchronization.

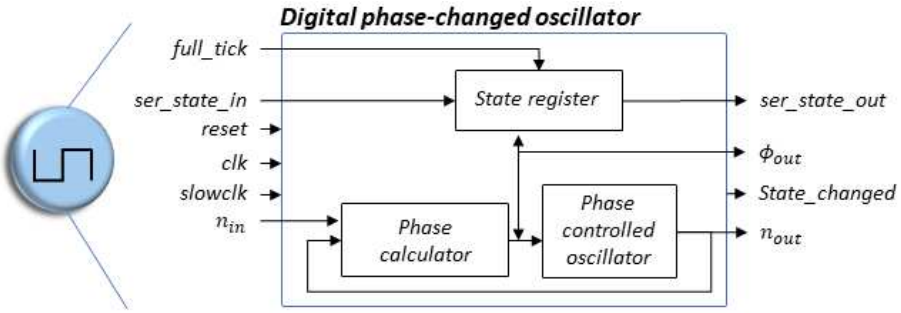


FIGURE 2.6 – *Operating principle of a single phase-changed oscillatory neuron.*

The process starts with the initialization of the output signal phase ϕ_{out}. It triggers the initial output oscillation n_{out} aligned to ϕ_{out}. Automatically, the synapse block computes the new input oscillation n_{in} with the phase ϕ_{in}. We initialize all neurons serially when $full\_tick$ is activated by connecting ser_state_out signal to ser_state_in signal of the neighbor neuron. Note, when initialization is over, a scan path between ser_state_out and ser_state_in is configured to load and read OHN's state in series.

Then, each neuron calculates the phase difference $\Delta\phi$ between ϕ_{in} and ϕ_{out} and uses it to update the new ϕ_{out} with a phase calculator block. The phase calculator block contains two edge detectors and a finite state machine (FSM). Edge detectors detect rising edges on n_{in} and n_{out} oscillating signals. FSM measures the time difference between n_{in}'s rising edge and n_{out}'s rising edge to define $\Delta\phi$. The $\Delta\phi$ value allows us to update the neuron output phase ϕ_{out} aligning n_{out} signal with n_{in} signal, as:

$$\phi_{out} = \phi_{out} + / - \Delta\phi \tag{2.4}$$

Note, the sign $(+/-)$ depends on the first rising edge detected. (-) if n_{out}'s rising edge is detected first and (+) if n_{in}'s rising edge is detected first. Also note, that the n_{in} signal phase is set by the weighted sum of the neuron's input signals, see 2.5.

Finally, the new ϕ_{out} is applied to the oscillating output signal n_{out} with a phase-controlled oscillator. The phase-controlled oscillator contains a circular shift register with a multiplexer. The shift register has 16 stages to represent square signals with different phases. We chose 16 phase stages to enable multi-state ONN computation meanwhile keeping a reasonable frequency. The 16-bit pattern $[1111111100000000]$ cycles continuously through time. So, through

the multiplexer selection bits, we select a shift-register state corresponding to a square signal with a distinct phase. This square signal becomes the new neuron output n_{out}. Figure 2.7 shows the logic diagram of the phase-controlled oscillator and Figure 2.8 the waveforms corresponding to stage 0 (in-phase, ϕ_{out}= 0°) and stage 2 (out-of-phase, ϕ_{out}= 45°). The register controlling the multiplexer stores the neuron state, or equivalently the phase of the neuron output. Note, that we use different clocks (driven by the system clock) to control the state register and the shift register. The latter is driven by a slow clock generated from the system clock so that the multiplexer's output cycles as long as its control register remains unchanged with a period $T_{osc} = 16 * T_{slowclk}$.

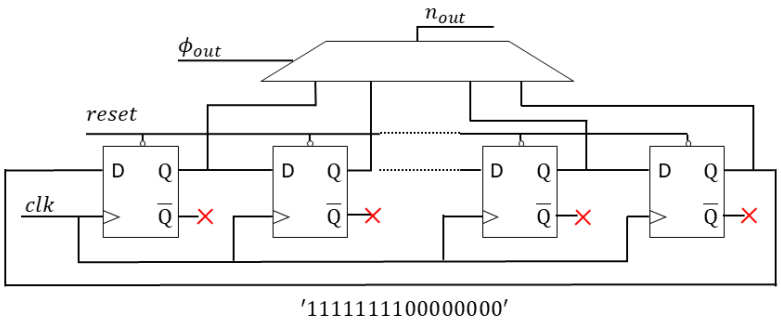


FIGURE 2.7 – *Logic diagram of the implemented phase-controlled oscillator.*

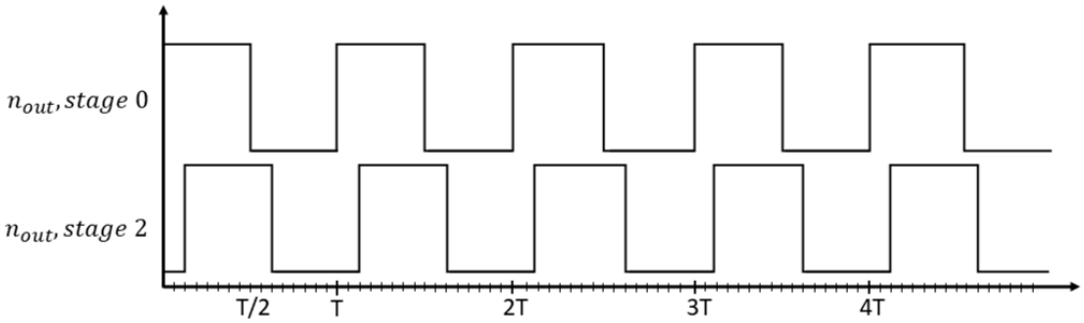


FIGURE 2.8 – *Output waveform of internal neuron shift-register for stages 0 and 2.*

2.4.2 Hard-coded 5-bit register synapses

The digital ONN synapses block contains weights and computes each neuron input oscillation n_{in} using an arithmetic logic circuit to generate the input signal to the i-th neuron as:

$$n_{in}[i] = sign(\sum_{j} w_{ij} - \sum_{k} w_{ik}) \tag{2.5}$$

where j extends to those neurons with $n_{out}[j] = 1$ and k to those with $n_{out}[k] = 0$.

Hard-coded synaptic weights are encoded using fixed-precision registers. At first glance, the precision of each synapse is set to 5-bit as HNN highlighted similar performances with full precision weights and with 5-bit precision weights. Note, that the synaptic block is the block that requires the most logical resources due to the massive parallel computation. Thus later, a new OHN design was proposed with a partially combinatorial synaptic block with some sequential computation which did not impact the ONN latency. In the next sections, results are presented with the partially combinatorial synaptic block.

2.4.3 Additional control block

In addition to neurons and synapses, the digital design requires a control block to control and monitor the OHN computation. It is mainly in charge of three tasks.

1. The initialization step that is required to carry out an OHN computation. The input state is serially applied with a scan path on the neuron's state registers while activating the $full\_tick$ signal.

2. The control block generates a slow clock to ensure OHN operation. The relation between the slow clock and the system clock is $T_{slowclk} = 4 * T_{clk}$ with a frequency divider of 2-bit length to speed up the system performance.

3. The generation of the steady ($steady\_check$) and the inconsistent ($inconsistent\_check$) signals. They indicate whether ONN gives a stable or unstable state. The steady signal is activated once the ONN reaches a stable state, meaning all neuron phases ϕ_{out} are stable for two oscillation periods (T_{osc}). The inconsistent signal is activated in case the ONN does not achieve any stable state after an arbitrary time of 10 oscillating periods.

To do so, the control block monitors neurons' oscillation activity. The combination of neuron blocks, synapse blocks, and control blocks creates the complete fully digital OHN design.

2.5 Validation and characterization of the digital OHN

The digital OHN is evaluated on simple AAM tasks and compared with an HNN Matlab emulator, using first simulation and then implementation to ensure the OHN operation on a real embedded platform. More than that, the digital OHN is demonstrated for real-time image recognition on a camera stream application. After validation, the network specifications and real performances of the system are extracted and compared with previous OHN implementations.

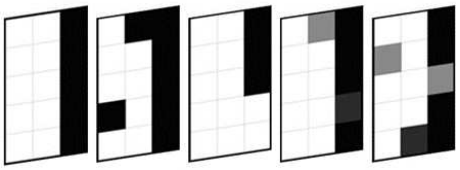
2.5.1 Simulation evaluation of the digital OHN

We first validate and characterize the digital OHN with simulation software tools before being implemented on FPGA. We carry out simulations with a 5x3 OHN and a 10x6 OHN configured for pattern recognition trained with the Hebbian learning rule. The 5x3 OHN is configured with three stored patterns with standard 5x3 bitmap representations of digits 0, 1, and 2, see Figure 2.9a. Each pixel of the image corresponds to a neuron and each pixel color is associated with the neuron phase, with white as in-phase (0^o), and black as out-of-phase (180^o). Grey-level pixels are encoded with intermediate phases. Similarly, the 10x6 OHN is configured with five stored patterns representing digits 0, 1, 2, 3, and 4, see Figure 2.9c. For each OHN, the test set contains both stored patterns and four corrupted patterns created with each stored pattern by changing several pixel values with opposite or intermediate values (black, white, or grey), see Figure 2.9. We use Hamming distance (HD) as a metric to measure the corrupted patterns' deviation from the stored ones. The HD between two patterns ξ^ν and ξ^μ of i elements is defined as:

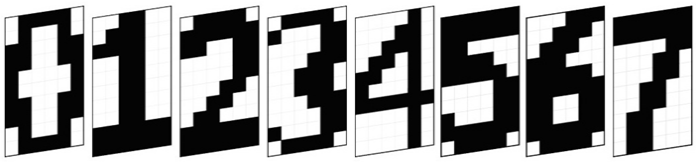
$$HD = \frac{1}{2}\sum_{i}(\xi_i^{\nu} - \xi_i^{\mu}) \tag{2.6}$$

In the test set, each stored pattern has four associated corrupted ones that are closer in their HD than any other stored patterns. Ideally, corrupted patterns are supposed to stabilize on the stored pattern with closer HD.

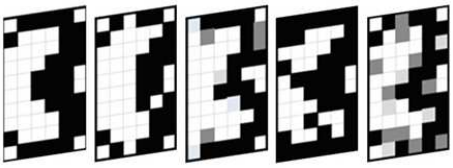
(a) *Training patterns of the 5x3 ONN.*



(b) *Test set associated with digit 1 for the 5x3 ONN.*



(c) *Training patterns of the 10x6 ONN.*



(d) *Digit 3 test set for the 10x6 ONN.*

FIGURE 2.9 – *Training and test patterns for digit recognition task.*

Simulation tests of the 5x3 OHN result in only 1 test pattern not being correctly retrieved. In contrast, after the simulation of the 10x6 OHN, 5 test images out of 25 do not converge to their respective stored pattern achieving a 20% error rate. It corresponds to the performance of a classical HNN performing the same task and confirms the digital OHN's equal capability to perform pattern recognition as HNN.

We also perform post-place and route simulations to characterize the OHN design following Vivado´s default strategies for synthesis and implementation. We set the target device to the Xilinx 7-series FPGA, the XC7Z020-1CLG400C since it is used for implementation afterward. We extract the maximum OHN input and operating frequency, as well as resource utilization of the two OHN sizes, 5x3 and 10x6. The input frequency corresponds to the clock input of the digital OHN, and the operating frequency corresponds to the frequency of the oscillators. Table. 2.1 indicates that both frequency and logical resources are highly dependent on the number of neurons. The smaller the network is, the higher the system clock frequency can be.

TABLE 2.1 – *Frequency limits and resource utilization estimated in simulation for Xilinx 7-series FPGA.*

Design size	**Max. input frequency**	**Max. operating frequency**	**LUTs** (%)	**Flip-Flops** (%)
5x3	83,33 MHz	5,21 MHz	958 (1.8)	721 (0.68)
10x6	64,10 MHz	4,00 MHz	6426 (12.08)	2756 (2.59)

2.5.2 Implementation evaluation of the digital OHN

Then, we implement the digital OHN design inside a Zybo-Z7 Digilent development board [179] and test the pattern recognition application. The board has many communication ports, memory spaces, user interaction tools, and a Xilinx Zynq-7000 system on chip (SoC). The SoC

integrates a dual-core ARM Cortex-A9 processor and the XC7Z020-1CLG400C, a Xilinx 7-series FPGA. Only FPGA resources are necessary for the digital OHN implementation.

Figure 2.10 shows the system-level architecture, including the digital OHN design, for performing pattern recognition on FPGA. The architecture includes the digital OHN described previously and a scheduler block to control it. The scheduler has four control blocks to monitor and check the OHN operations. First, the system clock is divided inside the slow clock block to ensure operations. Test patterns are stored inside the ONN controller and we use switches to select the input pattern. Next, the controller sends the input pattern to the OHN and waits until the end of OHN computation (*steady* signal activated). In the end, the ONN controller measures the OHN output state, applies a mask to identify the stored image, and the LED controller block turns *on/off* the corresponding LEDs, indicating correct and incorrect retrieval. The development board provides switches and LEDs needed by the architecture.

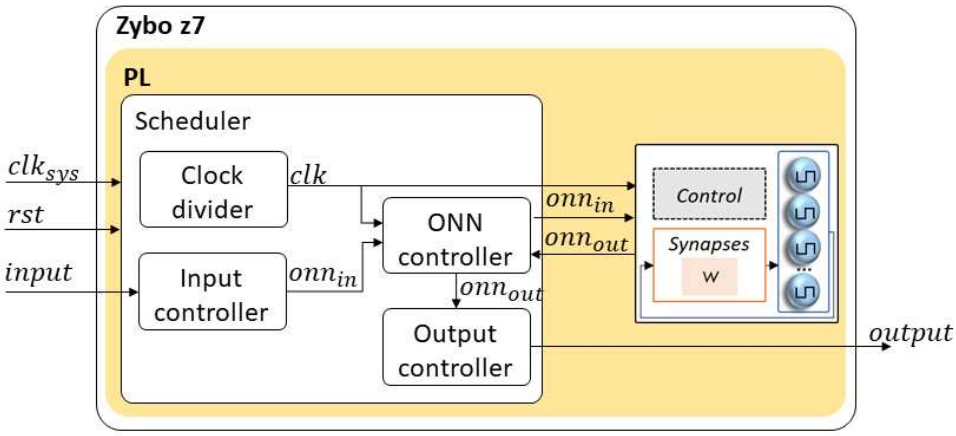


FIGURE 2.10 – *ONN FPGA implementation architecture for pattern recognition on Zybo-Z7 development board.*

First, we implement the OHN on FPGA performing equal pattern recognition tests as in simulation and comparing results. Table. 2.2 shows the multiple stored pattern combinations used for the 5x3 OHN and the 10x6 OHN implementation tests. As with simulation, the test set includes stored patterns with four corrupted versions of each, see Figure 2.9.

Table 2.3 presents the OHN implementation results for multiple sizes and training configurations. First, implementation and simulation tests give equal results for equal configurations. Also, the error rate increases with the number of stored patterns, such as with classical HNNs, validating the digital OHN implementation.

TABLE 2.2 – *Pattern combinations used for implementation characterization.*

ONN size	Number of stored patterns	Patterns
5x3	2	$0, 1$
5x3	2	$0, 2$
5x3	2	$1, 2$
5x3	3	$0, 1, 2$
10x6	3	$0, 1, 2$
10x6	4	$0, 1, 2, 3$
10x6	5	$0, 1, 2, 3, 4$
10x6	6	$0, 1, 2, 3, 4, 5$
10x6	7	$0, 1, 2, 3, 4, 5, 6$
10x6	8	$0, 1, 2, 3, 4, 5, 6, 7$

TABLE 2.3 – *Experimental error rate of the 5x3 and the 10x6 ONN implemented on FPGA for various training configurations.*

ONN	Stored patterns	Test images	Errors	Error rate (%)
5x3	0,1	10	0	0
5x3	0,2	10	0	0
5x3	1,2	10	0	0
5x3	0,1,2	15	1	6.67
10x6	0,1,2,3	20	0	0
10x6	0,1,2,3,4	25	5	20
10x6	0,1,2,3,4,5	30	9	30
10x6	0,1,2,3,4,5,6	35	21	60
10x6	0,1,2,3,4,5,6,7	40	35	87.5

Digital OHN frequency

We also evaluate the maximum operating frequency, corresponding to the maximum oscillation frequency of the digital OHN oscillators. Previously, we extracted the maximum clock frequency from the post-place and route simulation in Vivado, and here we try to confirm it from the OHN implementation by applying different frequencies and checking the error rate on the pattern recognition task.

OHN operating frequency is defined and generated by the system clock $clk_{sys} = 125MHz$ divided by multiple factors. A first configurable factor creates the OHN input clock clk, then the clk frequency is divided by 2 in the digital OHN design to ensure operation, and the clk divided by two is then used as a base signal to generate the oscillating signals, dividing by the period factor, in this case 16. Thus, OHN input frequency choice is limited by the configurable factor that allows the division of the frequency by multiples of 2. We perform FPGA implementation experiments on the same range of input frequencies to check if implementation and simulation results match. First, we set the OHN input frequency to 7.8125 MHz which is much lower than the frequency estimated by simulation static timing analysis. Then, we modify the configurable factor to try higher frequencies, up to the maximum input frequency of 125 MHz, corresponding to a configurable factor of 1.

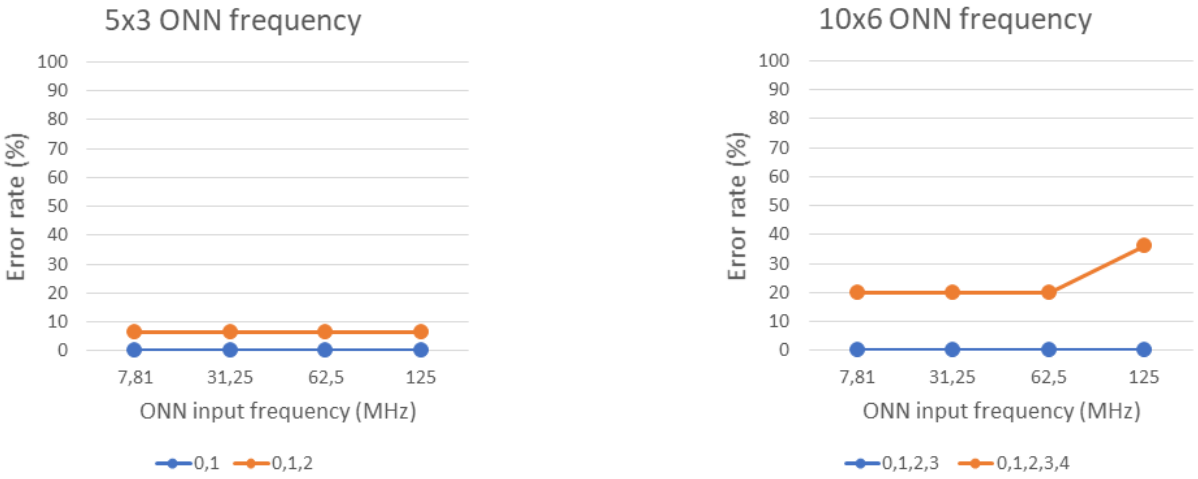
We use the error rate (E_R) metric to check the ONN operation on the previous pattern recognition test set. It is computed as:

$$E_R = \frac{\epsilon}{I_{tests}} \tag{2.7}$$

with ϵ the number of errors, and I_{tests} the number of test images. We consider an output as an error when the retrieved pattern does not correspond to any stored ones or when the ONN does not stabilize.

Experiments on 5x3 OHN perform similarly at 7.8125 MHz, 62.5 MHz, and 125 MHz as shown in Figure 2.11a. Thus, digital OHN can run a given test set at a higher input frequency than the evaluated limit ($83,33$MHz) for all tested training configurations. The difference in frequency can be explained by the way they are measured – in simulation, frequency was evaluated with global static timing analysis, whereas in experiments, frequency is evaluated on a specific test set. Figure 2.11b shows the 10x6 OHN error rate at different input frequencies for two training configurations. There is a trade-off between error rate, input frequency, and

training configuration. Also note, that for each training configuration, only an input frequency of 125 MHz impacts the 10x6 OHN error rate.



(a) **(b)**

FIGURE 2.11 – *Experimental results of the error rate (%) for various frequencies (MHz) and training configurations (training patterns) for* ***(a)*** *the 5x3 ONN, and* ***(b)*** *the 10x6 ONN.*

Considering simulation and implementation results, we assess the 5x3 OHN maximum operating frequency as 976.6 KHz, and the 10x6 OHN maximum operating frequency as 488 KHz. In the next experiments, related to time measurements, we define a common operation frequency for 5x3 and 10x6 OHNs to 488 KHz as it is the highest common operating frequency.

Digital OHN Computation Time

Then, we assess the computation time of the digital OHN with latency measurements. We experimentally measure the OHN initialization time needed to apply the input image, and the computation time needed to stabilize to a correct output state after initialization. The computation time is measured from the end of the initialization process to the *steady* signal's activation time. We measure OHN timings for training configurations that showed a 0% error rate to avoid inconsistent cases when the OHN does not converge. For inconsistent cases, the maximum computation time is clamped to 10 oscillating cycles. We experimentally measure the initialization time (t_{init}), and computation time (t_{comp}) of the OHN with an oscilloscope to calculate the number of frames per second (FPS) that OHN implemented on FPGA can treat. It is calculated as:

$$FPS = \frac{1}{t_{init} + t_{comp}} \tag{2.8}$$

Table 2.4 shows computation time, initialization time, and FPS results. It highlights that the serial initialization process increases the initialization time linearly with the increase of the OHN size. In comparison, the OHN computation time only slightly increases with size. It is an attractive feature of the ONN computing paradigm in which convergence is achieved in a few oscillation cycles independently of the number of synapses and neurons. It is also worth mentioning that the degradation of FPS performance is due to the initialization time. It could be mitigated by using a different initialization approach.

TABLE 2.4 – *Digital OHN timing performances for various sizes (number of neurons).*

OHN	Stored patterns	Initialization time	Computation time avg (us)	FPS (us)
5x3	[0,1]	2	5.04	142045
10x6	[0,1,2,3]	7.8	5.2	76923

Digital OHN resource utilization

We use Vivado post place and route tools to extract OHN resource utilization for various sizes. Table 2.5 shows that the 5x3 OHN design requires nearly ten times fewer resources than the 10x6 OHN design. It highlights one of the main digital OHN limits. An increase in the size drastically extends OHN's logical resources and depending on the FPGA, the number of neurons will be limited. For the given FPGA, the maximum OHN that can be implemented contains around 140 to 150 neurons.

TABLE 2.5 – *Resource utilization reported for multiple OHN sizes for Xilinx 7-series FPGA.*

#Neurons	#Synapses	LUTs (%)	Flip-Flops (%)
15	225	900 (1.7)	721 (0.68)
60	3600	6300 (12)	2756 (2.59)
100	10000	30033 (56)	4985 (5)
120	14400	38372 (72)	5970 (6)
140	19600	46900 (88)	6955 (7)
150	22500	65251 (123)	7447 (7)

Digital OHN power consumption

Finally, we extract the OHN power consumption from Vivado post place and route tools for various OHN sizes. For a fair comparison, we compute the energy per oscillation by considering the OHN oscillating frequency at 488 KHz. Table 2.6 showcases the power, the energy per oscillation, and the energy per computation considering a mean of 4 oscillation cycles per computation. It highlights that the energy per oscillation is similar for the three OHN sizes, and the energy per computation increases with the network size as it uses more oscillators in parallel.

TABLE 2.6 – *Power and energy consumption of the digital OHN for various sizes.*

#Neurons	15	60	140
Power	2 mW	11 mW	18 mW
Energy/osc.	67.7 pJ	93.1 pJ	65.3 pJ
Energy/comp.	4 nJ	22.3 nJ	36.6 nJ

2.6 Digits recognition from a camera stream

To prove the OHN's capability to perform real-world applications, a 10x6 OHN is implemented on FPGA inside a complete digit recognition system. We use a camera to stream OHN input images representing digits. Input images are displayed by a phone to the camera with a dedicated application. The camera is connected to the development board and sends input

images to the OHN. When the computation time is over, the output pattern is displayed on an external screen, see Figure 2.12.

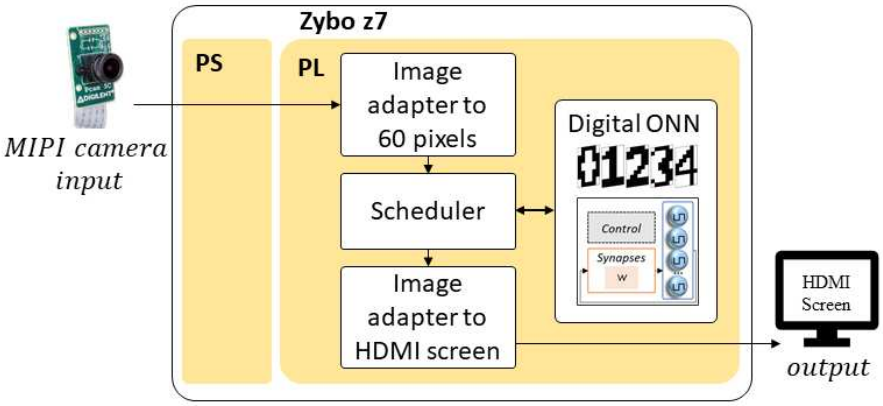


FIGURE 2.12 – *System-level architecture for digits recognition application.*

We use a Pcam 5c [180] camera which is connected via a MIPI camera serial interface 2 (MIPI CSI-2) with the Zybo-Z7 development board. We connect the external screen to the Zybo-Z7 via high definition multimedia interface (HDMI) communication. The image streaming from the camera to the screen comes from a Digilent Github project named Zybo-Z7-Pcam-5c [181], compatible with Xilinx's Vivado software 2018.2. We embed the digital OHN inside the image treatment flow. To do so, we binarize the camera's image in black-and-white and scale it down to 10x6 pixels. We rescale OHN output into a 1280x720 pixels image to display it on the screen. Both re-scaling steps use the Vivado HLS tool from Xilinx. We also use the ARM processor resources of the Zybo-Z7 development board to configure the camera. Based on the previous results, we configure the OHN with five stored digits, from 0 to 4, using the Hebbian learning rule. The test set comprises five trained images and 20 corrupted images, similar to the digital OHN characterization, so we expect equal results.

We use the output HDMI screen to identify recognized images and errors in the digital OHN image recognition application. We find five images not correctly recognized, see Figure 2.13. Output patterns displayed on the screen reveal that for each not-correctly recognized image, OHN is close to a correct reference pattern with only a few pixels wrong. However, the reference pattern is not necessarily the expected one, as for the image $3x$, we expected a digit 3, but the result is closer to a digit 2. Some errors can be explained by test images that correspond to corrupted digits too far in their HD from reference patterns.

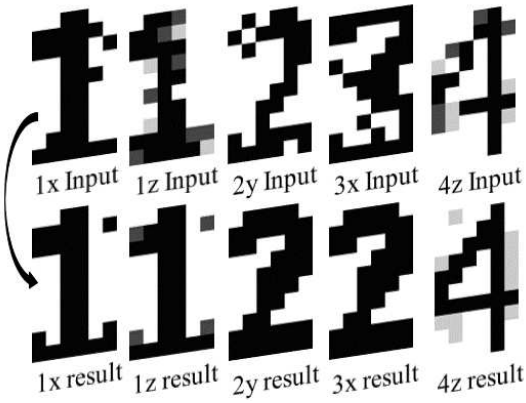


FIGURE 2.13 – *Representation of error images for 10x6 OHN digit recognition application showing incorrect pixels.*

With this application, we demonstrate the feasibility of using an ONN inside a complete design. We reach a 20% error rate with this application using Hebbian weights, but other learning algorithms might provide better accuracy. Besides, the 10x6 OHN takes $7.8\mu s$ to be initialized, $5\mu s$ to stabilize on average, and $160\mu s$ if it does not stabilize. As the camera provides an image every $15ms$, the 10x6 OHN does not create any latency for the image recognition application, so it respects real-time requirements. It validates the digital ONN design as a solution for real-time image recognition applications and encourages to look into other embedded applications.

2.7 Discussion and conclusion

In this chapter, we presented a proof of concept of the ONN neuromorphic computing paradigm configured with a fully connected architecture for pattern recognition, as an OHN, with a fully digital design. We validated the computing capability of 5x3 and 10x6 digital OHNs trained with the Hebbian learning rule performing pattern recognition both in simulation and FPGA implementation. Then, we embedded the 10x6 OHN into a complete image recognition application performing real-time digit recognition from a camera stream.

Here, we first highlight the advantages and limitations of the digital OHN and compare the digital design with alternative ONN implementations. The limited OHN size, as well as the specific architecture and learning algorithm used in the OHN, makes it challenging to compare with other neural networks, such as SNNs. Thus, we compare the digital design with other implementations of fully connected ONNs, in terms of size, oscillating frequency, computation time, and power consumption. Computation time corresponds to the number of oscillating cycles as it is dependent on the frequency of the system. Table 2.7 presents the performances of various analog or mixed-signal OHN implementations. Note, some are applied to AAM tasks [138, 182, 141] as the OHN digital design [170], while others are applied for COPs [161, 183, 184] such as Max-Cut. In the case of COPs, we take performances for graphs with dense connectivity, between 75% and 90%.

First, an interesting feature of the digital OHN in comparison with [141] is that by resorting to the 1-bit oscillation at the neuron's output, multipliers are avoided in the synapses block, while still retaining a multi-level neuron. This is possible because we encode the state in the neuron's oscillation phase. Then, the main limit of the digital OHN concerns the digital resources utilization (LUTs and Flip-Flops) which limits the size of the OHN. The current digital OHN design implemented on the $XC7Z020-1CLG400C$ FPGA is limited to around 140 neurons. In OHN, the block using the most resources is the synaptic block due to the important number of synaptic elements even after the modification of the digital OHN with a partially sequential synaptic computation instead of the fully combinatorial synaptic computation that was processed at first. Other alternative solutions to increase the OHN size could be to reduce the weight precision, from 5-bit signed weights to smaller precision, or to limit the number of weights, resulting in changes in the ONN architecture. However, even if the size is limited, Table 2.7 highlights that the digital OHN design achieves the largest implementation of an ONN with fully connected architecture reported in the literature. Additionally, this architecture provides the largest number of synaptic elements as it implements N^2 weights with self-coupling and double weights per synaptic element considering both directions while some of the compared architectures only consider $N(N-1)/2$ weights [138]. Implementing more weights with a larger precision allows more flexibility and can sometimes help to increase accuracy while it also requires more computational resources, limiting the network size no matter the implementation. In the next chapters, we explore novel ONN architectures to avoid the large number of synaptic elements and allow novel ONN applications.

TABLE 2.7 – *Comparison of the digital OHN with other fully-connected ONN implementations.*

	[138] 2011	[182] 2016	[141] 2018	[161] 2020	This work 2021	[183] 2023	[184] 2023
Neurons	8	20	100	30	140	16	48
Synapses	56	400	10000	900	19600	256	2304
Frequency	NA	1 GHz	1 GHz	45 KHz	488 KHz	1 MHz	25 kHz
Comp. cycles	1000	10	4	NA	3-5	100-150	250
Power	NA	226.5 μW	303 mW	1.76 mW	18 mW	10 μW	105 mW
Energy/osc	NA	0.011 pJ	0.3 pJ	1.3 nJ	65.3 pJ	0.6 pJ	87.5 nJ
Application	AAM	AAM	AAM	COP	AAM	COP	COP

In contrast, a fundamental advantage of the ONN paradigm is the fast computation. We previously highlighted the short-time computation required by the digital OHN. We can compute each pattern with an average of 5μs at 31.25 MHz input frequency for both 5x3 and 10x6 OHNs. So, the computation time is independent of the network size, which is an important feature to further explore design methods to upscale its size. However, the current OHN design uses a serial initialization of the neurons, which depends on the number of neurons, achieving more than 70000 FPS with the 10x6 OHN. Note, that the time required to run a task on the ONN is highly dependent on the frequency of the oscillators and the task it tries to solve. ONNs with faster frequency will compute faster. However, depending on the task to solve, ONNs do not require the same number of oscillation cycles to settle. ONNs configured for COPs require more oscillation cycles to compute than ONNs configured as OHNs. The digital OHN computes in the same number of computation cycles as other OHN implementations, but faster than the ONNs tested on COPs. However, the oscillation frequency of the digital OHN is quite low compared to other OHN implementations, even if higher than the ONN configured for COPs.

The latency of computation also impacts the energy consumption per operation. The digital OHN is not optimized to be highly low-power compared to ONNs developed on application-specific integrated circuits (ASICs). However, we believe it is still important to compare the various ONN implementations in terms of power and energy consumption. To do so, we derive the energy per oscillation cycle per oscillator to compare with the same baseline as we saw the computation latency defers depending on the application and the network size. If we compare the energy per oscillation, we see that the digital OHN design is in the same range as the other ONN implementations. All ONN implementations are in the range of the nJ or pJ per oscillation per oscillator.

Another advantage of the OHN is the easiness of training. Results reported in this chapter have been obtained using the Hebbian learning rule. Given the patterns to be stored and recognized, weights are calculated offline by using matrix operations, while training other neural models can be a very time-consuming operation. However, such low computation unsupervised learning rules have limited retrieval capacity and make ONN complex to compare with standard benchmarks. This is the focus of the next chapters, starting by exploring novel learning solutions for the OHN, before exploring novel architectures for novel edge AI applications. In particular, alternative learning rules have already been explored for HNN to improve the accuracy of AAM tasks, and in the next chapter, we explore novel unsupervised and supervised learning algorithms to improve the OHN precision on AAM tasks. Additionally, another challenge is to be able to perform learning continuously through life in evolving environments. In the next chapter, we also propose a solution to perform on-chip learning with unsupervised learning rules to allow continual learning capability.

The work presented in this chapter resulted in one publication in a scientific journal [170].

CHAPTER 3

OHN LEARNING FOR AUTO-ASSOCIATION

Contents

3.1 Introduction

Learning is essential to configure correctly a neural network to solve a specific task. Existing learning algorithms to train an OHN for pattern recognition are mainly unsupervised learning rules, which were first introduced for HNNs. In particular, the Hebbian learning rule, which was introduced first, is still the simplest to compute and has been widely applied to OHN. However, it has a really low capacity. The capacity of a network configured for pattern recognition or AAM is defined as its ability to learn and retrieve its training patterns from corrupted input information. It can be associated with the accuracy, precision, or error rate of a system. Thus, there is a need to explore alternative unsupervised learning rules for OHN. In this Chapter, in Section 3.2, we study the HNN state-of-the-art learning rules and how to apply them to OHN for AAM.

Furthermore, AI models need to learn continuously through time to adapt to evolving environments [31, 32, 33]. Efforts are currently concentrated first on supervised continual learning [185, 186] to improve the performance of classification models over time, and then on continual reinforcement learning to learn from the environment, for example in robotics [187, 188]. Continual learning algorithms expect to learn novel data while avoiding catastrophic forgetting [34, 35] of previously learned data. Additionally, continual learning demands to be implemented on-chip for fast and efficient performances. However, to allow continual on-chip learning, each synapse needs to be re-programmable in a real-time latency requiring additional space, and resources, and consuming more energy than systems without on-chip learning. In this Chapter, in Section 3.3, we also propose a system capable of performing OHN continual on-chip learning using the digital OHN design. We implement unsupervised learning algorithms compatible with on-chip learning and measure the performances of the system.

In parallel, we explore beyond unsupervised learning algorithms for OHN to improve AAM capacity and solve novel image classification tasks. Image classification is a primary computer vision task deployed in many industrial systems, such as healthcare or manufacturing systems. It is usually solved with conventional AI algorithms [10] trained with gradient back-propagation learning algorithms. However, back-propagation is computation-intensive, making it incompatible with hardware edge implementation. Recently, Scellier [58] adapted the contrastive Hebbian rule (CHR) [189] to perform supervised learning with energy-based RNN models, using the so-called hardware-compatible equilibrium propagation (EP) learning algorithm. Thus, in this Chapter, in Section 3.4, we adapt EP to the single-layer HNN and OHN configured for AAM, creating the AAM-EP supervised learning algorithm, and we apply it to a simplified MNIST image classification task using an HNN on Matlab and the digital OHN design.

3.2 Unsupervised learning for binary OHN

Unsupervised learning algorithms only use learning patterns to compute coupling weights, without additional feedback, unlike supervised learning algorithms, and are mainly used to solve clustering problems. In pattern recognition or AAM, each pattern becomes the point of attraction of various clusters created in the energy landscape, see Figure 3.1a.

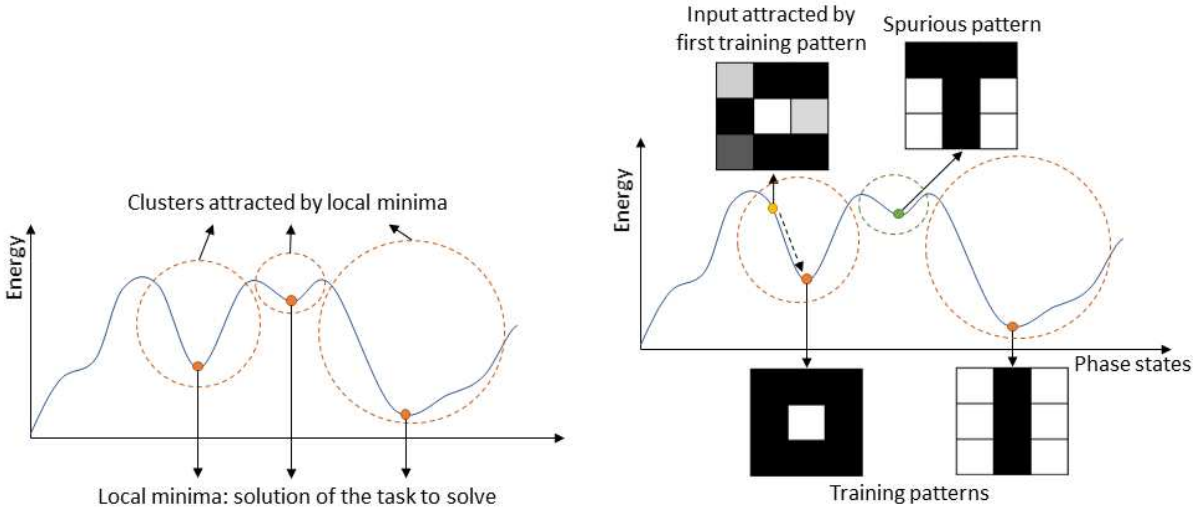


FIGURE 3.1 – *Simplified representation of an energy landscape for (a) a global interpretation, and (b) an interpretation in the case of pattern recognition.*

There exist mainly two features to categorize unsupervised learning rules for pattern recognition: locality which means that the update of the coupling weight between neuron i and neuron j only depends on activation values of neurons i and j on both sides of the synapse, and incrementality, which means that the update of the weights can be done pattern by pattern without forgetting previously learned patterns. The locality feature is often important to limit computation, for example for continual learning. The incrementality feature is important to be able to learn patterns one at a time while remembering previous patterns in the weight matrix. Then, learning rules can be characterized by the number of iterations they require to learn a single pattern. For example, some learning rules require only one iteration per learning pattern, so-called immediate learning rules, while others necessitate multiple iterations per learning pattern to be solved, so-called iterative learning rules. Finally, additional constraints on

the weight matrix might be imposed by the network hardware implementation, such as weight symmetry and self-coupling, corresponding to non-zero diagonal. In this Section, we discuss the main HNN learning rules, and how to adapt them for OHN. Also, we perform tests with adapted learning rules with the digital OHN.

3.2.1 State-of-the-art HNN unsupervised learning rules

The first unsupervised learning algorithm introduced for HNN to perform pattern recognition is the Hebbian learning rule [28]. It is inspired by the biological rule: "*Neurons that fire together, wire together*". The Hebbian learning rule is local, incremental, and immediate and is defined by a simple matrix multiplication such as the weight between neuron i and neuron j follows:

$$w_{ij} = \frac{1}{N} \sum_{k} \xi_i^k \xi_j^k \tag{3.1}$$

with k the number of learning patterns, ξ^k the k^{th} training pattern, and N the number of neurons in the network. The Hebbian learning rule is attractive for its speed and simplicity of computation while it has a limited absolute capacity $C = \frac{N}{2ln(N)}$, often approximated to $C = 0.14N$. The limited capacity of the Hebbian learning rule has motivated the investigation of alternative learning algorithms. First, the pseudo-inverse learning rule, also called the projection rule, was introduced in 1986 to guarantee the stability of the learning states by using a projection matrix. It reaches higher capacity than Hebbian, with $C = N$ [190] and is defined as:

$$w_{ij} = \frac{1}{N} \sum_{\nu=1}^{k} \sum_{\mu=1}^{k} \xi_i^\nu (Q^{-1})^{\nu\mu} \xi_j^\mu \tag{3.2}$$

with $Q = \frac{1}{N} \sum_{i=1}^{N} \xi_i^\nu \xi_i^\mu$. However, even if it is immediate, the pseudo-inverse learning rule is neither local nor incremental, making it unattractive for continual learning for example. In parallel, Diederich and Opper proposed two learning rules in 1987 [191], the Diederich-Opper rule I (DOI), and the Diederich-Opper rule II (DOII). DOI and DOII are local learning rules using a stability condition to verify if a pattern is correctly saved in the network. Thus, when learning a novel pattern, the stability condition is applied to the novel pattern and the previously learned patterns until the condition is met for all patterns. DOI and DOII are iterative learning rules but are not incremental. The stability condition checks if when the pattern is applied as input of the network, the update of the network retrieves correctly the pattern. For DOI, the condition applied is $\xi_i h_i >= T$ with $h_i = \sum_{j=1}^{N} w_{ij} \xi_j$ the local field of neuron i, and T a positive threshold often set to 1. Then, if the condition is not met, the update rule for the weight between neuron i and neuron j is defined as:

$$w_{ij} = w_{ij} + (N-1)^{-1} \xi_i^k \xi_j^k \quad \text{for} \quad j \neq i \tag{3.3}$$

And for DOII, the condition and update rules are slightly modified with condition $\xi_i h_i = 1$ and if the condition is not met, the update rule for the weight between neuron i and neuron j is defined as:

$$w_{ij} = w_{ij} + (N)^{-1}(1 - h_i\xi_i)\xi_i^k\xi_j^k \tag{3.4}$$

Note, DOII also includes self-coupling, meaning $w_{ii} \neq 0$. The absolute capacity of DOI and DOII is not precisely defined, however, DOI and DOII highlighted better capacity than Hebbian in practice [178]. Alternative local iterative learning rules [192, 193] were also developed in the same decade with better capacity than Hebbian. However, as mentioned before, iterative learning algorithms require more computation than most of the immediate learning rules and are often not incremental. Thus, later, the Storkey learning rule emerged as an attractive learning rule to improve the HNN capacity [194]. It was proposed in 1997 to improve the Hebbian learning rule capacity while keeping locality, incrementality, and immediacy. The Storkey learning rule is defined by:

$$w_{ij} = \sum_k \frac{1}{N}(\sum_{i=1}^{N}\sum_{j=1}^{N}\xi_i^k\xi_j^k - \xi_i^k h_{ji} - h_{ij}\xi_j^k) \tag{3.5}$$

with $h_{ij} = \sum_{l=1,l\neq i,j}^{N} w_{il}\xi_l$, a local field. Next, we study how to adapt the previously described learning rules to fit with OHN constraints.

3.2.2 HNN unsupervised learning rules adapted for OHN

Adapting HNN unsupervised learning rules to OHN requires several constraints due to hardware implementation constraints.

Originally, in HNN trained with Hebbian, the weight matrix is symmetric, meaning weights between two neurons in both directions have the same values, and the weight matrix diagonal has zero values avoiding self-coupling, to ensure stability. Later, to improve precision and capacity, novel unsupervised learning algorithms were introduced allowing asymmetric weight matrix [191, 192, 193] and self-coupling [191]. However, most OHN implementations, in particular analog ones, do not support self-coupling and non-symmetric weights as the coupling is often implemented with discrete analog components like resistors or capacitors [195]. We provide a classification of the unsupervised learning rules in Table 3.1.

TABLE 3.1 – *HNN learning rules features.*

Learning rules	Weight symmetry	Zero-diagonal	Local	Incremental
Hebbian [28]	x	x	x	x
Storkey [194]	x	x	x	x
Pseudo-Inverse [190]	x	x		
Diederich Opper I [191]		x	x	
Diederich Opper II [191]			x	
Gardner [192]		x	x	

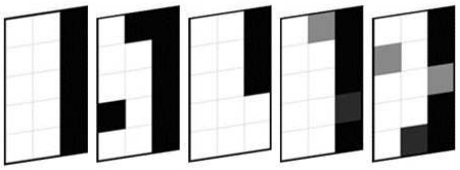
Most unsupervised learning algorithms introduced for HNN can be modified to be used with OHNs by adding constraints on the weight matrix, even if it can impact negatively the network capacity [178]. Consequently, even if the digital OHN supports non-symmetric weights and self-coupling, and there are ongoing efforts to develop alternative analog ONN designs

to allow self-coupling and non-symmetric weights [183], in the next section, we apply the previously mentioned learning rules imposing weight symmetry and no self-coupling to the digital OHN to compare the accuracy of the different learning rules. Note, using unsupervised learning algorithms introduced for classical HNN limits patterns to binary information while OHN with its continuous phase values could, in principle, stabilize to non-binary patterns corresponding to phases between 0^o and 360^o.

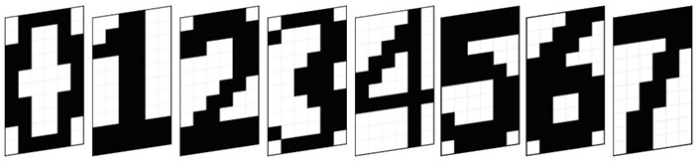
3.2.3 Comparison of unsupervised learning rules for digits recognition application

We apply unsupervised learning rules introduced for HNN to the digital OHN and compare them with an HNN simulated on Matlab. We consider the 10x6 digits recognition application with 10 training patterns representing digits from 0 to 9, and 5 test patterns for each training pattern, see Figure 3.2. We compare learning configuration with 1 pattern (digit 0), 2 patterns (digit 0 and 1), up to 10 patterns (digits 0 to 9), and test with corresponding test patterns. We report on the accuracy, considering test images for which the network retrieves the correct output pattern, on true negative patterns, for test images that stabilize to a training pattern but the wrong one, on spurious patterns, meaning test images that stabilize to an unknown pattern, and on inconsistent patterns, when a test image never stabilizes. Note, that inconsistency never happens with HNN, and true negative never happens with OHN in this test case but it could happen for other test cases.

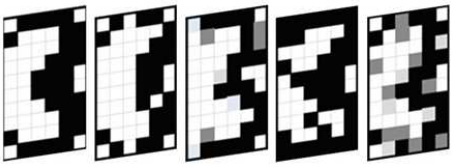
(a) *Training patterns of the 5x3 ONN.*



(b) *Test set associated with digit 1 for the 5x3 ONN.*



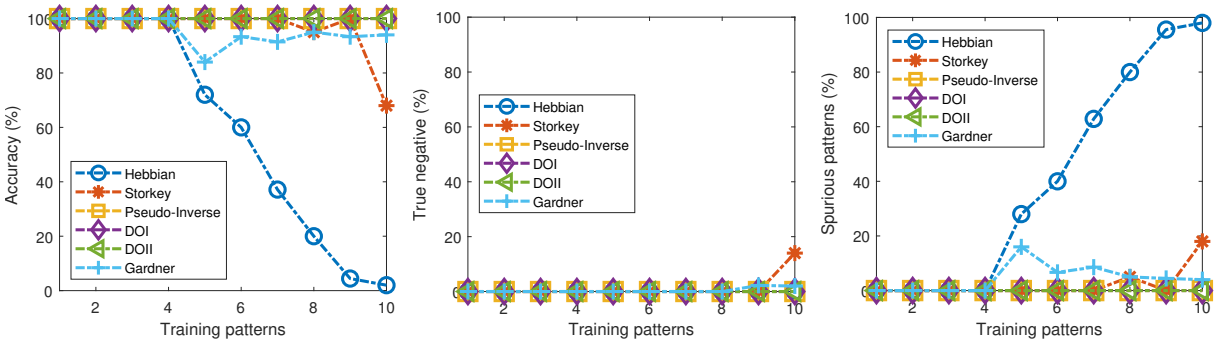
(c) *Training patterns of the 10x6 ONN.*



(d) *Digit 3 test set for the 10x6 ONN.*

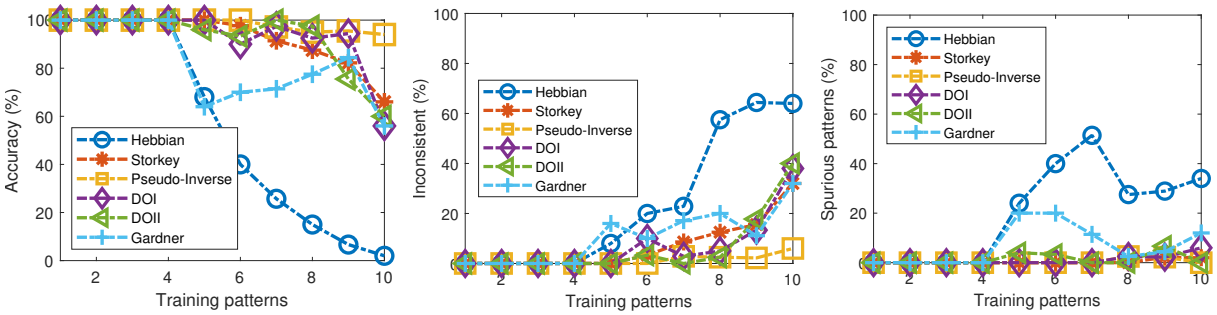
FIGURE 3.2 – *Training and test patterns for the digit recognition task.*

Figure 3.3 highlights that almost all learning rules perform well on all learning configurations for the HNN, in particular pseudo-inverse, DOI, and DOII. Hebbian is able to learn 4 patterns, like Gardner, however, the accuracy drops rapidly afterward. Accuracy in the digital OHN has similar behavior to HNN but achieves lower accuracy, see Figure 3.4. This is mainly due to the reduction of the weight precision to 5-bit signed. The learning rule achieving the best accuracy results for OHN is the pseudo-inverse. Note, that accuracy also depends on the training patterns, thus our results are only true for this data set. It explains also why for some learning rules, like Gardner, accuracy is better for learning configuration with 9 patterns than with 5 patterns. Some configuration are harder to learn due to the correlation between patterns and accuracy does not only depends on the number of training patterns but on the training patterns themselves. Additionally, Figure 3.4 highlights that when HNN output is incorrect, it corresponds to a spurious output, while with the digital OHN, it is divided between spurious patterns and inconsistency.



(a) *Accuracy* **(b)** *True negative* **(c)** *Spurious*

FIGURE 3.3 – *HNN results from Matlab simulation of the 10x6 digits recognition application.*



(a) *Accuracy* **(b)** *Inconsistent* **(c)** *Spurious*

FIGURE 3.4 – *Digital OHN results on the 10x6 digits recognition application.*

In this section, we highlighted that OHN supports various unsupervised learning rules introduced for HNN. However, all learning rules are applied offline using software to generate the weights, before implementing them in the digital OHN hardware. As mentioned previously, in some applications, continual learning can be necessary to adapt to evolving environments, such as in robotics. Thus, in the next section, we study how to perform continual on-chip learning with the digital OHN design for pattern recognition.

3.3 Implementing on-chip learning for binary OHN

We define OHN on-chip learning for pattern recognition as the ability of an ONN-computing system to learn new patterns by updating OHN coupling weights meanwhile avoiding catastrophic forgetting of previously memorized patterns. This Section provides a study of the state-of-the-art HNN unsupervised learning rules for OHN on-chip learning. Then, it introduces the on-chip learning system architecture compatible with the digital OHN design. Afterward, we implement different unsupervised learning rules in the system architecture and provide performances of the system in terms of capacity, resource utilization, latency, and energy consumption. We first implement a small-scale 5x3 OHN performing continual on-chip learning of three patterns representing digits. Later, we study the scalability of the system for larger OHN sizes and more general pattern recognition tasks.

3.3.1 Adaptation of OHN unsupervised learning rules with on-chip learning

As previously mentioned, there are two main features to categorize unsupervised learning rules for pattern recognition: locality and incrementality. Concerning continual on-chip learning, the locality is important to implement the update of the weights in each synapse and limit resource utilization. However, the locality is not mandatory as the update of the weights is not always integrated and implemented at the synapse level. Incrementality is important for continual learning to ensure remembering previously learned patterns, memorized in the weight matrix when learning a novel pattern. To avoid catastrophic forgetting, some algorithms require iterative learning but it uses more computing and memory resources, and it is often non-incremental, making it unsuitable for continual on-chip learning. Thus, when we consider OHN continual on-chip learning, we focus on local and incremental unsupervised learning algorithms introduced for HNNs to be compatible with OHN. Table 3.1 highlights that based on the learning rules studied in [178], there are only two unsupervised learning rules that satisfy the OHN continual on-chip learning constraints, Hebbian and Storkey. We implement both Hebbian and Storkey learning rules in a digital OHN on-chip learning architecture. Note, that adding learning capacity to every synapse can be costly in terms of resources, so it is important to also consider sparsity and small weight precision in the weight matrix. In particular, we study the impact of weight precision on OHN performances.

3.3.2 On-chip learning implementation

We propose a system capable of performing on-chip-learning using the digital OHN design in the Zybo-Z7 development board [179]. The Zybo-Z7 is based on a ZYNQ processor equipped with a processing system (PS), a dual-core Cortex-A9 processor, and programmable logic (PL) resources equivalent to an Artix-7 FPGA. First, for the OHN on-chip learning architecture, OHN digital design is implemented using PL resources as described in Chapter 2 and in [170]. However, instead of controlling the OHN from PL, we control the OHN from PS, see right part of Figure 3.5. Then, we modify the SoC architecture to include learning in PS and limit the PL resource utilization, see Figure 3.5.

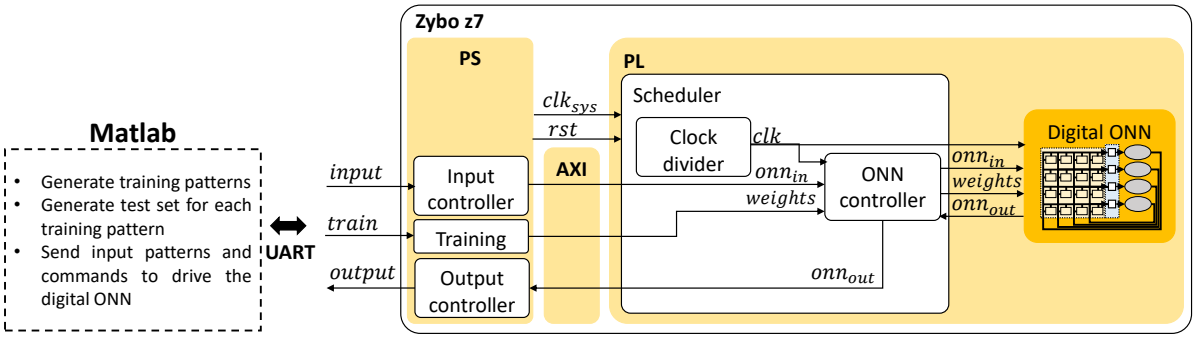


FIGURE 3.5 – *Architecture for OHN on-chip learning system for implementation tests.*

Communication between PS and PL uses the AXI4-lite parallel communication protocol. We use PS as master and PL as slave such that when PS receives an external command and pattern, it controls the digital OHN in PL. The learning process starts when PS receives an external learning command in parallel with an input pattern. It engages the update of the weights on PS following the implemented learning rule before sending the updated weights to the digital OHN in PL through the AXI4-lite bus. Note, that during weight update, OHN is in reset mode. Once the weight update is over, OHN comes back into inference mode and informs PS that the

weight update is done. The inference process starts when PS receives an input pattern with an inference command, such that PS transmits the input pattern through the AXI4-lite bus to the digital OHN in PL, the digital OHN infers, and it sends back its output pattern to PS through the AXI4-lite bus. Note, that an additional command performs a reset of the weights to zeros if necessary. AXI4-lite communication accesses four 32-bit AXI4 registers to send and receive information. The transmission latency during inference and training depends on the OHN size.

We implement the Hebbian and Storkey HNN learning rules in our digital OHN on-chip learning architecture and evaluate the performances through four metrics, resource utilization, capacity, latency, and energy consumption. We analyze the resource utilization of our OHN on-chip learning architecture as it determines the cost of implementation of our solution in hardware. Memory capacity is defined by the number of patterns a network (HNN or OHN) can correctly learn and retrieve. It can be evaluated by learning patterns in the network and verifying if the network retrieves the correct training pattern when one of the training patterns is presented. However, we believe it is also necessary to verify if the network can retrieve the correct training pattern from corrupted input information, corresponding to none of the training patterns, to evaluate the robustness to noise. Thus, we train the OHN with some patterns and test if it can retrieve them and corrupted versions of them. Then, we measure the latency for both the training process and the inference process. Finally, we extract energy consumption from post-place and route simulation in Vivado to compare with other OHN designs.

We start by evaluating the OHN on-chip learning architecture with a small-scale 15-neuron OHN trained on a simple digit recognition [196] task before studying the scalability of the on-chip learning architecture for larger ONN sizes [197].

3.3.3 Validation of on-chip learning with a 5x3 OHN

At first, we validate the OHN on-chip learning system architecture on a small-scale 5x3 OHN trained and tested on a simple dataset, see Figure 3.2. We use switches to represent input images and LEDs for the output pattern, both connected to PS. We initialize all weights to zeros and use a button as the learning command to start a learning process. When we push the training button, the PS starts processing the new weights using the current input image. After processing, all 225 weights are sent through AXI4-Lite to the OHN in PL. During experiments, we train our model with the three training patterns one by one and we test all 15 test images to assess the accuracy. We use both Hebbian and Storkey learning rules on the digital OHN configured with 5-bit signed weights.

Table 3.2 displays resources, accuracy, and timing characteristics of the on-chip learning solution compared with the off-chip learning OHN [170]. Note, we report and compare accuracy after learning the 3 memorized patterns. Table 3.2 validates the on-chip learning architecture as accuracy with on-chip or off-chip learning is equal. Furthermore, computation time changes from one learning rule to another depending on the calculation complexity. However, the transmission time is stable because the number of weights sent to PL depends on ONN size, stable in this case. We report a transmission time of 86 μs to send the 225 weight values, and computation time of 33μs for the Hebbian learning rule, and 77μs for the Storkey learning rule. Finally, our solution highly increases PL resource utilization in comparison with the off-chip learning solution due to the re-programmable synapses. However, the learning algorithm does not influence resource utilization. These experiments validate the OHN on-chip learning architecture for a small-scale 15-neuron OHN and provide a first intuition on the OHN on-chip learning architecture performances. However, as latency and resource utilization depend on

the OHN size, an additional study on the scalability of the system is necessary to assess more general performances of the OHN on-chip learning architecture.

TABLE 3.2 – *Performances of the on-chip learning design for 5x3 digits recognition compared to off-chip design.*

	Hebbian	Storkey
Resources on-chip		
- LUTs	8203 (15.42%)	8203 (15.42%)
- Flip-Flops	3305 (3.11%)	3305 (3.11%)
Resources off-chip		
- LUTs	958 (1.8%)	800 (1.5%)
- Flip-Flops	721 (0.68%)	721 (0.68%)
Accuracy on-chip	93.33%	93.33%
Accuracy off-chip	93.33%	93.33%
Learning time	119 μs	163 μs
- Computation	33 μs	77 μs
- Transmission	86 μs	86 μs

3.3.4 Scalability of the OHN on-chip learning architecture

We study the scalability of the on-chip learning architecture for larger OHN sizes [197]. Furthermore, resource utilization is limited by the re-programmable synaptic weights, so we study the impact of weight precision. We first evaluate HNN capacity on Matlab with Hebbian and Storkey learning rules for various HNN sizes and weight precisions. Then, we implement Storkey and Hebbian in the on-chip learning architecture to extract the resource utilization for various OHN sizes, and the capacity, latency, and energy metrics for a 25-neuron OHN. A test flow is set up and automatized for testing the digital OHN on-chip learning architecture using Matlab to send commands and patterns to the system through a universal asynchronous receiver transmitter (UART) communication protocol, see Figure 3.5.

HNN capacity for various size and weight precision

We evaluate the capacity of N-neuron HNN networks trained with up to N random training patterns, by testing with corrupted input patterns generated from training patterns with up to $N/2$ flipped pixels, represented by the hamming distance. An inference cycle is performed for each input pattern. Note, that the size of the network, as well as the correlation between the training patterns, impact the capacity of the network, so we perform 100 trials for each configuration. We study the impact of weight precision on HNN accuracy for various HNN sizes. In particular, we analyze the capacity of HNN trained with Hebbian and Storkey for three HNN sizes, 25, 50, and 100 neurons, as well as for five weight-precision, 2-bit, 3-bit, 4-bit, 5-bit, and full precision.

Figure 3.6 shows the HNN capacity for a 100-neuron HNN trained with Storkey with 1 up to 100 training patterns and tested for 100 trials with corrupted input patterns with 1 up to 50 Hamming distance (HD). A black pixel represents that over the 100 trials, for a given configuration, all tests were successful, while a white pixel points out that none of the tests were successful. The capacity lines highlight, for each number of training patterns, the maximum HD of corrupted input patterns supported by the network, such that the network successfully

associates the corrupted input pattern with a training pattern for at least θ trials over 100, with $\theta = \{85; 90; 85; 100\}$. Then, to simplify the readability of our results, we choose to represent only the capacity lines for one value of θ. We choose $\theta = 90$ to have results representative of a majority of cases and to allow some error tolerance.

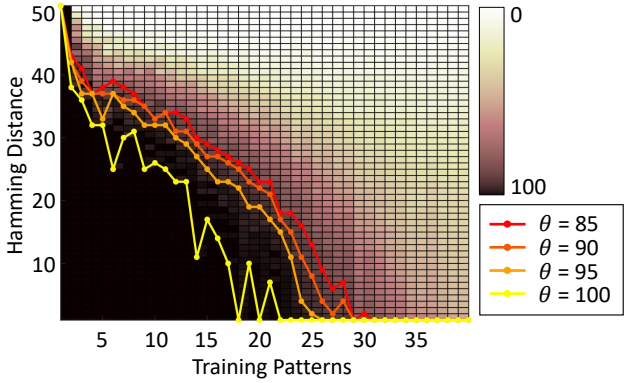


FIGURE 3.6 – *Capacity of a 100-neuron HNN trained with Storkey with 100 training patterns tested with corrupted input patterns with different hamming distances (1 up to 50 flipped pixels) with the training patterns. The capacity lines represent for each number of training patterns the maximum hamming distance of corrupted input patterns supported by the network, such that the network successfully associates the corrupted input pattern with a training pattern for at least θ trials over 100.*

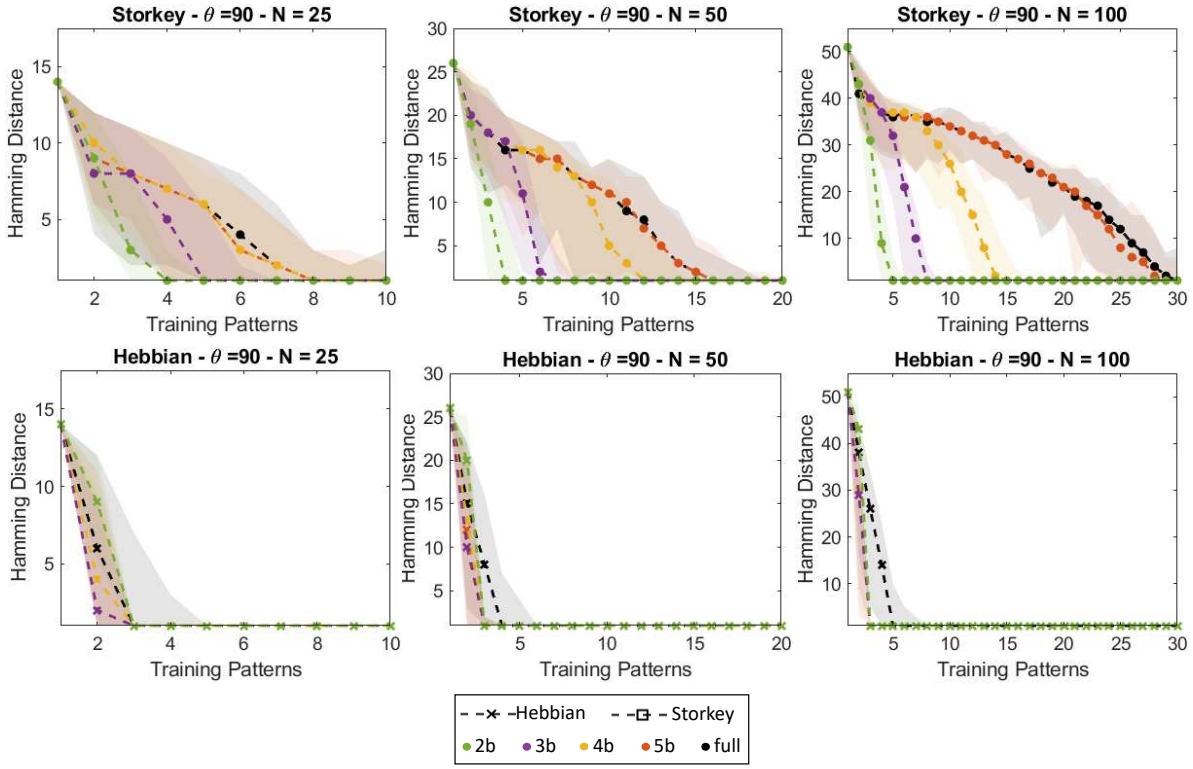


FIGURE 3.7 – *Capacity of HNN networks of N={25,50,100} neurons trained with Storkey, or Hebbian with various weight precision. The capacity is represented, for each network size, for each learning rule, and each number of training patterns, by the maximum hamming distance of corrupted input patterns supported by the network, such that the network successfully associates the corrupted input pattern with a training pattern for at least θ=90 trials over 100 (90%).*

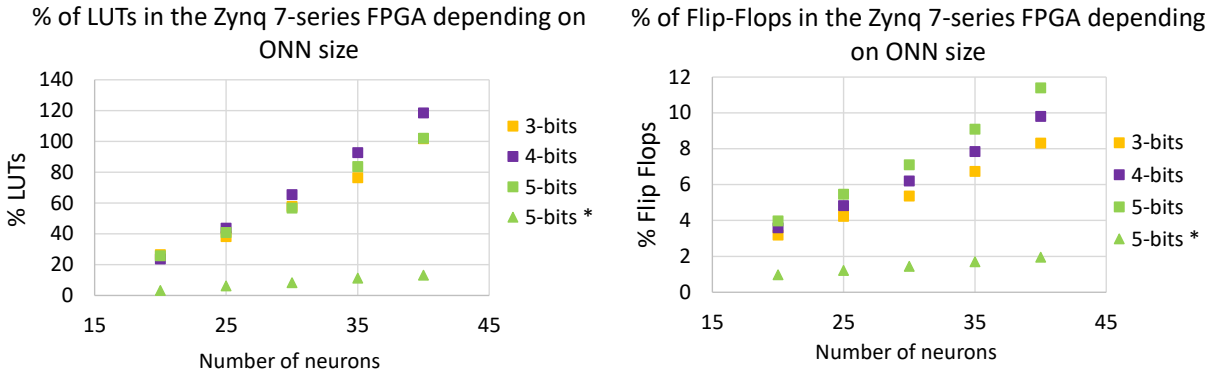
Figure 3.7 shows the HNN capacity lines for $\theta = 90$ for the Hebbian and Storkey learning rules for the different weight precisions and network sizes. Figure 3.7 also plots the error

bounds for each weight precision configuration. Figure 3.7 first highlights that HNN trained with Storkey can retrieve a larger number of training patterns when initialized with more corrupted input patterns, with larger HD, thus HNN trained with Storkey shows better capacity than HNN trained with Hebbian for all weight precision configurations, as it was already proven from literature [194]. Then, Figure 3.7 displays that for Storkey learning, using 5-bit weight precision, HNN obtains a similar capacity than considering full weights precision. Note, the impact of reducing weight precision to 4-, 3-, or 2-bit precision depends on the network size. The larger the network is, the more important the impact of the reduction of the weight precision on the network capacity.

OHN on-chip learning resource utilization

After, we implement Hebbian and Storkey learning rules in the digital OHN on-chip learning architecture and first study the impact on resource utilization. In the proposed architecture, a large number of LUTs are used as reconfigurable memory of the weight matrix, due to the fully connected OHN architecture. To limit the impact of re-programmable synapses, we analyze the impact of reducing the weight precision on resource utilization, and we expect the reduction of the weight precision to also reduce LUT utilization.

In Figure 3.8, we report on the number of LUTs, as well as the number of Flip-Flops necessary for various digital OHN sizes, for 3-, 4-, and 5-bit precision. Figure 3.8 indicates that for some OHN sizes, reducing the weight precision does not reduce the number of LUTs. For example, for the 35-neuron OHN, the number of LUTs is larger for the 4-bit precision than for the 5-bit precision. We believe it depends on the configuration of the FPGA, which provides fixed-size LUTs. Additionally, the reduction of the weight precision from 5 bits to 3 bits does not significantly reduce the resource utilization as expected, limiting the OHN size to up to 35 neurons for on-chip learning implementation. Next, we consider a 25-neuron OHN to report on its capacity and latency.



(a) *LUTs.* **(b)** *Flip-Flops.*

FIGURE 3.8 – *Resource utilization of the OHN on-chip learning architecture for various OHN sizes for different weight precision. We compare with the previous digital ONN with random hard-coded weights in a 5-bit precision (5 bits*).*

25-neuron OHN capacity for various weight precision

Figure 3.9 presents capacity lines obtained for a 25-neuron digital OHN trained on-chip with both Hebbian or Storkey for 3 different weight precision (3, 4, and 5 bits) compared with

HNN trained with the same configuration. Figure 3.7 shows that for Storkey, HNN and OHN have similar capacities. However, considering Hebbian, Figure 3.9 demonstrates OHN has a better capacity than HNN. Figure 3.9 also shows fewer OHN capacity variations depending on the weight precision than HNN. This was unexpected as it was not observed in previous configurations, but this is, to the best of our knowledge, the first large-scale capacity tests performed with the digital OHN. We believe the difference might come from the difference in the system dynamics between HNN and OHN. Classical HNN can only take two state values, $\{-1; 1\}$, because of the sign activation function. However, the OHN activation function allows it to take multi-state or continuous values during dynamical evolution. Thus, even if an OHN trained with binary patterns will stabilize to binary phase states $\{0^o; 180^o\}$, the activation function, which is difficult to derive, allows non-binary phase states during phase dynamics. We believe that the phase dynamics of the OHN evolve slowly from a corrupted input pattern to the correct training pattern, while the sharp HNN activation function may evolve too fast, reaching the wrong training pattern. HNN may require more precise weights, as with Storkey, to take the correct decision, while the OHN can still evolve to a correct training pattern even with less precise weights. However, we believe it requires additional investigation to draw more precise conclusions.

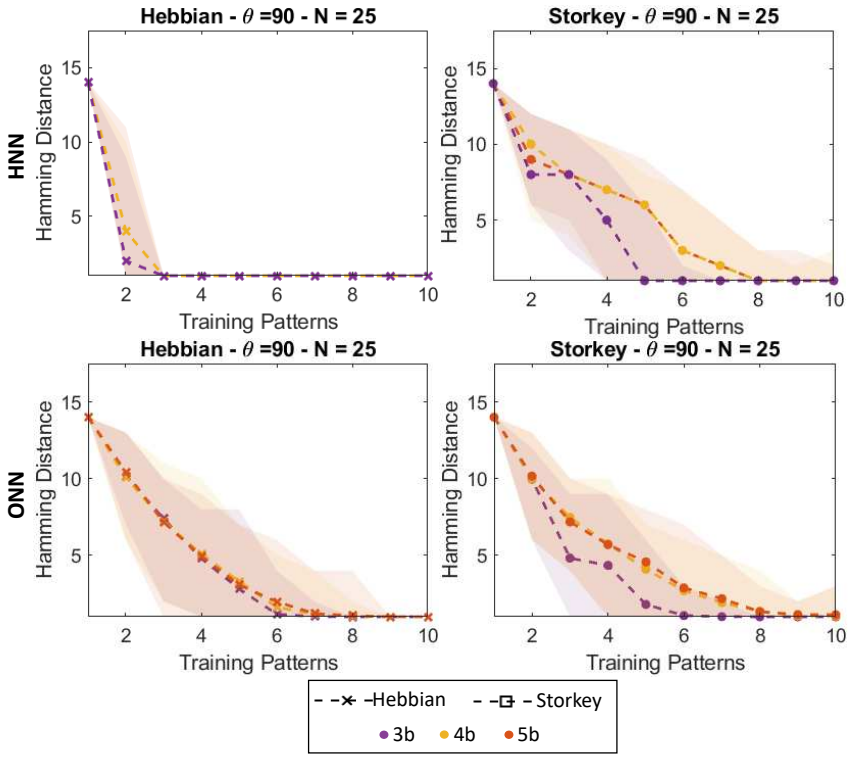


FIGURE 3.9 – *Capacity of 25-neuron HNN and OHN trained with Hebbian or Storkey.*

25-neuron OHN latency for various weight precision

We measure the latency of the 25-neuron OHN for on-chip learning and compare it with previous implementations of 15-neuron on-chip OHN, and off-chip OHN. The latency is divided into three parts, the OHN computation latency, the weight computation latency, and the transmission latency. The OHN computation latency is by default stable no matter the weights and size of the network, so we expect it to stay stable. The weight computation latency mainly depends on the learning rule and computation complexity of the learning rule. The transmission latency depends on the weight precision and the network size. Concerning inference,

Table 3.3 shows that OHN computation takes around 2 to 3 oscillation cycles to compute, similar to the 15-neuron on-chip OHN [196] and the off-chip OHN [170]. Then, the transmission of OHN input and output takes 27μs which is 1.5 times higher than the OHN computation. Note, increasing the OHN size will also increase the transmission latency, while the OHN computation should stay stable. Thus, the architecture increases the inference latency compared to smaller-scale on-chip OHN and off-chip OHNs because of information transmission from PS to PL, and reversely.

TABLE 3.3 – *Measurements of latency for OHN training and inference with OHN oscillation frequency $F_{onn} = 97.7KHz$ and PS clock frequency $F_{PS} = 667MHz$.*

25 neurons				15 neurons [196]
Training				
Weights	3 bits	4 bits	5 bits	5 bits
Hebbian learning	55 μs			33 μs
Storkey learning	210 μs			77 μs
Weight precision	140 μs			NA
Weight transmission	18 μs	71 μs	175 μs	86 μs
Total Hebbian	213 μs	266 μs	370 μs	119 μs
Total Storkey	368 μs	421 μs	525 μs	163 μs
Inference				
Input transmission	9 μs			NA
ONN computation	17 μs			NA
Output transmission	18 μs			NA
Total	44 μs			NA

Concerning training, we differentiate the latency into 3 steps, one to perform the training algorithm in PS, another to rescale weights to the corresponding weight precision, and finally to transfer weights from PS to the OHN in PL. Table 3.3 highlights that Storkey requires more computation time than Hebbian due to the more complex Storkey algorithm, see 3.1, and 3.5, and the sequential processing of PS. Then, weight transmission increases drastically with the increase of the weight precision and the number of neurons. Reducing the OHN size and weight precision has an important impact on reducing transmission latency because we use AXI4-Lite with 32-bit parallel transmission.

The on-chip learning architecture, for a network of 25 neurons, allows computing Hebbian in 55μs, and Storkey in 210μs. Additionally, to allow reducing weight precision to 3, 4, or 5 bits, additional treatment is necessary, taking 140μs. Then, transmission time depends on the weight precision taking between 18 μs and 175 μs. In total, training a fully connected ONN, configured for 5-bit signed synapses, with a novel training pattern takes 370μs with Hebbian and 525μs with Storkey. Thus, because Hebbian and Storkey have similar precision in the digital ONN design, it can be more interesting for a system with real-time constraints to implement Hebbian rather than Storkey.

25-neuron OHN energy consumption for various weight precision

We extract the estimated post-place and route power consumption of the digital on-chip OHN design on Vivado considering the xc7z020-1clg400c target, and we compare it with the digital off-chip OHN implementation [170] and with other fully-connected OHN implementations [141, 161, 183]. We compute the energy per neuron per oscillation to have a common baseline for comparison. Table 3.4 highlights that the digital on-chip OHN requires more energy

per oscillation than the digital off-chip OHN [170], certainly because of the additional LUTs resources necessary for the re-programmable synapses. Also, the digital on-chip OHN is in the same energy range as the analog OHN from [161] as they operate at a lower frequency than the other implementations [141, 183]. Using a higher OHN frequency reduces the oscillating time, ultimately reducing the energy per oscillation. The digital OHN frequency is currently limited by the FPGA. Note, [170] computes with a lower frequency but takes advantage of the large-scale network.

TABLE 3.4 – *Comparison of the digital ONN with re-programmable synapses with other fully-connected ONN implementations.*

	[141] 2018	[161] 2021	[183] 2023b	[170], 2021	This work
Neurons	100	30	16	140	25
Power	303 mW	1.76 mW	160 μW	18 mW	10 mW
Frequency	1 GHz	45 kHz	1 MHz	488 kHz	187.5 kHz
Energy/osc	0.3 pJ	1.3 nJ	10 pJ	65.3 pJ	2.13 nJ

3.3.5 Discussion of the OHN on-chip learning

In this section, we studied learning algorithms and provided an implementation to perform continual on-chip learning with the digital OHN for pattern recognition. It highlights that HNN unsupervised learning algorithms are compatible with OHN on-chip learning only if they are local and incremental, and if they produce a symmetric weight matrix without self-coupling. We identified two HNN learning rules compatible with OHN on-chip learning, Hebbian and Storkey. Both exhibit similar capacity results when performing on-chip learning on a 25-neuron OHN, making them both suitable for continual OHN on-chip learning.

The proposed architecture takes advantage of a Zynq processor equipped with both PS and PL resources to implement the fully connected digital OHN [170] with re-programmable synapses in PL and execute the learning algorithms in PS. First, it is important to highlight that the on-chip learning architecture does not require many changes from the digital OHN design, making it easy to adapt and install. The main limitation is the scalability of the architecture due to the re-programmable synapses demanding a large number of LUTs, even with reduced weight precision, limiting the OHN size up to 35 neurons. In comparison, the digital OHN without re-programmable synapses could reach around 140 neurons. Another limitation of the architecture scalability is the latency induced by the separation between OHN learning and computation in PS and PL. On one side, PS allows to implement and compute a large panel of unsupervised learning algorithms, executing them sequentially with a fast frequency of $F_{ps} = 666MHz$. On the other side, it generates latency to transmit the weights from PS to PL, increasing with the OHN size. An alternative solution is to implement the training algorithms using the parallel properties of PL resources to provide fast training and remove the transmission latency. However, we believe it would utilize additional PL resources, including LUTs, which are already limited. Another solution is to use other communication bus than AXI-Lite between PS and PL, such as AXI-stream which provides faster parallel transmission. Overall, the on-chip learning architecture can train a 25-neuron OHN in hundreds of microseconds, between 350μs and 550μs which is the first solution to perform OHN on-chip learning.

Next, in order to showcase the OHN on-chip learning architecture, we need to test it on realistic edge applications. The digital ONN design has already been used for sensor data treatment in various applications [170, 198], so we are confident in the integration of the on-chip

learning architecture with different sensor treatment applications. Possible applications could be in the robotics domain where real-time continual learning is often necessary, and where the digital OHN design already showcased good performances [198]. Using OHN on-chip learning would allow training the OHN continuously through time depending on the environmental configuration given by the sensory information. We believe that using the OHN on-chip learning architecture can be beneficial in the case of applications with large-scale inputs where all possible configurations can not be anticipated. We propose a solution to perform real-time OHN on-chip learning for an obstacle avoidance application in Chapter 4 [199].

3.4 Supervised learning for binary-OHN MNIST classification

We previously demonstrated the digital OHN performances to solve pattern recognition using unsupervised learning rules. Additionally, we introduced a system architecture to perform OHN on-chip learning using unsupervised learning rules. However, the capacity obtained with unsupervised learning rules to train OHN off-chip or on-chip for pattern recognition is limited. In comparison, usual ANN models taking care of image processing tasks are mainly multi-layer networks, like CNNs, trained with supervised back-propagation algorithms to perform image classification. Image classification is a primary computer vision task deployed in many industrial systems, such as healthcare or manufacturing systems.

Multiple benchmarks and datasets exist to evaluate models on image classification. The main ones include MNIST [9] [26], ImageNet [15], and CIFAR-10. MNIST contains grayscale 28x28 labeled images of handwritten digits, while ImageNet and CIFAR-10 classify objects using larger and more complex colored images. They are all used to a large extent for assessing AI-model performances. Even though image processing and pattern recognition are two different tasks, authors in [200] adapted HNN to solve a simplified MNIST classification task using the Storkey learning rule. They obtain 61.5% precision, while typically CNNs achieve around 99% on the standard MNIST classification task [1]. More recently, authors in [58] adapted the CHR [30] to perform supervised learning with energy-based RNN models, using the equilibrium propagation (EP) learning algorithm.

In this work, we study how to perform a simplified MNIST image classification using an OHN trained with EP. We start by testing the classification method developed by [200] on HNN to OHNs, meanwhile evaluating the HNN and OHN capacity performances with unsupervised learning rules. Then, we adapt EP to single-layer HNN and OHN networks, which are AAM networks, thus performing AAM-EP to train them for MNIST classification and study if it can improve capacity. We evaluate the precision on simplified MNIST test sets with HNN simulated with Matlab, and digital OHN design simulated on FPGA [170].

3.4.1 MNIST classification with OHN

Pattern recognition and image classification are two distinctive tasks. Thus, one needs to adapt pattern recognition, solved by AAM networks, to perform classification. In [200], authors propose a solution to apply HNN on MNIST classification using the Storkey learning rule. In this work, we replicate their method to evaluate and compare the precision of HNN and OHN on a simplified MNIST set for different unsupervised learning configurations. We first present the MNIST set and the simplified version we use in this work before describing methods to classify the simplified MNIST set using AAM models.

The MNIST set was created to assess neural networks' performances on image classification tasks. It contains 28x28 gray-scale labeled images of handwritten digits, from 0 to 9. It is organized in two sets, a training set with 60000 images, and a test set with 10000 images. State-of-the-art solves the MNIST classification problem with CNN models trained with supervised back-propagation [10] and can achieve more than 99% of accuracy. In this work, we employ a simplified MNIST set containing the same number of training and test images, where each image is pre-processed to be transformed into a 10x10 black and white image, see Figure 3.10. We focus on a 10x10 format because the digital OHN design is limited in size and resources. The transformation of each image follows three steps. First, each 28x28 image is cropped to 20x20 removing the black background to reduce similarities among images before being resized to a 10x10 image by taking average values of 2x2 neighbor pixels. Finally, we binarize each 10x10 image into black and white using a threshold. Later, we study different binarization thresholds and their influence on the simplified MNIST classification task.

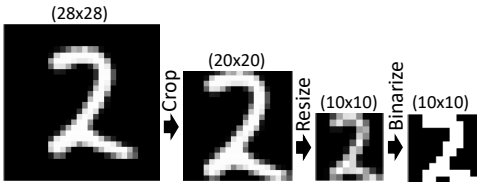


FIGURE 3.10 – *Simplified MNIST classification image pre-processing method.*

There are some clear distinctions between pattern recognition and classification tasks. On one hand classification problems associate input information with output classes, hence, input and output can have different dimensions. On the other hand, pattern recognition tasks associate a corrupted input with a clean memorized output, where both have the same dimensions. To solve the MNIST classification problem using AAM, authors in [200] propose to train an HNN network with one pattern per label, so one image per digit. Thus, inference will stabilize to one of the training patterns corresponding to a digit label class, equivalent to the MNIST classification task, see Figure 3.11.

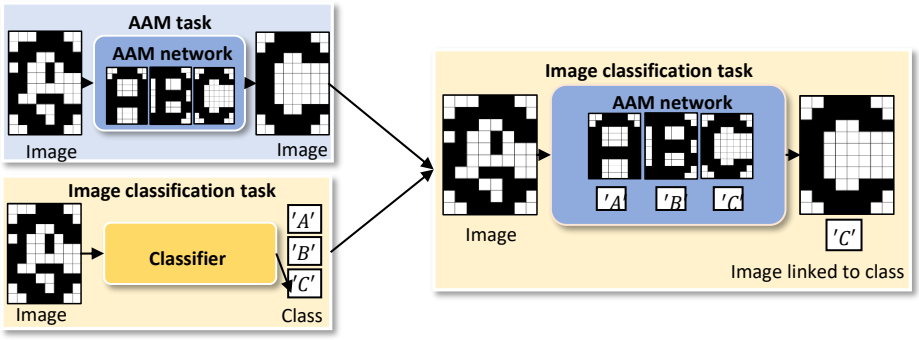


FIGURE 3.11 – *Difference between AAM and image classification tasks, and the adaptation of AAM network for image classification task.*

MNIST classification with AAM starts with the training step to configure the weights and the choice of the training patterns, which is key for high precision. Each training pattern must be the best representation of its digit, such that each image of that digit from the MNIST set will stabilize to that training pattern. Authors in [200] propose to perform a mean on each gray scale image with the same label from the MNIST training set of 60000 images, and we re-use the same method. We define 10 training patterns corresponding to the 10 digits which will be learned as stable points for both HNN and OHN networks. The 10 training patterns are created from the 60000 training images by grouping them by digits and computing a mean image for each digit such that we obtain 10 28x28 gray scale images representing digits between 0 and 9.

After, we apply the pre-processing on each training pattern to obtain ten 10x10 black and white images, with one image per digit being the training pattern associated with the corresponding digit, see Figure 3.12. The binarization threshold determines the number of black or white pixels in each image. In AAM tasks, having uncorrelated training patterns is also key for high precision [194, 178]. The more patterns are correlated, the harder it is to dissociate them. The correlation of patterns is evaluated by the Hamming distance (HD) metric d, which calculates the number of different pixel values between two patterns. The gray scale in MNIST images is encoded between 0 and 1. Thus, as in [28], we study the ideal threshold to have the largest HD between training patterns. Figure 3.13a shows the average HD d_{avg}, and the minimum HD d_{min} in between the 10 training patterns depending on the binarization threshold θ. It highlights that HD is maximal for θ=0.3, meaning training patterns are less correlated. Figure 3.13b depicts the HD between the 10 patterns generated after pre-processing with θ=0.3, and Figure 3.13c prints the resulting 10 training patterns.

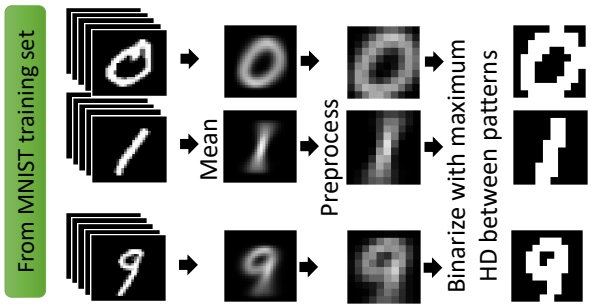
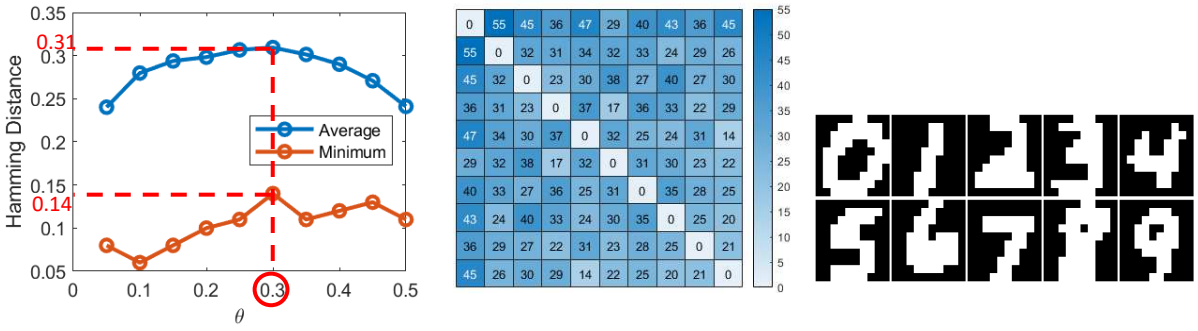


FIGURE 3.12 – *Definition of training patterns from MNIST training set. Pre-processing binarizes each training pattern to be converted into black and white.*



(a) *Average and mean training patterns HD for various θ.*

(b) *HD between training patterns binarized with θ=0.3.*

(c) *Training patterns with θ=0.3.*

FIGURE 3.13 – *Definition of MNIST training patterns.*

Once we have training patterns, we need to define synaptic weights with the learning algorithms. We first use unsupervised learning rules described in Section 3.2.1 to evaluate the performances of HNN and OHN for MNIST classification, before adapting the supervised EP algorithm, creating the AAM-EP. With supervised AAM-EP, we study first if it can train an AAM model from scratch, initializing with random weights, and then we study if it can improve the capacity of a pre-trained network, initializing with weights generated by unsupervised learning algorithms, see Figure 3.14. For each training solution, we start by studying the best HNN training configuration using Matlab simulation. Then, weights of the best HNN precision are normalized into a 5-bit signed representation to be compatible with digital OHN design. We simulate the digital OHN design using the Vivado design tool with xc7z020-1clg400c FPGA target.

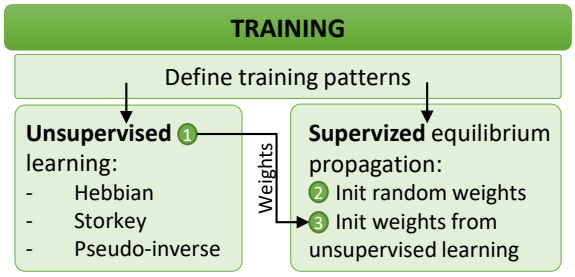


FIGURE 3.14 – *Study of training options to perform MNIST classification with HNN and OHN.*

We evaluate precision from inference results on the MNIST test set of 10000 images. Inference starts by initializing neurons with one of the test images. Then, the network evolves until stabilization. The stable state is then compared to the 10 training patterns to define the output class. An exact match between OHN output and the corresponding training pattern is considered a correct classification. MNIST results are evaluated through four metrics. First, we evaluate the precision representing the percentage of tested images that stabilize to the correct training pattern. Then, we compute the true negative metric, which counts the number of images which stabilize to one of the training patterns but not the expected one. We also add the percentage of spurious outputs incorporating images that stabilize to none of the training patterns but to another non-memorized image. Finally, for the ONN, an additional metric is necessary to highlight the percentage of images that never stabilize to an output. We call them inconsistent images. We report results obtained on the 10000 images from the simplified MNIST test set.

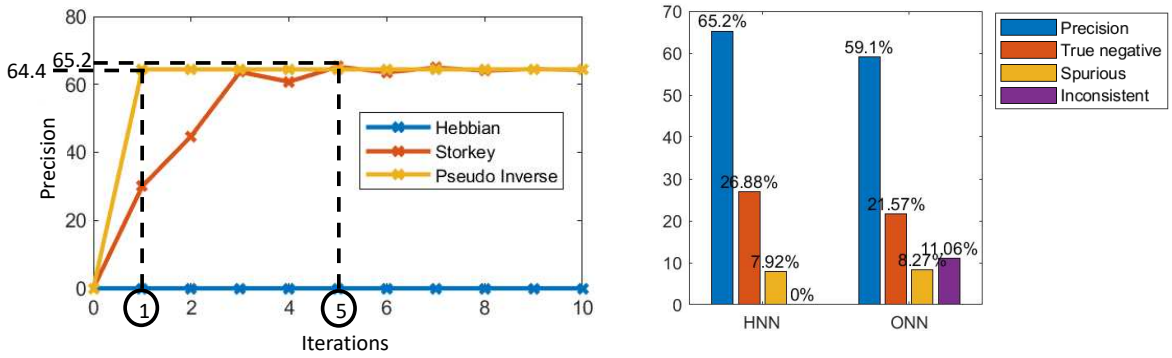
3.4.2 Unsupervised learning for MNIST Classification

We study three unsupervised training algorithms applied off-chip: Hebbian, Storkey, and Pseudo-inverse on HNN. Note, we study the impact of iterative learning on Storkey, Pseudo-inverse, and Hebbian. Then, we extract the weights giving the best HNN precision and integrate them inside the digital OHN to compare precision.

Figure 3.15a highlights the precision obtained using HNN for unsupervised training configurations. Configurations include the learning rules, Hebbian, Storkey, or pseudo-inverse for up to 10 iterations. Pseudo-inverse and Hebbian are not sensitive to iterative learning as precision does not change depending on the iterations, but Storkey is sensitive. Figure 3.15a also shows that the best precision, 65.2% is obtained using the iterative Storkey learning rule after 5 iterations. However, training HNN with Pseudo-inverse can reach a precision close to the best with 64.4% after a single iteration. We configure the digital OHN design using the synaptic weight values obtained with the best configuration, 5 Storkey iterations, to compare OHN precision with HNN precision on the simplified MNIST classification task.

Figure 3.15b shows HNN and OHN have similar trends. First, HNN precision and true negative percentages are higher than OHN. The difference is certainly due to the normalization of weights into 5-bit signed integers in the digital OHN design. Also, the number of spurious patterns detected with OHN is slightly higher than with HNN, and inconsistent patterns often happen with OHN while never with HNN. Thus, we believe that HNN decides more easily of a stable output, even if it is a true negative, while OHN can hesitate between different outputs and keep bouncing between patterns. Figure 3.15b also reports, to the best of our knowledge, the best precision obtained with AAM networks trained with unsupervised learning algorithms to solve a simplified MNIST classification task. We report on 65.2%

precision with HNN and 59.1% precision with OHN. However, the reported precision is lower than the state-of-the-art precision of neural network models solving MNIST classification problems with supervised learning, reaching around 99% accuracy. Hence, next, we propose a solution using AAM-EP to train HNN and OHN on our simplified MNIST set to investigate if supervised learning can increase capacity precision.



(a) *HNN precision for multiple unsupervised training configurations.*

(b) *Results of OHN and HNN trained with 5 iterations of the unsupervised Storkey learning rule.*

FIGURE 3.15 – *Results of HNN and OHN on simplified MNIST classification task trained with unsupervised learning rules.*

3.4.3 Supervised AAM-EP learning for MNIST Classification

The common algorithm to solve supervised problems with multi-layer RNN models is back-propagation through time (BPTT) [201, 202]. Even though it is efficient and gives really high precision, equilibrium propagation (EP) [58] was proposed as a supervised learning algorithm requiring less computation for energy-based RNN models, taking inspiration from the contrastive Hebbian rule (CHR) [189]. EP demonstrated efficiency in solving the MNIST classification task using multi-layer energy-based continuous RNNs. HNNs and OHNs are also energy-based RNNs, yet made with a single-layer architecture of non-continuous neurons. Thus, in this paper, we propose, the AAM-EP, an adaptation of EP for single-layer AAM networks.

EP computes the gradient of an objective function, similar to the HNN energy function, that propagates in the layers. This gradient back-propagation is transparent in the weight update algorithm, and the final weight update equation is intuitive and simple to apply. Note, authors in [203] showed that EP and BPTT have similar gradient updates in an RNN, so achieving similar precision. The EP algorithm defines two learning phases to update the weights. The first phase, called the free phase, clamps the input information to the input layer and waits until neurons of the following layers stabilize. Then, EP performs a second phase, called the weakly clamped phase, where input neurons and output neurons are clamped with the input information associated with the expected output information. In [58], they show that the signal back-propagated during the weakly clamped phase corresponds to the derivative error of their objective function, and they define the following weight update algorithm:

1. Clamp input and let the network evolve until all neurons from hidden and output layers stabilize.
2. Save the stable state ξ_i^0 of each neuron i.
3. Weakly clamp with β the expected output and let the network evolve until all neurons from hidden layers stabilize.
4. Save the new stable state ξ_i^β of each neuron i.

5. Update weight w_{ij} between neuron i and neuron j as:

$$w_{ij} = w_{ij} + \frac{1}{\beta}(\xi_i^\beta \xi_j^\beta - \xi_i^0 \xi_j^0) \tag{3.6}$$

Using β=1, in [58], authors show that RNNs with 1, 2, or 3 hidden layers can solve the MNIST classification problem and reach more than 95% of precision. Eventually, EP is a low-computation, energy-based learning algorithm expected to fit with analog computing paradigms. Thus, it makes it attractive for ONN applications. However, EP can not be applied directly to HNN or OHN as they are single-layer energy-based models. Thus, we adapt EP into AAM-EP for HNN and OHN, considering the previous training patterns defined for unsupervised learning as the corresponding digit labels for each image from the training and test set, see Figure 3.13c. AAM-EP uses these new input/output pairs from the MNIST training set to clamp input and output during the two training phases.

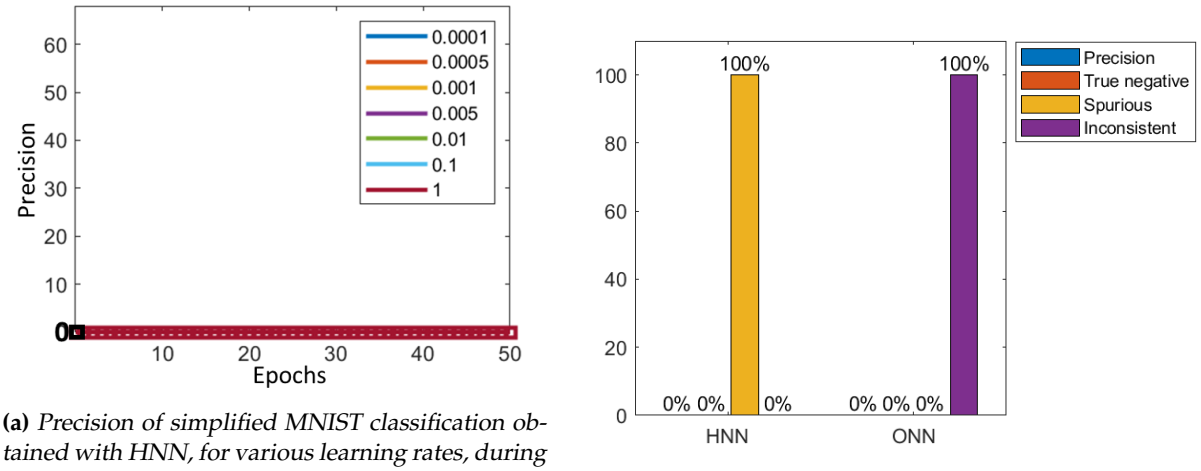
EP learning is a two-phase algorithm, with a first free phase with input neurons clamped, and a second weakly clamped phase with additional weakly clamped output with factor. The clamping principle works well for a multi-layer network with at least one hidden layer [58]. However, in HNN or OHN cases, input and output neurons are the same, and there are no hidden layers. Thus, we propose the AAM-EP algorithm as follows:

1. Clamp the input image in the network; Consider ξ_i^0 as activation of neuron i from the input image.
2. Clamp the expected output image with β=1 in the network; Consider ξ_i^β as activation of neuron i from the expected output image.
3. Use the two activation states to update the weight between neuron i and j as:

$$w_{ij} = W_{ij} + \alpha(\xi_i^\beta \xi_j^\beta - \xi_i^0 \xi_j^0) \tag{3.7}$$

Note, that we remove the factor $1/\beta$ because we use β=1, and we add a learning rate factor α in order to regulate the weight update for each training iteration (each image). Also note, that recently, authors in [204] also proposed to use EP for pattern recognition with ONN. In [204], authors apply supplementary neurons to clamp training patterns during the clamping phase, while in this work we do not require additional neurons. Also, in [204], authors perform phase dynamics simulations of memristor-based OHN to validate their method, while here we perform simulations of a digital OHN design.

During tests, we consider various ranges of learning rates between 0.0001 and 1. Initialization of the weights is also important to achieve high precision. In this work, we first initialize weights randomly, with small values between $[-1; 1]$ to study if AAM-EP can train an AAM network from scratch. After, we initialize networks using weights computed previously with unsupervised learning to study if AAM-EP can improve the precision of an already trained network. At first, we apply AAM-EP for numerous epochs and observe the HNN precision at each epoch for various learning rates. Each epoch applies a random minibatch of 1000 pre-processed images from the MNIST training set to update the weights. Precision can slightly change from one trial to another as minibatch images are randomly chosen from the full simplified 10x10 MNIST training set. The precision for each epoch is first computed on the full simplified 10x10 MNIST test set considering HNN on Matlab. In the second step, we select the best configuration and collect corresponding weights to evaluate the AAM-EP learning algorithm with the digital OHN.



(a) *Precision of simplified MNIST classification obtained with HNN, for various learning rates, during numerous epochs, starting with weights initialized randomly.*

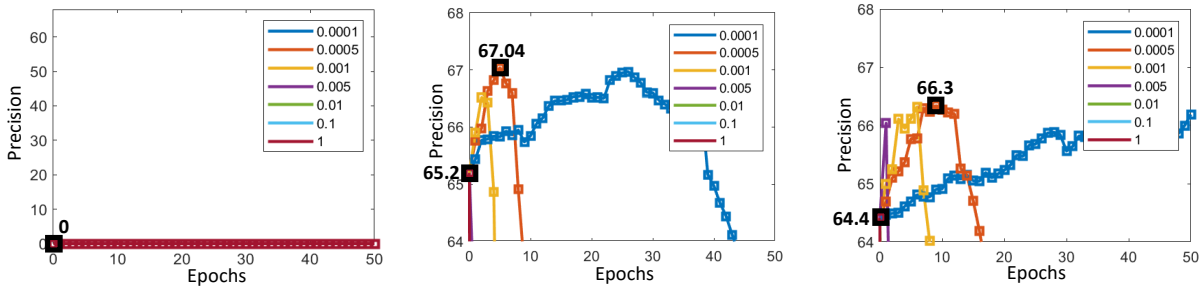
(b) *Results of ONN and HNN trained with AAM-EP algorithm with random weight initialization after 10 epochs.*

FIGURE 3.16

To begin with, Figure 3.16a displays HNN precision evolution for multiple learning rates during 50 epochs, with weights initialized with small random positive values. It shows that for all the tested learning rates, during several epochs, the precision stays 0%. Moreover, Figure 3.16b compares results between HNN and OHN, obtained using weights achieved with learning rate $\alpha = 0.0005$ after 10 epochs. Note, that we tried various learning rates and epochs and obtained the same results. It highlights that for HNN, each image stabilizes to a spurious pattern, while for OHN, neuron states continuously evolve without reaching stabilization. To sum up, using AAM-EP learning from scratch with HNN or OHN, does not result in high precision on the simplified MNIST classification task. Subsequently, we reproduce the precision tests with weights initialized using unsupervised learning. More precisely, we initialize weights with the best learning configuration obtained for each unsupervised learning algorithm: weights generated with Hebbian after one iteration, with Storkey after 5 iterations, and with Pseudo-inverse after one iteration. Figure 3.17 shows the precision of the HNN trained with AAM-EP during 50 epochs for various learning rates when weights are initialized with Hebbian, pseudo-inverse, and 5 iterations of Storkey. It highlights that for weights initialized with Hebbian, the AAM-EP does not modify the network to allow the classification of the simplified MNIST task. It also illustrates, for Storkey and pseudo-inverse, a particular behavior in which precision increases during the first couple of epochs and decreases afterward. The larger the learning rate is, the faster the increase and decrease phenomenon is observed. Considering pseudo-inverse initialization, the maximum precision is obtained with learning rate $\alpha = 0.0005$, after 9 epochs, for which HNN reaches 66.3% precision. Considering Storkey initialization, the maximum precision is obtained with learning rate $\alpha = 0.0005$, after 5 epochs, and HNN reaches 67.04% precision. For both pseudo-inverse and Storkey initialization, AAM-EP increases the HNN precision by around 2%. We assume this phenomenon is due to the weight's initial values. If the unsupervised learning already set weights to an acceptable network configuration, then the AAM-EP algorithm can slightly help to increase precision up to a certain point after which it modifies the previous configuration and reduces drastically the HNN precision. However, if the weights initialization does not bring the network to an acceptable configuration, such as with Hebbian or random weights initialization, the AAM-EP can not modify enough the network to reach a good configuration.

We configure the digital OHN with the best HNN configuration, that is weights obtained after 5 epochs of AAM-EP with learning rate $\alpha = 0.0005$ with Storkey initialization. Figure 3.18 plots the precision obtained with both HNN and OHN. Note that obtained precision from

Figure 3.17b and Figure 3.18 are different. As we use a mini-batch of 1000 randomly chosen images at each epoch, from one run to another, computed weights and obtained precision can be slightly different. Figure 3.18 shows that for both HNN and OHN, precision increases with the use of the AAM-EP supervised algorithm. Additionally, the number of true negatives stays approximately stable for HNN and decreases for OHN, and the number of spurious patterns decreases. Thus, we deduce the AAM-EP algorithm helps to differentiate and reinforce training patterns such that input patterns are better associated with training patterns. For example, spurious patterns often appear to be close to training patterns with some wrong pixels. We believe AAM-EP helps to modify local weights associated with wrong pixels such that HNN and OHN stabilize to correct training patterns. HNN and OHN precision increase of about 2%, and 3.5%, respectively. Table 3.5 assembles the best HNN and OHN precision of the three learning configurations, unsupervised learning only, supervised learning only, and both unsupervised and supervised learning.



(a) *1 iteration of Hebbian.* **(b)** *5 iterations of Storkey.* **(c)** *1 iteration of Pseudo-inverse.*

FIGURE 3.17 – *Simplified MNIST classification precisions obtained with HNN, for various learning rates, during numerous epochs, starting with weights initialized with unsupervised learning rules.*

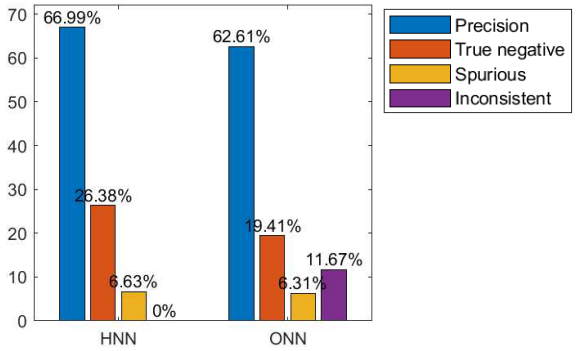


FIGURE 3.18 – *Results of OHN and HNN trained with AAM-EP algorithm with α=0.0005 after 5 epochs with weights initialized after 5 iterations of Storkey.*

TABLE 3.5 – *Best precision results obtained with both HNN Matlab emulator and OHN digital design for the three training configurations.*

	Storkey 5 iterations	EP All α/epochs	EP and Storkey α = 0.0005, epochs = 5
HNN	65.2%	0%	67.04%
OHN	59.1%	0%	62.61%

3.4.4 Benchmarking and discussion

In this section, we proposed a supervised learning solution for AAM networks with the AAM-EP. More than that, we highlighted that the supervised AAM-EP learning algorithm can

help to increase the precision of HNN and OHN networks pre-trained with unsupervised learning algorithms for the MNIST classification task. However, using supervised learning can be more demanding in terms of computational efforts.

In this work, training was performed in Matlab. However, EP was presented as a hardware-aware learning algorithm, and an extension of this work could explore the integration of AAM-EP in digital or in analog designs. We derive from results obtained in Matlab the computational efforts for the different learning algorithms, as well as an estimation of the learning latency if learning was implemented on the digital design. We evaluate the computational efforts using the metric of the number of multiply and accumulate operations (NMAC$_{OP}$) required for each learning method. Table 3.6 shows a comparison of the computational efforts for the various learning methods for a general case, as well as for the MNIST classification application. It shows that the AAM-EP increases drastically the NMAC$_{OP}$ per training compared to the unsupervised learning algorithms. Also, considering a system frequency of F_{sys}=31.25 MHz, and parallelism in the $NMAC_{OP}$, Hebbian learning can compute in 1 clock cycle in $32ns$, Storkey in $480ns$, and Pseudo-inverse in $96ns$, while AAM-EP requires approximately 50 ms to train HNN or OHN for MNIST classification task. Thus, supervised AAM-EP takes longer to compute than other unsupervised learning algorithms. However, depending on the application, using AAM-EP to increase precision while also increasing computational efforts and latency can be interesting.

TABLE 3.6 – *Computational efforts required for training depending on the learning algorithm for a network of N neurons, for k training patterns, after it iterations or ϵ epochs. $\#_{cycles}$ represents the number of cycles necessary for the OHN inference, we consider 3 cycles on average and a frequency F_{sys}=31.25MHz.*

Learning rules	$NMAC_{OP}$	$NMAC_{OP}$ MNIST	Latency MNIST
Hebbian	$(kN^2)it$	100K	32 ns
Storkey	$(kN^2 + 2kN)it$	510K	480 ns
Pseudo-inverse	$((kN^2) + 2(N^2k))it$	300K	96 ns
AAM-EP	$\#_{img}\epsilon(\#_{cycles}N^2 + 2N^2)$	5000K	50 ms

We report an HNN maximum precision of 67% and an OHN maximum precision of 62.5%. In comparison, [200] reported 61.5% HNN precision, while performing classification of a simplified MNIST set with 14x14 black and white images, with additional pattern optimization. So, on one hand, the AAM-EP has, to the best of our knowledge, the highest reported precision of single-layer AAM networks performing classification of handwritten digits from MNIST set. And on the other hand, state-of-the-art multi-layer RNNs trained with EP can solve the complete MNIST classification problem with more than 90% precision, see Table 3.7. However, multi-layer models are often heavy, requiring a lot of resources and latency to be trained. We also compare HNN and OHN with state-of-the-art ANN models in terms of resources and latency by comparing the number of parameters to tune the network, as well as the number of epochs necessary to obtain the best precision. Table 3.7 highlights that HNN and OHN are single-layer models requiring a low number of parameters compared with multi-layer models. In terms of training latency, OHN and HNN, reach their maximum precision after less than 10 epochs, while most of the multi-layer networks necessitate more epochs to converge to their maximum precision. Thus, ONN can be of interest for applications with a restricted amount of resources, but allowing approximate precision.

In this section, we saw that our proposed AAM-EP learning algorithm improves the already known precision of AAM on the handwritten digits classification task but does not overtake multi-layer RNN model precision on the same task. The main difference between AAM networks and usual CNN models is the architecture. AAMs are single-layer fully-connected net-

works, while CNNs are multi-layer models. Thus, a possible improvement is to explore multi-layer associative memory architectures [205], like heterogeneous associative memory (HAM), in order to have a more coherent architecture with classification tasks [206]. HAM networks using perceptron neurons, such as the Linear Associative Memory [207], or the Bidirectional Associative Memory [208, 209] could replace the single-layer HNN or OHN. In particular, the recent work from Rudner explored multi-layer ONN for simplified MNIST pattern recognition trained with back-propagation through time and achieved more than 70% precision [202], encouraging further investigation for multi-layer ONNs. In particular for low-cost learning solutions, avoiding BPTT. For example, the EP algorithm was first introduced as a supervised learning algorithm for energy-based multi-layer RNNs, thus we also expect it to be compatible with multi-layer HNN and ONN. Also, there are other hardware-aware learning algorithms that are compatible with multi-layer networks, which could help increase HNN and ONN precision [210].

Another important difference between HNN, OHN, and conventional ANN multi-layer models is the state precision. Current HNN and OHN limit pattern recognition to binary patterns. However, models achieving high classification precision are mainly continuous models. Thus, we believe exploring multi-state pattern recognition with OHN could help achieve higher capacity precision.

TABLE 3.7 – *Comparison of various network models trained with EP on the MNIST classification task.*

	HNN OHN	RNN (784-H-10) H = 500 [58]			BNN Conv. Output (O) [65]		A. RNN (784-H-10) [211]	SNN (784-H-10) [91]
Arch.	100-FC	1H	2H	3H	O=10	O=700	H=100	H=500
Param.	10k	397k	647k	897k	33.76k	>2M	79.4k	397k
Epochs	<10	15-20	40	140	30	10-20	10	5-10
Precision	67%	97%	98%	97%	11%	98%	96.5%	97%

3.5 Discussion and conclusion

In this chapter, we explored OHN learning algorithms for pattern recognition. Learning is a key aspect of obtaining high accuracy with a particular network model applied to a specific task. Here, we focus on the fully connected OHN model to solve pattern recognition or AAM tasks. More than that, we test the OHN model based on the digital design from Chapter 2.

A state-of-the-art network to solve AAM tasks is HNN which is trained with unsupervised learning rules. Thus, as a first step, we explored how to adapt unsupervised learning rules introduced for HNN to OHN in order to analyze OHN accuracy and capacity on the pattern recognition task compared to HNN. We adapted the learning rules from [178] to fit the OHN constraints by imposing the symmetry and the 0-diagonal in the weight matrix. We tested various learning rules on a 10x6 OHN trained for digit recognition and compared the results of the different learning rules on the digital OHN design using synapses with a limited weight precision and on an HNN on Matlab with full-precision synapses. HNN and OHN obtain similar accuracy on the same task with a small difference certainly coming from the synaptic weight precision. The difference mainly reduces the OHN accuracy and creates additional OHN inconsistencies. In the first experiments, we validated the OHN for pattern recognition compared with HNN, using various unsupervised learning rules, no matter their characteristics, local or increment, but respecting the symmetry and 0-diagonal in the weight matrix.

However, non-local and non-incremental learning rules are not compatible with continual on-chip learning. Continual learning is necessary for some applications where the system needs to adapt to a changing environment, such as reinforcement learning in robotics. Thus, in this chapter, we also proposed a system to perform continual on-chip learning with the digital OHN on FPGA using the processing system of a Zynq processor. We applied the local and incremental Hebbian and Storkey learning rules to it respecting the symmetric and 0-diagonal weight matrix. We validated the on-chip learning system on pattern recognition and highlighted the fast OHN training and computation time. However, we also pointed out the scalability limitation to 35 fully-connected neurons due to the important digital resource utilization of the re-programmable synapses.

In both cases, using unsupervised learning rules to train an OHN off-chip or on-chip resulted in limited accuracy on the pattern recognition task. Even if HNN is state-of-the-art for this kind of application, it is not the best solution. Thus, as OHN and HNN are close in architecture, if they use equal learning rules they obtain similar accuracy. In comparison, AI models are usually trained with supervised back-propagation, and obtain higher accuracy on more complex tasks. Thus, in order to improve OHN accuracy in solving pattern recognition tasks, we explored a novel learning solution with the supervised equilibrium propagation (EP) algorithm. We adapted EP to single-layer HNN and OHN, building AAM-EP, to solve a simplified 10x10 MNIST classification task. We computed AAM-EP in Matlab based on the HNN model before applying weights to the digital OHN. We pointed out the computational efforts required to perform AAM-EP for the simplified MNIST set. It is 10 times higher than for other unsupervised learning algorithms, while lower, requiring fewer parameters and epochs than for multi-layer models. We also highlighted the AAM-EP algorithm can increase OHN accuracy from usual unsupervised learning algorithms by 3%, up to 62.5% which is, to the best of our knowledge the best-reported accuracy for OHN on a simplified MNIST set. However, multi-layer models trained with EP achieve more than 90% accuracy on the complete MNIST set, and the recent multi-layer ONN trained with BPTT achieved more than 70% accuracy [202]. Thus, AAM-EP is more advantageous in terms of computational efforts than more complex ANN models but is still limited to solving complex tasks efficiently.

In this chapter, we studied how to efficiently learn with OHN, from unsupervised to supervised learning algorithms, to assess OHN performances on pattern recognition. In both unsupervised and supervised learning cases, OHN does not achieve high accuracy compared to more conventional ANN models. We believe the main limitations are 1) the binary neuron activation that can limit the final accuracy of the system, and 2) the single-layer OHN architecture. Thus, in the next chapters, we explore novel ONN architectures to improve the scalability and efficiency of the ONN computing paradigm. More than that, we investigate how to apply ONN with novel architectures to realistic edge applications.

The work presented in this chapter resulted in two journal publications [197, 212] and one conference paper [196].

CHAPTER 4

CASCADED OHNS FOR MULTI-LEVEL AUTO-ASSOCIATION

Contents

4.1 Introduction

In the previous chapters, we introduced the digital ONN implementation on FPGA, configuring the ONN with fully-connected architecture, as an OHN for pattern recognition. We validated the design, tested it, and demonstrated its performance on a digit recognition application using a camera stream. We also emphasized the limitations of the current digital OHN, with first the scalability limitation due to the OHN architecture including significant synaptic elements, and inducing large resource utilization [170]. Then, we pointed out the capacity limitation with the Hebbian learning rule and studied alternative unsupervised learning rules to improve OHN accuracy for pattern recognition, implemented off-chip and on-chip [196, 197], and supervised learning for image classification [212]. However, the current OHN architecture combined with the different learning algorithms does not overcome the accuracy of conventional ANN models on simple benchmark data sets such as MNIST. Furthermore, the OHN architecture limits ONN for pattern recognition applications. However, ONN physical computation aims to solve more complex AI tasks on edge devices. Even if we were able to apply OHN for image classification by transforming the problem into a pattern recognition task, OHN might not be compatible with other AI tasks. We believe the main limitations of the OHN come from its fully-connected architecture, limiting the OHN size, learning performances, and applications. Thus, there is a need to explore novel architectures for ONN, beyond OHN.

Conventional ANN models are usually built with multi-layer architectures to achieve high accuracy [14]. Thus, we take inspiration from multi-layer ANN models to create layered ONN architectures. There are various ways to create ONN layers. Some can consider connected layers without internal-layer connections, such as classical layered ANN models. However,

such architecture requires an important adaptation of the digital design, as well as a study for possible learning solutions. As a first step, in this chapter, we propose an alternative layered architecture considering cascaded OHNs. In this case, we consider each layer as an OHN, and we send information from one layer to the other feed-forwardly. We show the cascaded OHN architecture is able to solve robotics and image processing applications.

In this chapter, we first detail the cascaded OHN architecture, how to perform inference and learning with it, and how to implement it digitally. Then, we demonstrate the digital cascaded OHN architecture for robotic applications, with an obstacle avoidance (OAV) task. Robots are often equipped with multiple sensors with real-time treatment constraints, where edge computing is attractive. Here, we propose a solution to perform real-time OAV on various mobile robots equipped with proximity sensors. More than that, we demonstrate the real-time OAV application with cascaded OHNs trained on-chip from its environment. The feed-forward connection is quite simple to implement in the digital design, while it can be challenging with analog designs. Thus, at last, we propose a solution, considering analog OHNs with analog feed-forward majority gates, creating cascaded OHNs in analog for an image edge detection application.

4.2 Cascaded digital OHN inference and learning

As a reminder, the OHN inference starts with the initialization of all oscillators with input phases. Then, oscillators interact with each other in parallel thanks to the coupling and the physical synchronization effect of coupled oscillators, and phases evolve through time until stabilization. After stabilization, the oscillators' phases represent the OHN output. In the case of cascaded OHNs, we consider two OHNs performing inference one after the other, such that the second OHN can use output information from the first OHN. Thus, inference starts with the inference of the first OHN until stabilization. Then, the oscillators' output phase information of the first OHN is used as input phases of the second OHN which can start its inference process, see Figure 4.1. Note, it is possible to have cascaded OHNs with equal size, transferring each neuron output of one OHN to the corresponding input neuron of the next OHN, or to have cascaded OHNs with different sizes, thus transferring only part of the neuron outputs as input for neurons of the next OHN, as shown in Figure 4.1. Also note, that it is possible to cascade more than 2 OHNs with the same principle.

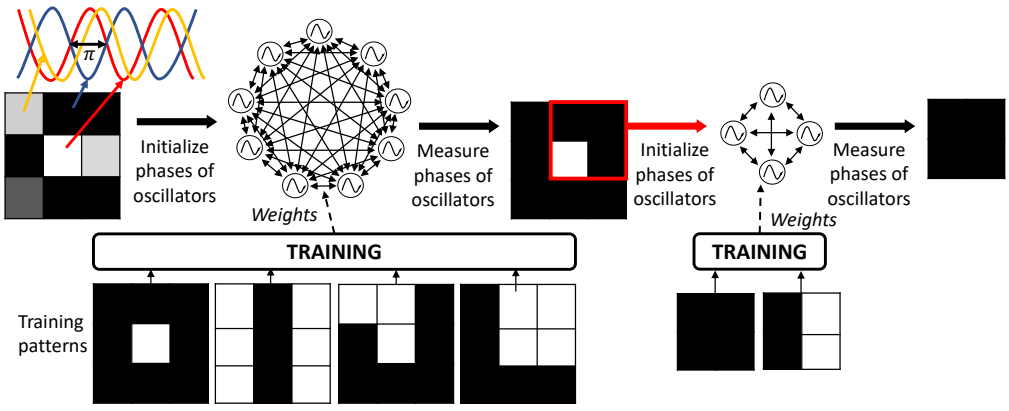


FIGURE 4.1 – *OHN computing with 2 cascaded OHNs.*

To implement the cascaded OHN architecture inside the FPGA, we reuse the digital OHN design previously described in Chapter 2. Instead of implementing one OHN, we implement

multiple OHNs in digital and control them one after the other. For example, considering two cascaded OHNs, we implement both networks digitally and control them starting by the inference of the first OHN and waiting until its stabilization to use the first OHN output, partially or completely, as input of the second OHN. Then, we start the inference of the second OHN and use its output as the output of the full cascaded-OHN design, see Figure 4.2. The ONN controller takes care of the synchronization of the two OHNs as well as the feed-forward propagation of the phase information between the first OHN and the second OHN inferences in the digital design by controlling the inputs and monitoring the outputs of each OHN.

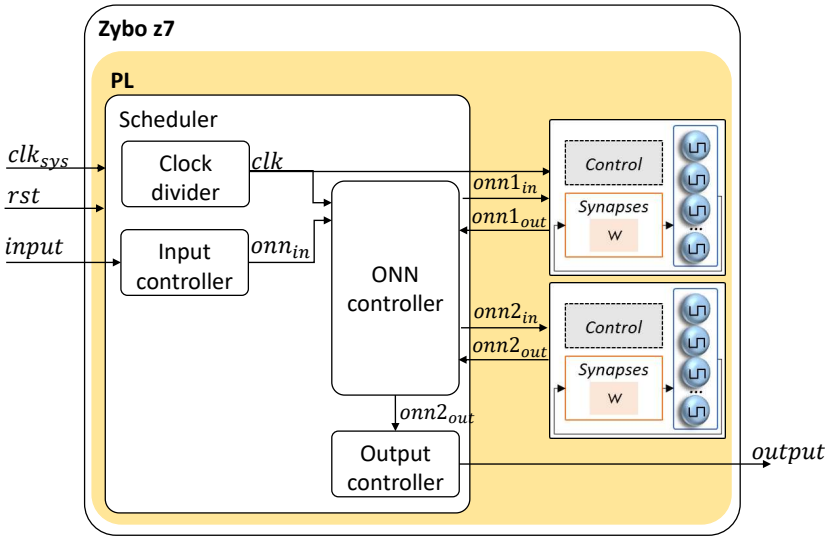


FIGURE 4.2 – *Digital implementation and control of 2 cascaded OHNs inside the FPGA.*

We consider learning of a cascaded OHN design as the configuration of multiple OHNs for pattern recognition. We use learning algorithms for auto-associative memory previously described in Chapter 3 to train each OHN individually. Before training, it is necessary to anticipate the role of each OHN in order to define the corresponding training patterns. Training patterns and the number of cascaded OHNs depend on the final application. In the next section, we demonstrate the cascaded digital OHN architecture for sensor treatment in robotics, with the obstacle avoidance (OAV) application. Then, we modify the OAV system to create an all-in-one system on chip (SoC) capable of reading, conditioning, and treating sensory data. After, we use the OAV on SoC to study real-time on-chip learning in a real edge application meanwhile learning from the environment. Finally, we propose an analog solution to cascade two analog OHNs using an analog feed-forward majority gate and apply it to the image edge detection application.

4.3 Cascaded OHNs for obstacle avoidance (OAV)

We apply the cascaded OHNs architecture to a robotic application, performing OAV on a mobile robot. Robotics is one of the domains where edge computing is widely deployed, and fast and low-power consumption are important constraints, because robots are equipped with many sensors, are often battery-dependent, and need to react instantaneously. Navigation is one of the main challenges for robots to move safely in their environment, and to navigate safely, robots also need to avoid obstacles in real-time [213]. Here we focus only on the OAV task. We consider the OAV task as the capacity to detect obstacles from sensory data and to react by avoiding the detected obstacles in real time. We choose OAV for its simple adaptation to pattern recognition tasks, and we show that computing with cascaded OHNs is a compatible

and attractive approach for robotic edge applications. We first consider an open-loop system with sensory data as input and direction decisions as output and implement it in the industrial robot E4 developed by A.I.Mergence [198].

4.3.1 E4 robot

E4 robot is a surveillance robot developed to ensure office security. It detects human intrusions and environmental risks, such as water, fire, and smoke.

E4 robot is a 3-side triangle robot that is 35 cm high, with a 36 cm front side and two 34 cm sides, see Figure 4.3. The robot can navigate smoothly in the environment, avoiding obstacles in order to detect anomalies, like water floods or intrusions. Additionally, the robot can interact non-verbally with humans, using sound and luminosity interactions. Available functionalities are:

- 360^o wide human detection
- Sound detection and classification
- 3D environment mapping
- Environment navigation
- Obstacle avoidance
- Flood water detection
- Remote control
- Non-verbal human interaction

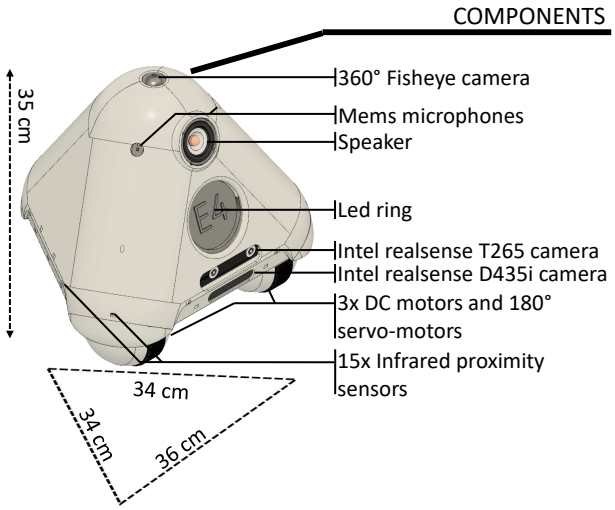


FIGURE 4.3 – *E4 robot functionalities and components.*

All these functionalities are enabled by multiple sensors and cameras placed all around the robot, see Figure 4.3. The robot analyses the surroundings, sounds, and images through sensors, microphones, and smart cameras to detect intrusions. Smart cameras are used to map the environment and ensure good navigation. Fifteen additional time of flight (ToF) proximity sensors placed all around the robot ensure correct navigation by detecting obstacles and hollows while avoiding them. Finally, the robot can communicate with the outside world and alert users of an anomaly via a speaker and an LED ring.

The robot's global architecture is shown in Figure 4.4. The interface board drives all peripherals, like sensors and cameras, through a CAN communication bus protocol. In addition, specific boards are used for each peripheral. For example, to control LEDs, an additional LED

control board is used between the interface board through the CAN bus and the LEDs themselves. In this work, we integrate cascaded OHNs implemented on an FPGA board inside the E4 robot to perform the OAV application. Thus, we modify the global architecture by connecting an FPGA board to the OAV board to compute the OAV application with cascaded OHNs, see Figure 4.4.

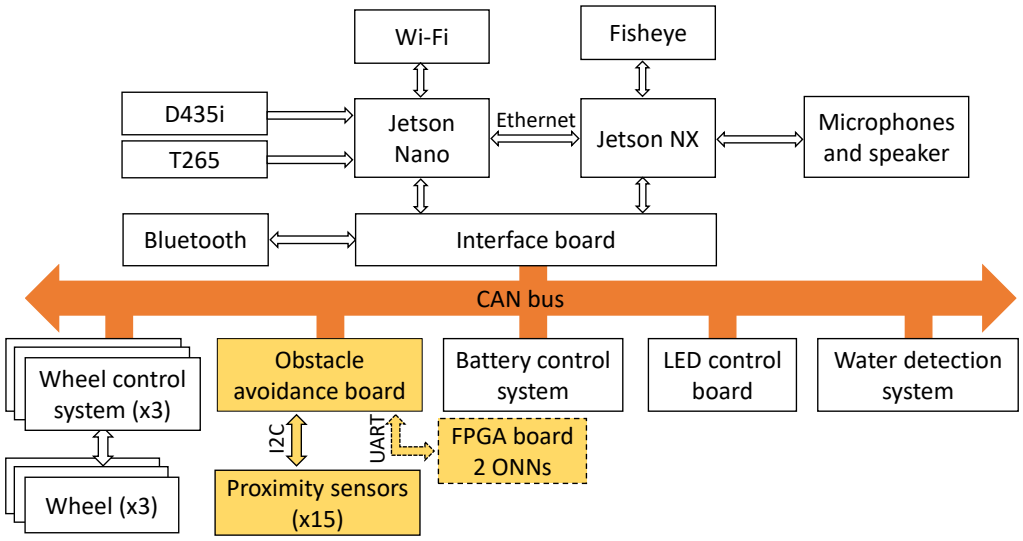


FIGURE 4.4 – *E4 robot system architecture.*

4.3.2 Default OAV process on E4

OAV application on E4 uses 15 ToF proximity sensors connected to the OAV board. The 15 ToF sensors are referenced as VL53L0X and can give distances from $0m$, up to $2m$. They are positioned all around the robot, as shown in Figure 4.5. Nine sensors are directed horizontally to detect front objects, and the six others are directed to the ground to detect small ground obstacles and hollows.

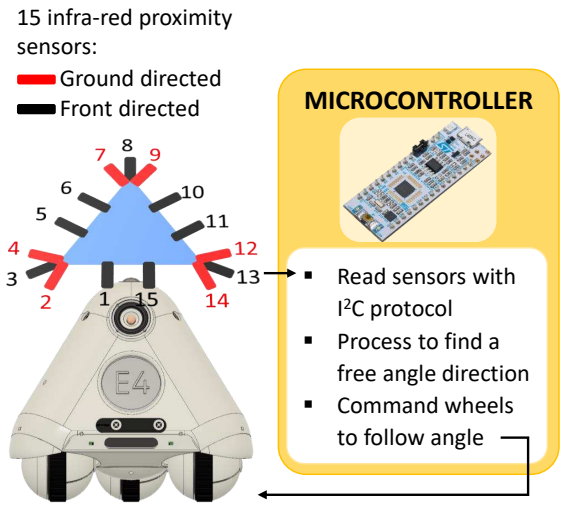


FIGURE 4.5 – *E4 robot components for OAV application.*

OAV board is a Nucleo board equipped with an STM32 microcontroller. The communication between the OAV board and the proximity sensors is defined by the sensor's manufacturer

as the inter-integrated circuit (I^2C) protocol. The I^2C protocol is a serial communication protocol often used in embedded electronics between the microcontroller and sensors. Only two wires are necessary for communication, a clock wire to synchronize both sides of the communication and a data wire to transmit data. The OAV algorithm inside the OAV board first reads the distance measured by each fifteen sensor values using the I^2C protocol. Then, it uses a Braitenberg algorithm [214] to detect if there is an obstacle or a hollow in front of each sensor. From this information, the OAV algorithm deduces directions that are free of obstacles. Finally, one of the available directions is chosen and sent to the wheel control system to avoid obstacles. In this original OAV algorithm version, no information from the previous robot direction is used to define the final direction, however, it can be included to anticipate future directions and improve the robot moving flow.

4.3.3 Cascaded OHN for OAV on E4

OAV application performs two main functions: 1) detect the obstacles, and 2) avoid them by finding a free direction. We propose to utilize an OHN for each of these functions, thus in total, two cascaded OHNs to perform the OAV application. The first OHN detects obstacles based on the information gathered from ToF proximity sensors, while the second OHN determines available angle directions from the detected obstacles.

The ToF proximity sensors measure distances between $0m$ and $2m$ and use the I^2C communication protocol to transmit measured distances to the OAV microcontroller board. We use a thermometer encoding technique to encode each sensor value into a 5x1 column image. The thermometer encoding is adjusted depending on the orientation of the sensor, see Figure 4.6. If the sensor is horizontally directed, we directly encode the measured value in a 6-state column (from 0 to 5). However, if the sensor is ground-directed, we define a default ground value and encode the difference between the measured and the ground value. Based on these encoded data, we create a 5x9 image by considering one front-directed sensor for each column of the image. Also, we add the ground-directed sensor values from sensors close to each front-directed sensor, see Figure 4.6. We fix the maximum value of the sum of sensors to 5 as each column only has 5 pixels. If the sum is greater than 5, we round it to 5. This method allows us to create areas for each front-directed sensor that takes into account values from both front- and ground-directed sensors. Finally, we obtain an image representing a 360^o map of the environment around the robot. Each column represents an area around the robot, and the number of black pixels in each column corresponds to distance with possible obstacles for each area, see Figure 4.6. This sensory data encoding is done on the OAV micro-controller board, and the final map image is then sent to the FPGA. It allows us to represent the all-around environment of the robot in terms of obstacles and hollows.

As aforementioned, the first OHN detects obstacles and hollows, while the second OHN defines the free directions (angles) that the robot can take. The first OHN gives as output the black pixels on each column depending on the proximity of the obstacle or hollow. To do so, we train the first OHN with 512 images corresponding to all possible combinations of full-column images representing a wide range of obstacles and hollows. For example, if an obstacle is close enough to be avoided, the OHN will output a black column in the corresponding area of the obstacle. However, if there is no obstacle or if the obstacle is far enough to be avoided later, the OHN will output a white column in the corresponding area. So, OHN identifies if there are obstacles or hollows in the different defined areas, see Figure 4.7a.

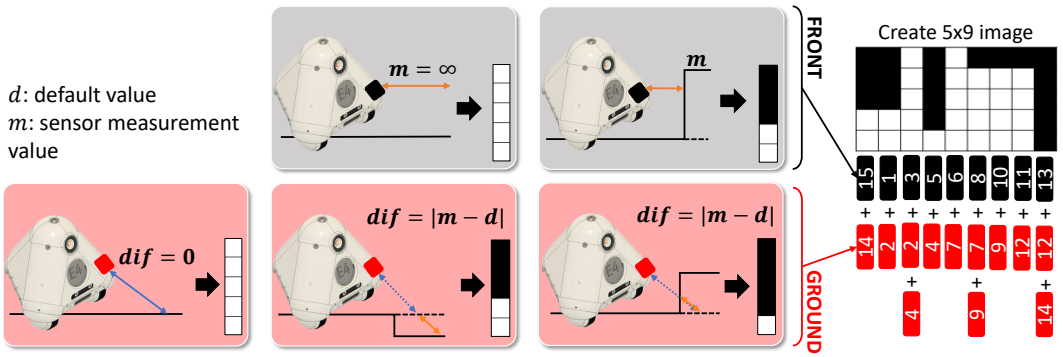
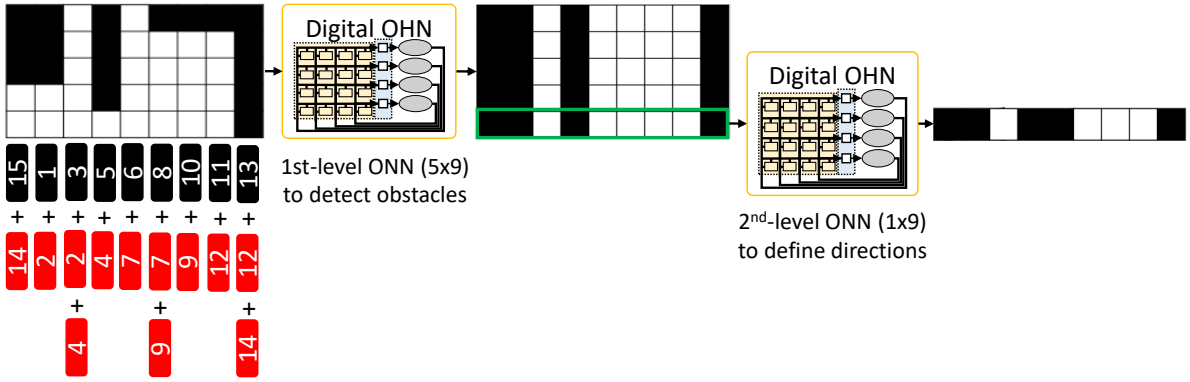
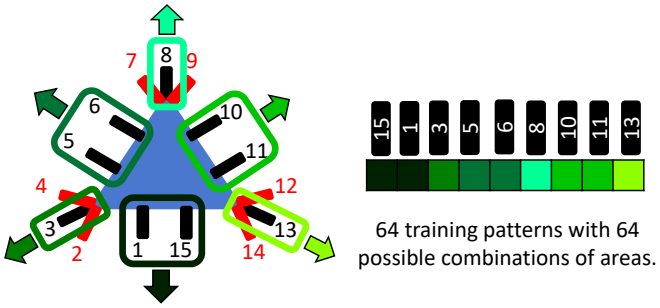


FIGURE 4.6 – *Encoding data from sensors into images, using thermometer encoding with multiple sensors per column.*



(a) *Cascaded OHNs flow for the OAV application.*



(b) *Training pattern definition of the second cascaded OHN for OAV on E4.*

FIGURE 4.7 – *Cascaded OHNs for OAV application.*

Training of the 512 patterns is performed with the simple Hebbian learning rule [28]. Note, that it is too large compared to the usual Hebbian capacity limit but it works because patterns are uncorrelated. We noticed that by learning all 512 patterns, weights are equivalent to training 9 independent OHNs, one for each column, with 2 patterns. The output of the first OHN contains an image of black and white columns and each row is identical. Thus, the second OHN uses one row from the first OHN as input, and outputs patterns corresponding to free directions. We define six possible directions corresponding to the three triangle phases and the three triangle angles. We combine columns corresponding to those six areas and train the second OHN with all possible combinations of those six areas, see Figure 4.7b. There are 64 possible outputs for the second OHN, which is also trained with the Hebbian learning rule [28]. Note, 64 patterns are again too large to be learned with Hebbian on an 8-neuron OHN. Thus, we consider additional self-coupling, as it showed good performances to train a large number of

correlated patterns in HNNs [215]. The second OHN outputs are then sent back to the OAV board, which decides the direction to follow. We define the best direction as the middle of the larger available area. Note, black and white pixels are associated to $\{-1\}$ and $\{+1\}$ during the training process with Hebbian, and to $\{0^o\}$ and $\{180^o\}$ oscillator phases during the inference process with OHN, respectively.

4.3.4 Cascaded digital OHN implementation on E4

We test our solution with the two cascaded OHNs implemented on an FPGA board and integrate it into the E4 robot. OHNs are implemented on the CMOD A7 development board from Digilent, which contains an Artix-7 FPGA, following the digital OHN design [170]. Sensory data measurements and their encoding to OHN are performed via the microcontroller on the OAV board. We connect the OAV board with the FPGA board using a universal asynchronous receiver transmitter (UART) communication protocol, see Figure 4.8. UART communication is a well-known serial communication protocol used in embedded electronics that is asynchronous. It uses only two wires, which is an attractive solution for embedded devices. One wire communicates from device A to B, while the other wire communicates between device B to A. Its main drawback is the communication latency due to serial information transmission. We choose UART first for its simple implementation, and then because we show later that in this OAV application, the latency of sensor measurements is higher than UART latency.

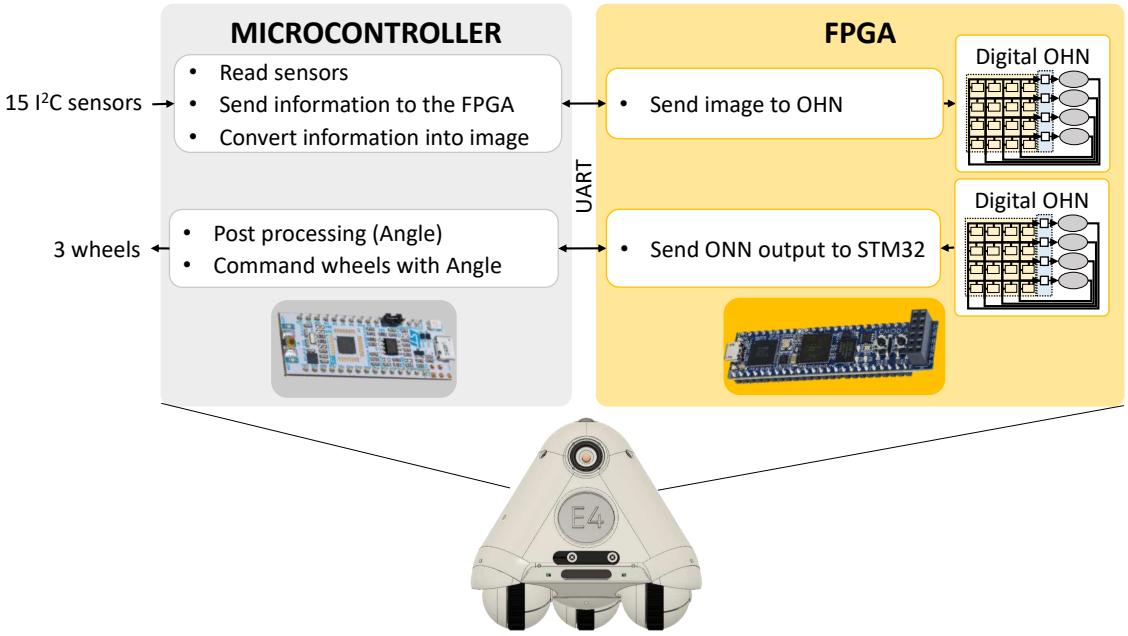


FIGURE 4.8 – *System architecture of the cascaded OHN for OAV on FPGA inside the E4 robot.*

The entire process starts with the OAV board, which reads the distance information from the 15 ToF proximity sensors using the I^2C communication bus. The OAV board microcontroller encodes the information into a 5x9 image and sends the image to the FPGA board through the UART communication interface. The computation of both OHNs, one by one, is done inside the FPGA board. We initialize the first OHN and let it compute until stabilization. The first OHN output initializes the second OHN and after stabilization, the output of the second OHN, corresponding to obstacle-free areas around the robot, is sent back to the OAV board through the UART communication interface. The OAV board finally chooses one of the best directions from all obstacle-free directions and sends the final direction to the wheel control system to

move the robot in the corresponding direction. We repeat this process every time new sensory data are available.

4.3.5 OAV on E4 system performances

We report on the OHN OAV solution's accuracy, timing, and computational resources implemented on the FPGA. In addition, we compare with classical software solutions. To do so, we develop and implement equivalent algorithms in the microcontroller of the OAV board. We create a first algorithm to detect obstacles from sensory data measurements and a second algorithm to reproduce the behavior of the second OHN and define available directions. We evaluate the computation time of each algorithm and compare it with the computation time of each OHN. We measure the computation time of both OHN and software solutions using an oscilloscope sampling at 100MHz and we round values to microsecond precision. Performance results are shown in Table 4.1, and computation time comparisons between OHN and software algorithms are shown in Table. 4.2.

TABLE 4.1 – *Performances of the OAV function on the E4 robot.*

Demonstrator characteristics OHNs Performances FPGA - Artix 7: xc7a35	2-OHN solution OHN 5x9	 OHN 1x9
#Training patterns	512	64
LUTs - 100%: 20 800 (%)	32.11	
Flip-Flops - 100%: 41 600 (%)	7.74	
OHN frequency F_{osc}(KHz)	187.5	
OHN Initialization	15 us	4 us
Computation time	10 us (2 T_{osc})	10 us (2 T_{osc})
Accuracy (%)	100	100
Full system performances (FPGA frequency: 12 MHz)		
15-sensor measurement	27 ms	
UART sensor value transmission	8 ms	
UART direction reception	500 us	
FPS	30	
OHNs Init and Comp. time	44 us (OHN 5x9 + OHN 1x9)	
Robot lifetime estimation	2h/3h	

TABLE 4.2 – *Computation time comparison between software solutions and OHN solution.*

	Software solution (STM32 @80MHz)	OHNs solution (@187.5 KHz)
Detect obstacles	15 us	10 us (2 osc cycles)
Define direction	5 us	10 us (2 osc cycles)
Full system	20 us	20 us

First, we observe that our cascaded OHN for OAV solution uses a small number of resources, up to 32% of LUTs, and up to 8% of Flip-Flops of the Artix-7 FPGA. We note that Artix-7 FPGA is a small size and low-power FPGA, meaning that our cascaded OHNs can easily be integrated into industrial systems without a significant power overhead. Both OHNs achieve 100% accuracy. So, for each possible sensory input, both OHNs stabilize on the correct

output information to allow the robot to move in a direction without obstacles or hollows. Additionally, the total initialization and computation time of both OHNs is about 40 μs, which is much shorter than the time needed to read the 15 distance values from the ToF proximity sensors. Note, that the measurement time for the 15 sensors has also been optimized by using the continuous I^2C reading method.

As mentioned in Chapter 2, the digital OHN initialization process is serial (one bit at a time), thus, the initialization time increases linearly with the number of neurons while the computation time stays constant due to parallel computation. This application shows that only two oscillation cycles are necessary for OHN to stabilize, reaching around 10 μs computation time for each OHN. Thus, our OHN solution satisfies the real-time constraints given by the latency of sensor measurements. A video of the E4 robot with cascaded OHNs configured for OAV is available on Youtube. OHN computation time is irrelevant in this application as the sensor's measurement limits timing performances. OHN can be advantageous for other robotic applications treating sensory data with real-time constraints.

Comparing the OHN computation time with classical software solutions shows that OHNs are as fast as using software algorithms implemented on a micro-controller, see Table 4.2. We compare only computation times without initialization time to have a meaningful comparison. It is also important to state that the OAV microcontroller runs at 80 MHz while OHN only oscillates at 187.5 kHz. This indicates that using OHN with higher frequency capacity can surpass the microcontroller timing performances for OAV applications. Note, that the software solution reproduces the OHN behavior using software functions implemented on the STM32 microcontroller. Thus, the accuracy obtained with the software solution is also 100%. In addition, timing performances of software solutions implemented on micro-controllers grow linearly with the number of data to treat, while the computing time of OHN remains constant (i.e., only a few cycles to stabilize) when increasing its size. So, if there are many data to treat, OHN computation will be faster than a software solution. Another solution to improve timing performances in case of a large number of sensory data is to use multiple OHNs to compute in parallel. For example, computing the two OHNs implemented for OAV in parallel could increase the final cascaded OHN computation time.

Finally, we show in Table 4.2 that the global timing performance of our OHN solution is slowed down by the UART serial communication between the OAV board and the FPGA. Another solution for applications with higher real-time requirements will be to consider a complete SoC implementation, including sensor reading, with either programmable logic and an integrated processor, or only with programmable logic. Such integration inside a complete industrial robot requires anticipation during the system design and development. In the next section, we propose a SoC implementation integrating sensor reading in the processing system (PS) of the Zynq of a Zybo development board. However, we do not implement it on the E4 industrial robot, but on a development robot based on an Arduino control board.

4.4 All-in-one SoC architecture for OAV on Arduino robot

We previously highlighted the cascaded OHN architecture can perform OAV on an industrial mobile robot equipped with proximity sensors. However, the current system architecture, considering separated chips for sensor reading, conditioning, and treatment, limits the latency performances of the system. Thus, in this section, we modify the system to perform sensor reading, conditioning, and treatment in a single chip. In this case, it requires more adaptations to integrate the system into an already built robot, such as the E4 robot, thus we use a deve-

lopment robot from Arduino. The robot is a unicycle mobile robot furnished with a two-wheel drive connected to direct current motors driven by an Arduino UNO board. The Arduino board is based on a microcontroller to control the robot's direction, we call it the control board. For the OAV application, we equip the robot with 8 ToF proximity sensors equivalent to the one used in E4, referenced as VL53L0X, evenly distributed in the front part of the robot, see Figure 4.9. Thus, similar to the OAV on E4, the system reads the proximity sensor values to detect obstacles around the robot and treats the detected obstacle information to define a new direction and avoid obstacles. The system is still open-loop. The difference is that we perform OAV using a SoC capable of reading, conditioning, and treating sensory data based on the cascaded OHN architecture.

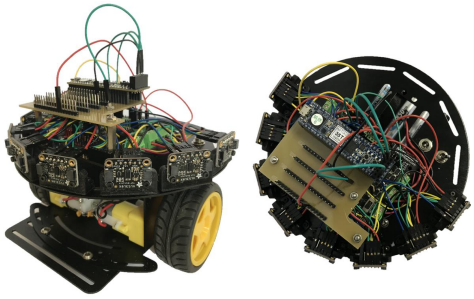


FIGURE 4.9 – *Arduino development robot equipped with 8 proximity sensors evenly distributed in the front.*

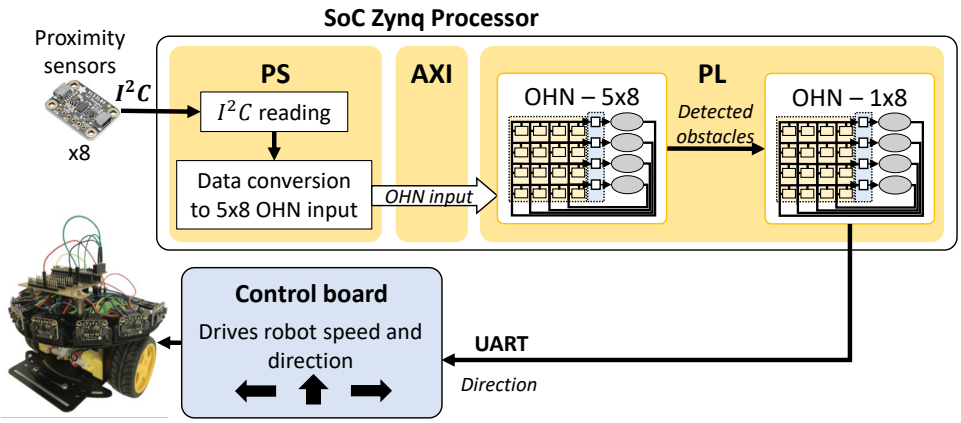


FIGURE 4.10 – *Architecture of the OHN-based SoC for OAV on the Arduino development robot.*

4.4.1 SoC architecture for OAV

We consider a similar ONN on SoC architecture, as in Chapter 3, Section 3.3.2 taking advantage of the Zynq processor of a Zybo-Z7 development board equipped with both PL and a PS. So, digital OHNs are implemented with PL resources, and outputs are directly connected to PL while inputs are connected to PS. The SoC architecture first reads sensory values and converts them to an OHN-compatible format inside PS. Then, the OHN input is transmitted using the AXI4-Lite communication protocol to PL for processing. After processing, the output is directly sent from PL to the control board. For example, in the case of the OAV application, we connect the 8 proximity sensors to the PS of the SoC and read them through the I^2C serial communication protocol. Then, we condition the sensor values into an OHN-compatible input format, and we transmit the OHN input to PL to perform the OHN computation. As with OAV on E4, once cascaded OHN computation is completed, we associate the output with a novel ro-

bot direction. Finally, we transmit the output direction via UART from PL to the control board. The control board still drives the robot motors to move the robot in a specific direction, see Figure 4.10.

4.4.2 OAV on mobile robot

The cascaded OHN configuration is similar to the E4 robot, but with only 8 proximity sensors, and all of them are horizontally directed. Thus, the robot can detect obstacles but not hollows. The first OHN is a 40-neuron OHN that takes as input an image generated in PS from the sensor values. We re-use a 5-pixel thermometer encoding, associating one sensor per column, creating a 5x8 image for OHN input. We set pixel resolution to 55 mm based on our observations from real condition tests. Consequently, we input distances from 0 mm to 275 mm, see Figure 4.11.

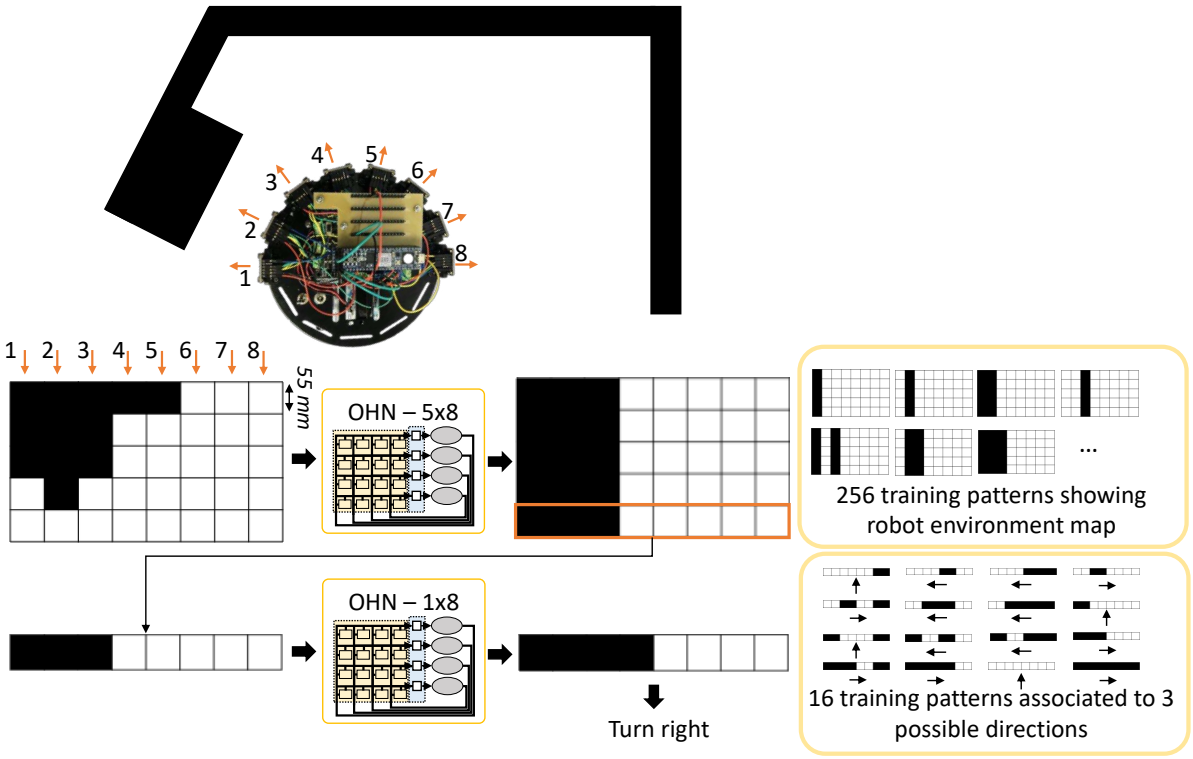


FIGURE 4.11 – *Cascaded OHNs configuration for the OAV application.*

We configure the OHN with 256 learning patterns representing all possible black-and-white column combinations using the Hebbian learning rule [28]. Thus, the first OHN generates images representing an environment map of the robot front area with black columns representing obstacles, and white columns representing no obstacles, see Figure 4.11. Again, the role of the second OHN is to define the available directions from the environment map given by the first OHN. One row of the first OHN output serves as an input to the second OHN, so the second OHN is an 8-neuron OHN with 1 row and 8 columns. We configure the second OHN to output possible directions for the robot. To train the second OHN, we use 16 patterns representing possible OHN directions, see Figure 4.11. We associate each learning pattern with one possible direction. Once the two cascaded OHN computations are complete, the system sends the following direction to the robot control board using UART, which moves the robot in the corresponding direction. Training is also performed with the Hebbian rule [28].

4.4.3 OAV on SoC performances

Table 4.3 shows OHN performances as well as results of the entire SoC system. First, we point out the first OHN reaches 100% accuracy. It means that for all possible inputs (1 679 616 possibilities), the first OHN computes and stabilizes to the correct learning pattern. However, the second OHN only obtains 74% accuracy. So, for each possible input (256 possibilities), the second OHN computes and stabilizes to the correct learning pattern for 74% of the cases. For the other 26%, the OHN either stabilizes to another non-learned pattern, called a spurious pattern, or does not stabilize. In this case, we have a backup post-processing algorithm that uses output information from the first OHN (detected obstacles) to compute the robot's direction. The algorithm replaces the second OHN and chooses the direction with fewer obstacles. Note that we performed experimental tests on the robot with real conditions, and for 1000 inferences, spurious cases occur for less than 5% of the input patterns.

TABLE 4.3 – *Performances of the OAV application with OHNs embedded on a SoC compared with the previous version implemented on E4.*

OHNs				
	OHN-SoC for OAV Zybo-Z7 board		OAV on E4 CMOD A7 board	
Characteristics	First OHN	Second OHN	First OHN	Second OHN
Size (#neurons)	40	8	45	9
Synapses resolution	5 bits	5 bits	5 bits	5 bits
LUTs (% utilization)	2401 (4.5%)		6679	
Flip-Flops (% utilization)	2242 (2.1%)		3220	
Training images	256	16	512	64
ONN computation time	24 us	17 us	25 us	14 us
Accuracy	100 %	74 %	100%	100%
Full System				
LUTs (% utilization)	2950 (5.5%)		6679	
Flip-Flops (% utilization)	2979 (2.8%)		3220	
8-sensor measurement time	18 ms		27 ms	
AXI transmission	4 us		-	
FPS	40		30	
Battery spec	6V - 2850mAh		-	
Current consumption	700 mA		-	
Robot lifetime	4h		2h/3h	

Next, we highlight that the digital implementation of the two cascaded OHNs uses only a small amount of resources, with less than 6% of the LUTs, and around 3% of the Flip-Flops. Also, the entire OHN-based SoC, integrating the AXI bus protocol and OHNs, does not drastically increase resource utilization. It indicates that our SoC architecture can easily be integrated into a larger system without significant PL resource impact.

Also, we report a sensor reading time of $18ms$, a conditioning and transmission time from the software unit to FPGA of $15us$, a computation time of the two OHNs of $40us$, and a transmission time from the FPGA to the control board of $1ms$, reaching around 40 FPS. It shows that OHN computations are much faster than the time required to read the 8 sensor values. While the AXI parallel protocol allows a fast transmission time, it does not reduce the full-system latency. Note, that we still use a UART communication to transmit the cascaded OHN output direction to the control board. However, instead of transmitting serially both cascaded OHN inputs and outputs through UART, we only transmit output direction and use AXI

parallel protocol otherwise. However, as before, for a more time-constrained application, we could improve the latency by implementing every step, from sensor reading to converting and processing, in the PL resources of the SoC processor but it would increase the PL resources utilization.

This work presents a novel real-time edge solution to perform sensing-to-action computing with cascaded OHNs. The OAV SoC architecture is similar to the SoC architecture introduced for on-chip learning in Chapter 3, taking advantage of the Zynq processor with PS and PL. Continual on-chip learning is useful for systems in evolving environments, such as robots navigating in an unknown environment, and trying to avoid obstacles. In Chapter 3, we introduced the on-chip learning architecture and validated it on pattern recognition tasks. However, we did not apply it on a realistic edge application. Thus, in the next section, we propose to apply on-chip learning to the OAV application, considering learning from the environment.

4.5 On-chip learning for OAV

We previously used the mobile robot OAV application considering an all-in-one SoC with cascaded OHN computing capable of reading, conditioning, and treating sensory data. Additionally, we proposed a system architecture to allow on-chip learning with OHN implemented digitally, as described in Chapter 3, Section 3.3.2. However, we did not apply the on-chip learning to a concrete edge application. Thus, here we propose to combine the OAV application with additional on-chip learning capability.

4.5.1 On-chip SoC architecture

We consider the cascaded OHN-SoC architecture from Section 4.4.1 implemented in the Arduino robot equipped with 8 ToF proximity sensors, see Figure 4.10. However, in the previous cascaded OHN-SoC, the final direction output was sent from PL to the robot control board while here the control board is only connected to PS. Thus, first, the proximity sensor values are read and encoded into the 5x8 image in PS. Then, PS sends the image to the cascaded OHNs for inference. After computation, the cascaded OHN output is sent back to PS to be associated with a direction that is finally sent from PS to the Arduino control board. Previous OAV applications were open-loop systems using proximity sensor measurements as input and outputting a novel direction for the robot. However, in robotics, closed-loop systems are important to ensure correct decisions. In this work, we propose a closed-loop OAV with feedback from cascaded OHN output to a training block to permit training or re-training the OHNs if necessary. In particular, we propose two training solutions, one considering prior knowledge, and the other one without prior knowledge.

We propose an architecture that incorporates the on-chip learning in the ONN-based SoC architecture configured for robot OAV, see Figure 4.12. In particular, we propose to keep a pre-trained first-level OHN to detect obstacles, while we consider the second OHN, which defines the direction, as non-trained. The open-loop OAV system becomes a closed-loop system. The closed-loop system integrates feedback from the detected obstacles to modify the second OHN output decisions if necessary. Thus, we include a transfer of the first OHN output to PS to train the second OHN.

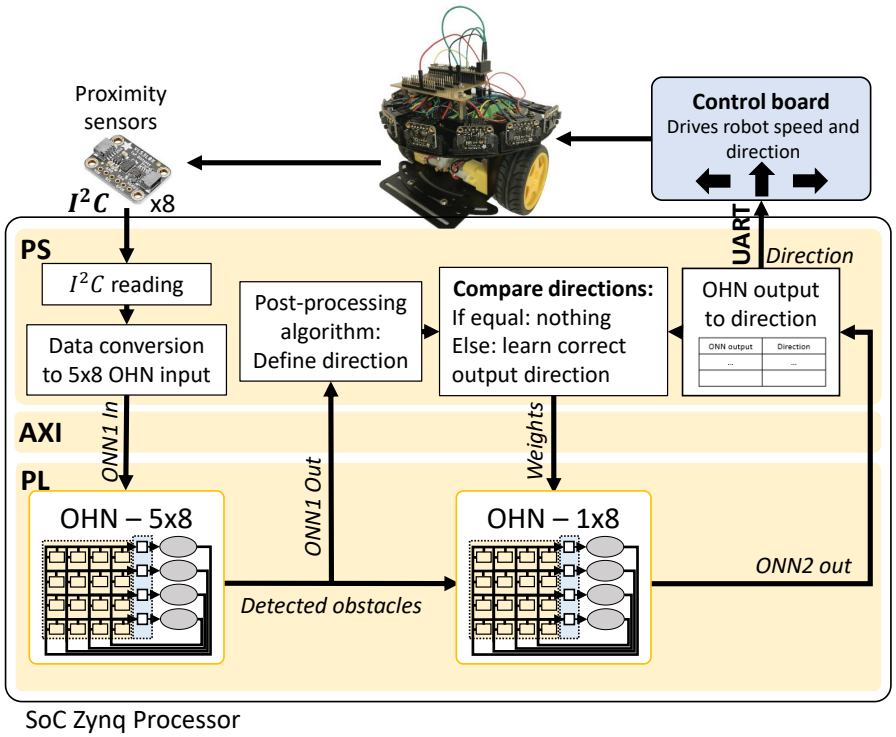


FIGURE 4.12 – *Zynq architecture for OAV with two cascaded OHNs and second OHN on-chip learning based on post-processing from environment obstacles.*

4.5.2 Training OHN on-chip for OAV

The OAV process starts by reading the 8 sensor measurement values from PS through the I_2C protocol. Sensory data are converted to an image to be compatible with the first OHN, OHN_1. The obtained image is sent from PS to PL as input of OHN_1, pre-trained to detect obstacles [216]. The OHN_1 output, $OHN1_{out}$ is then sent to the second OHN OHN_2 as well as back to PS to be used as input of a post-processing (PP) algorithm. Both PP and OHN_2 compute in parallel. The OHN_2 output, $OHN2_{out}$ is sent back from PL to PS to be converted to a direction information dir_{OHN2} from a table which associates OHN_2 patterns, TP_{OHN2}, to directions. Note, if the table is empty or if $OHN2_{out}$ does not correspond to any known training patterns TP_{OHN2}, direction dir_{OHN2} is set as unknown. In parallel, the PP also computes a direction dir_{pp} from $OHN1_{out}$. Both directions dir_{OHN2} and dir_{pp} are compared and a difference between both directions triggers the training of a novel pattern for OHN_2. The PP is an important factor as it determines when training is triggered. It computes the best direction depending on the position of the obstacles around the robot given by the first OHN output $OHN1_{out}$. The PP is described in Algorithm 1.

The training process triggered by the post-processing is different depending if the solution uses prior knowledge or not. In the solution with prior knowledge, the table associating TP_{OHN2} to dir_{OHN2} is known and integrated into the system with patterns from [216]. Thus, every time a novel training process is triggered, one of the patterns from the table, TP_{OHN2}, is memorized in the OHN using Hebbian. Once all patterns are learned, the OHN is fully configured and the post-processing is not used anymore, see Algorithm 2. However, in a more realistic application, there might not be prior knowledge or only a part of it. Thus, we also propose a solution with no prior knowledge.

Algorithm 1 Post-processing algorithm

N_{max}: max. neighbour sensors without obstacles, θ_s: sensor angle, $\theta = avg(\theta_s(N_{max}))$
if $N_{max} < 4$ **then**
  if $\theta < 0$ **then**
    $dir_{pp} = left$
  else
    $dir_{pp} = right$
  end if
else
  if $((\theta < 30)$ and $(\theta > -30))$ **then**
    $dir_{pp} = front$
  else
    if $(\theta <= -30)$ **then**
      $dir_{pp} = left$
    else
      $dir_{pp} = right$
    end if
  end if
end if

Algorithm 2 Training algorithm with prior knowledge

Init W_1, W_2
Init TP_{OHN2}: training patterns from [216]
Init $dir(TP_{OHN2})$: direction associated to TP
for Each sample S_t **do**
  $OHN1_{out} = OHN_1(S_t, W_1)$
  $OHN2_{out} = OHN_2(OHN1_{out}, W_2)$
  if $OHN2_{out} == TP_{OHN2}$ **then**
    $dir_{OHN2} = dir(TP_{OHN2})$
  else
    $dir_{OHN2} = Unknown$
  end if
  $dir_{pp} = post - processing(OHN1_{out})$
  if $dir_{OHN2} == dir_{pp}$ **then**
    $Nothing$
  else
    $W_2 = W_2 + H(TP_{OHN2}(i))$ with $dir(TP_{OHN2}(i)) = dir_{pp}$
    $i = i + 1$ until all $TP_{OHN2}(i)$ learnt
  end if
end for

In the solution without prior knowledge, in the beginning, the table associating TP_{OHN2} to directions is empty. So, when a novel training command is triggered, the second OHN memorizes its input, $OHN1_{out}$, as a novel training pattern, TP_{OHN2}, using Hebbian, see Algorithm 3. The associated direction included in the table is computed by PP. Note, the opposite pattern, $-OHN1_{out}$, also becomes an entry of the table with its associated direction computed by the post-processing, because with Hebbian, when a pattern is learned, its opposite version is also automatically learned. After numerous cycles without any difference between the direction from the second OHN output, and the direction from the post-processing, we consider the second OHN trained, and the post-processing algorithm stops computing.

Algorithm 3 Training algorithm without prior knowledge

Init W_1, W_2
Init TP_{OHN2} = $dir(TP_{OHN2})$ = empty
for Each sample S_t **do**
  $OHN1_{out} = OHN_1(S_t, W_1)$
  $OHN2_{out} = OHN_2(OHN1_{out}, W_2)$
  if $OHN2_{out} == TP_{OHN2}$ **then**
    $dir_{OHN2} = dir(TP_{OHN2})$
  else
    $dir_{OHN2} = Unknown$
  end if
  $dir_{pp} = pp(OHN1_{out})$
  if $dir_{OHN2} == dir_{pp}$ **then**
    $Nothing$
  else
    $TP_{OHN2}(i) = OHN1_{out}$
    $W_2 = W_2 + Hebbian(TP_{OHN2}(i))$
    $dir(TP_{OHN2}(i)) = dir_{pp}$
    $TP_{OHN2}(i+1) = -OHN1_{out}$
    $dir(TP_{OHN2}(i+1)) = pp(-OHN1_{out})$
    $i = i + 2$
  end if
end for

We test both solutions by first performing simulations of an HNN-based emulator on Matlab, and then by implementing the system in the robot and using real sensory data. In Matlab, we generate random input samples corresponding to possible sensory data and we apply it as input of the first HNN-based emulator. In the real robot system, inputs are real inputs from proximity sensors. For each input, either Algorithm 2 or Algorithm 3 is applied. We evaluate accuracy after each input sample. Accuracy is evaluated only on the second OHN as we know the first OHN has 100% accuracy. Accuracy is evaluated from all possible inputs of the second OHN, 256 possible inputs, with two different methods. First, *accuracy TP* is computed by comparing $OHN2_{out}$ with all TP_{OHN2}. If the $OHN2_{out}$ corresponds to one TP_{OHN2}, *accuracy TP* increases. It means that OHN_2 is able to take a decision from the corresponding input, no matter if the decision is correct. Second, *accuracy direction* is calculated by comparing dir_{OHN2} and dir_{pp}. If both directions are equal, *accuracy direction* increases. It allows controlling if OHN_2 stabilizes to one of the training patterns TP_{OHN2} and if the associated direction dir_{OHN2} is correct, compared with the direction generated by the post-processing.

Figure 4.13 shows accuracy evolution depending on the input samples through time, as well as the evolution of the number of training patterns depending on their associated direction using the HNN emulator on Matlab for the first solution with prior knowledge. It highlights that all training patterns are learned after less than 40 input samples, which is fast. Also, it shows that after training all patterns TP_{OHN2} from the association table, accuracy is equal to accuracy using hardcoded weights from [216], meaning incremental Hebbian gives the same results as Hebbian in this application. However, it displays that accuracy can be higher before reaching final accuracy when all patterns are learned. Thus, there might be a better PP solution that would stop learning novel patterns in case the accuracy is reduced. Figure 4.13 also shows final accuracy is equal for 5-bit signed weights and full precision weights. Thus, we should obtain equal results with the digital OHN implemented on FPGA.

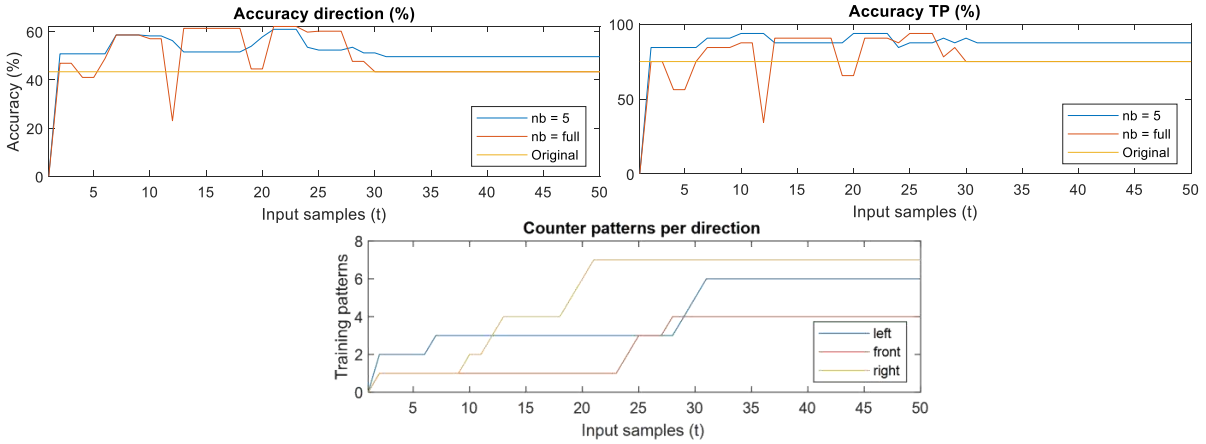


FIGURE 4.13 – *Simulation results of the HNN emulator on Matlab for the first solution with prior knowledge. Accuracy is presented for two weights precision, $nb = 5$ for 5-bit signed precision, and $nb = full$ for full signed precision. $Original$ accuracy corresponds to accuracy for hardcoded weights [216].*

Figure 4.14 shows accuracy evolution depending on the input samples through time, as well as the evolution of the number of training patterns using the HNN emulator on Matlab for the second solution without prior knowledge.

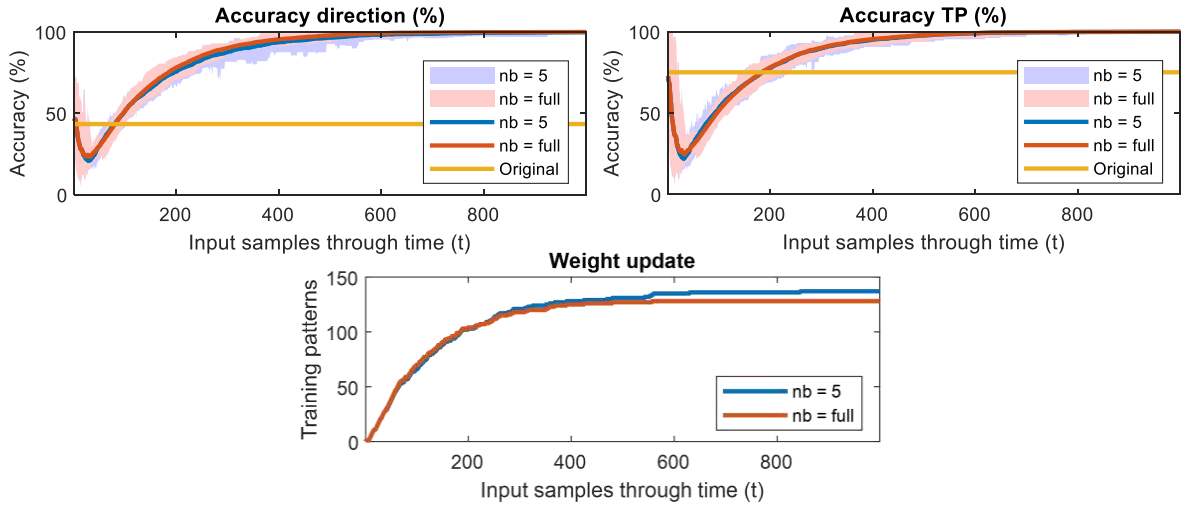
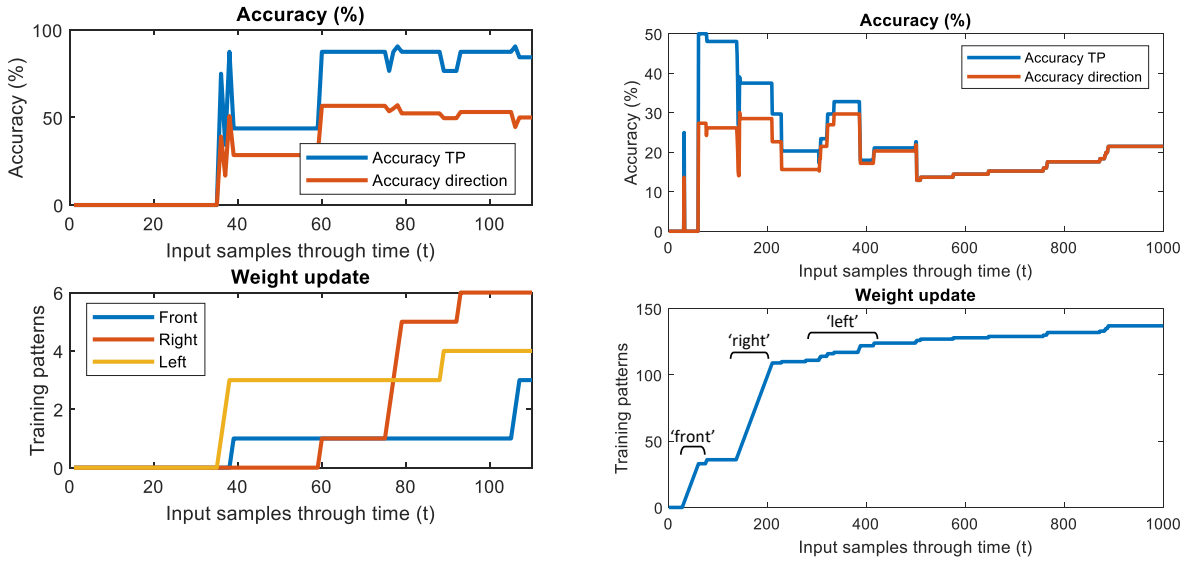


FIGURE 4.14 – *Simulation results of the HNN emulator on Matlab for the solution without prior knowledge. nb represents the bits' precision. $Original$ corresponds to accuracy for hardcoded weights from [216]. Accuracy is computed for 50 different trials, the colored area corresponds to min-max bounded accuracy, and colored lines correspond to average accuracy results.*

In this case, obtaining a stable high accuracy takes more time than with the solution with prior knowledge, around 600 samples to stabilize and stop learning novel patterns. It starts with a high learning rate during the 200 first samples, and then the learning rate reduces. The solution without prior knowledge allows to reach a higher accuracy than the solution with prior knowledge, almost 100% accuracy for both *accuracy TP* and *accuracy direction*. In particular, after 900 random samples, the OHN has learned all possible patterns. However, it is not interesting because if all patterns are learned, the association table contains all possible OHN inputs associated with a direction, so no matter which OHN output it can decide on a direction. However, we strongly believe this behavior comes from the random input patterns, thus all possible inputs of OHN_2 are crossed while in a real environment, real obstacles will not match all possible OHN_2 inputs, limiting the number of training patterns. Figure 4.14 also highlights that weight quantization to 5-bit signed precision does not impact much accuracy and weight

update evolution. Thus, we expect similar experimental results with the OHN implemented on FPGA.

Figure 4.15 shows accuracy evolution depending on the input samples through time, as well as corresponding evolution of the number of training patterns using the OHN-based on-chip learning OAV system implemented on the Zynq board with real proximity sensors. Figure 4.15a considers the first solution with prior knowledge. It shows that it takes between 80 and 110 cycles to learn all possible training patterns. However, this depends on the environment and the obstacles the robot will encounter. It is possible for the robot to never encounter some of the obstacle configurations which would trigger learning of some training patterns. As expected, the final obtained accuracy, after learning all patterns, is equal to the previous accuracy obtained in simulation.



(a) *Experimental results of the full system implemented on Zynq for the first solution with prior knowledge.*

(b) *Experimental results of the full system implemented on Zynq for the second solution without prior knowledge.*

FIGURE 4.15 – *Digital OHNs results for solutions with and without prior knowledge.*

Figure 4.15b considers the second solution without prior knowledge. It shows that the number of training patterns almost stabilizes after 500 samples, however, accuracy is still low compared to simulation results. This behavior confirms that in a real environment, not all input obstacle configurations happen, and so there are fewer training patterns, reaching lower precision. However, if after 500 samples, the number of training patterns does not increase a lot, it means that both dir_{OHN2} and dir_{pp} are equal almost all the time, thus for the encountered input samples, the system takes a correct decision. Figure 4.15b also shows the incremental learning capacity. For example, in the beginning, the robot encountered a front obstacle and learned to retrieve it. Later, the robot faced an obstacle from the right and learned how to take a decision from it, until it learned the main patterns corresponding to the three directions.

Table 4.4 shows performances of the OHN-based on-chip learning close-loop system configured for OAV. It highlights that sensor measurements are still slower than the latency required for OHN inference, including the latency of data transmission. So, the system performs OAV in real-time given timing requirements from sensor reading time, as in the previous OAV demonstrators. Also, training latency is fast, the post-processing only takes a couple of microseconds to compute and the Hebbian computation requires tens of microseconds. Note, the post-processing could replace the second OHN directly as it is much faster. However, considering the PS frequency of 325 MHz, compared to the OHN oscillation frequency of 187.5 KHz,

using OHN with a faster frequency could be of interest for time-constrained systems. The slowest part is the transmission of weight values through AXI from PS to PL which can be optimized with parallel weight transmission and reducing weight precision to allow sending more weights at a time. However, this may reduce OHN precision. Additionally, Table 4.4 shows that the system uses a small amount of PL resources. Note, that both solutions, with or without prior knowledge, use the same amount of PL resources, as the difference between the two methods is performed in PS. However, we know from [197] that the number of resources increases drastically with the number of reconfigurable synapses, in the case of OHN with on-chip learning. In this work, only the second OHN, which has only 8 neurons, uses reconfigurable synapses, allowing for low-resource implementation.

TABLE 4.4 – *On-chip OAV learning system performances on Zybo-Z7 board.*

OHN-level (OHN frequency: 187.5 KHz)		
	OHN1	***OHN2***
Size (#neurons)	40	8
Synapses resolution	5 signed bits	5 signed bits
LUTs (%)	4529 (25.7%)	
Flip-Flops (%)	3155 (9.0%)	
Training	Pre-trained 256 TP	Incremental training
Accuracy	100%	NA
System-level (PS frequency: 325 MHz)		
LUTs (%)	5183 (29.5%)	
Flip-Flops (%)	4127 (11.7%)	
Inference latency		
- Sensor measurement	18 ms	
- Transmission & OHN comp.	122 μs	
- OHN comp.	51 μs	
Training latency		
- Hebbian comp.	13 μs	
- Transmission	495 μs	
- Post-processing comp.	1.85 μs - 3.1 μs	

We noticed that the first solution using prior knowledge converges quite fast to final weight values once all training patterns are trained. However, the second solution without prior knowledge does not have a limited number of training patterns, thus it requires a larger number of samples to reach stabilization in the number of training patterns. However, such statements are really dependent on the environment, and how fast the robot will explore it. Thus, it requires more investigations on realistic environmental data to understand better the number of samples necessary to have a stable system. Finally, we highlighted that both solutions allow for real-time efficient OHN on-chip learning and inference given the latency requirements from sensor measurements. The system is able to learn a novel pattern in real-time, avoiding most of the obstacles from the environment after numerous samples, validating the proposed solutions. This is, up to our knowledge, the first time OHN incremental and dynamical training is applied to a real-world edge application.

In this chapter, we proposed a novel ONN architecture with cascaded OHNs implemented in digital, that allows performing novel edge applications, like the OAV on mobile robots. More than that, we use the OAV system to apply on-chip learning to a real-world application, considering cascaded OHNs trained from an evolving environment. For all demonstrators, cas-

caded OHNs are implemented digitally, making it easy to perform feed-forward transmission between the two cascaded OHNs. However, feed-forward connections with analog oscillators can be challenging. In the next section, we propose an analog solution to cascade two OHNs using majority gate layers. We apply the system to a novel edge application, performing image edge detection.

4.6 Analog cascaded OHNs for image edge detection

Previously, we discussed the need for novel ONN architectures to go beyond OHN, due to scalability and performance limitations. That is why we proposed a novel architecture, cascading OHNs, that we implemented with digital OHNs on FPGA, and that we applied to an OAV application, embedded in mobile robots. Cascaded OHNs operate as independent OHNs connected feed-forwardly such that they compute one after the other, considering the output of one OHN as input of the next one. In analog, ONN has already been applied efficiently as OHN with various oscillators' implementations [124, 131, 141, 217]. However, to our knowledge, most of the proposed OHN implementations do not permit uni-directional connections, making it impossible to implement feed-forward cascaded OHNs. In the meantime, one of the advantages of the ONN computing paradigm is its analog natural computation. Thus, in this work, we propose a novel architecture to cascade two OHNs with a feed-forward (FF) layer based on an analog majority gate (MG) function. More than that, we apply the solution with two cascaded OHNs using a FF-MG layer to an image processing application, as the image edge detection. We start by describing the analog cascaded OHN system, from the analog OHN itself to the FF-MG layer. Then, we explain how we apply it to image edge detection.

4.6.1 Analog FF-MG layers for cascaded FC-OHNs

In this work, we propose a solution to cascade two analog OHNs with an analog-based FF-MG layer.

Analog OHN: In this work, we consider CMOS relaxation oscillators coupled with resistances, see Figure 4.16. The resistive couplings are mapped from the weights generated with unsupervised Hebbian learning rules following the method from [195]. Once the coupling is configured and the circuit is mapped correctly, we initialize each oscillator. The initialization consists of applying a VDD_{osc} to each oscillator. The phase shift is encoded in the delay of activation of the VDD_{osc}. For example, if we want to initialize two oscillators with 180^o phase shift, we will activate VDD_{osc1} at $t = 0$, and activate VDD_{osc2} at $t = T/2$, with T the period of oscillation. After initialization, each VDD_{osc} stays active so oscillator phases evolve until stabilization. Stabilization is detected after numerous period cycles with stable phases. The final phase state corresponds to the output information of the OHN.

Analog FF-MG: In this work, we propose a solution to cascade two OHNs. Thus, after the first OHN layer stabilization, we use the oscillating outputs as input of the FF-MG layer, as shown in Figure 4.16. In Figure 4.16, two FF-MG layers are represented, each layer taking as input three oscillating signals from the first OHN layer, performing MG function, and using the two FF-MG output as input of a second-layer OHN. The FF-MG layer contains various levels. A first level of analog buffers is necessary to send the first-layer OHN outputs to the FF-MG layer without impacting the OHN behavior. Then, the second level uses oscillations from analog buffers and performs an MG function with a summing amplifier. In Figure 4.16, each

MG takes three oscillating signals as input and outputs a signal with a phase corresponding to the majority phase of the three input oscillations. The third level is a non-linear amplifier which saturates the signal to replicate an analog-to-digital conversion. From this digital signal, the fourth level synchronizes the various MG layers together. To do so, a latch is placed in between the linear amplifier output and the next-layer oscillator input. When the latch is enabled, the VDD_{osc} of the oscillator from the next layer is activated following the phase shift given by the linear amplifier. The enable signal V_{en} is set in the same moment for each FF-MG layer to synchronize them for the next OHN layer, see example in Figure 4.17.

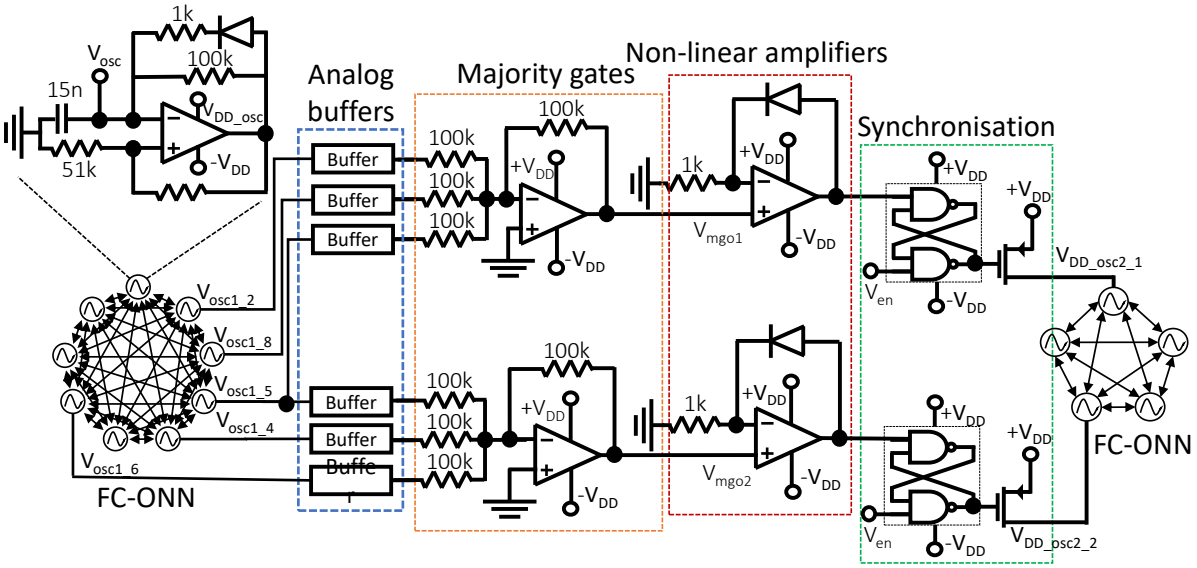


FIGURE 4.16 – *Two cascaded analog FC-ONNs with two analog MG FF layers. VDD = 3V.*

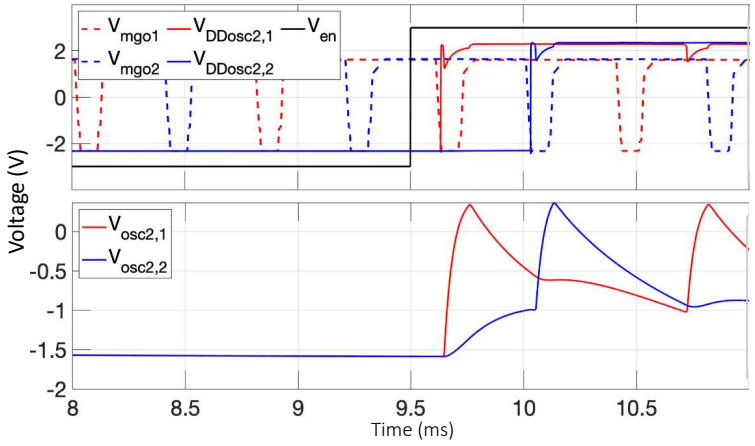
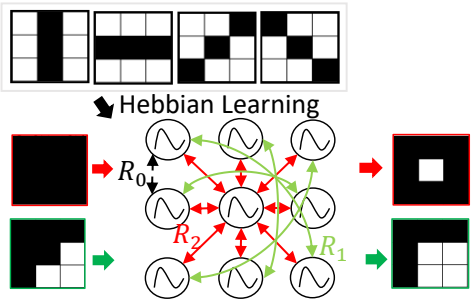


FIGURE 4.17 – *Experimental results of internal signals for 2 FF-MG layers, and second layer oscillator outputs. V_{mgo1}: output of output amplifier, V_{en}: enable signal common to both FF-MG layers, VDD_{osc}: input voltages of each second layer oscillator.*

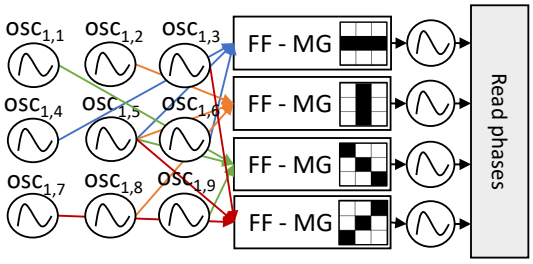
4.6.2 Application to image edge detection

We apply the architecture to perform image edge detection. Image edge detection is an important image processing task used as the first step of many broader problems, like image segmentation, or object detection [218, 219]. Image edge detection algorithms mainly use 3x3 convolution filters to scan an image and assign a gradient to each pixel of the image which indicates if an edge is detected. We try to solve image edge detection by replacing convolution

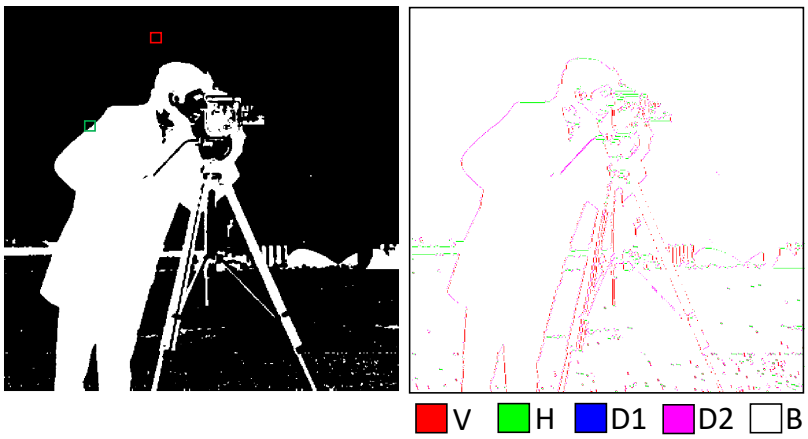
filters with the analog cascaded OHN architecture. Recently, [143] proposed a solution using an OHN trained to perform pattern recognition with 4 patterns representing 4 edges (horizontal, vertical, and diagonals), and using a post-processing algorithm to define if an edge is detected or not.



(a) *Weight of first-layer OHN. Neutral* $R_0 = 45k\Omega$*, positive* $R_1 = 40k\Omega$*, and negative* $R_2 = 47k\Omega$*.*



(b) *Architecture of the second layer for each egde.*



(c) *Simulation results for image edge detection on black and white image.*

FIGURE 4.18 – *Cascaded FC-ONN configuration and results for image edge detection.*

By analyzing their solution, we noticed that the post-processing can be adapted as a combination of FF-MG layers. Thus, we use the analog FF-MG solution instead of the post-processing using the same first-layer OHN [143] to build an analog-based all-in-one architecture. The first OHN is trained with 4 patterns representing the 4 main edges (horizontal, vertical, and diagonals), using the Hebbian rule [28] which configures the coupling resistances, see Figure 4.18a. Then, when the first OHN is stable, we use its output as input of 4 FF-MG layers to detect each edge, see Figure 4.18b. Note, we consider the second-layer OHN with neutral 0-couplings. We test and validate our solution with a hardware breadboard for two edges, horizontal and vertical, and we generalize for four edges in simulation. Note, that the current hardware implementation serves as a proof-of-concept for the FF-MG functionality, thus, it is not optimized for low-power computation. However, one can envision the FF-MG capability to be implemented on OHNs via low-power oscillator devices, like VO_2 [220, 221]. Also note, that peripherals to encode and decode phase information consume a significant amount of power [221].

4.6.3 Performances

This section presents the performances of our analog-based FF-MG architecture to cascade two OHNs adapted to image edge detection. Figure 4.18c and Figure 4.19 show the results of

our image edge detection solution on the *cameraman* large-scale image. Visually, it shows that we can detect edges efficiently. Table 4.5 shows the Jaccard Similarity Coefficient (JSC) metric that uses the state-of-the-art Canny algorithm [222] as ground truth (GT), and is compared with the state-of-the-art Sobel algorithm [223]. Table 4.5 shows that the two cascaded analog OHN using a FF-MG layer miss some important edges, and do not place correctly all detected edges.

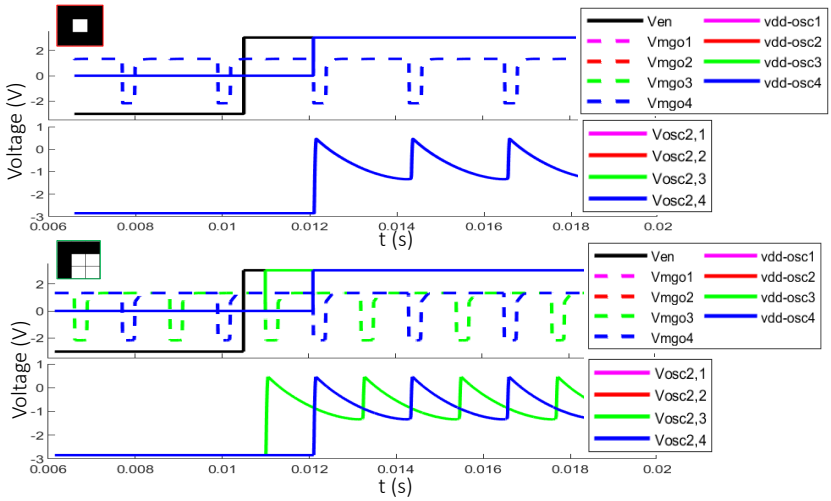


FIGURE 4.19 – *Simulation results of analog FF-MG layer for two images.*

TABLE 4.5 – *Precision of cascaded analog OHNs on image edge detection compared with Sobel based on Canny ground truth (GT)*

GT: Canny	Sobel	This work
JSC	0.5204	0.1866
Overlapping	7277	2021
Union	13983	10832

Another, important parameter for the image edge detection algorithm is latency. However, to our knowledge, there is no alternative analog image edge detection in the literature. Later, in Chapter 5 we propose digital solutions to perform image edge detection and compare them with the current analog solution, as well as with other digital image edge detection implementations. Without comparison, Table 4.6 reports the latency of our solution.

TABLE 4.6 – *Latency performances of our solution.*

Latency	analog (F_{osc}=3MHz)
3x3	OHN: 1.67 μs FF-MG: 0.67 μs
28x28	1.83 ms
512x512	613 ms

The cascaded OHN solution requires the stabilization of an OHN and an MG layer. The analog OHN stabilizes in 4-5 oscillation cycles and the analog FF-MG layers stabilizes in only 1 to 2 cycles. Note, we consider a parallel execution of the four MG layers to speed up the process. In total, to process a large image of 512x512 pixels, our solution requires 613 ms, which is quite long if we consider the typical video flow of one image every 33 ms. To allow real-time image processing with our analog cascaded OHN with FF-MG layer, we require parallelization of multiple cascaded OHN systems.

This work introduces an analog-based architecture to cascade two analog OHNs using an FF-MG activation function. The FF-MG layer is almost fully analog, apart from the digital latch

level. Analog computing can be particularly of interest for edge AI where a lot of matrix-vector multiplication operations can be performed in an energy-efficient manner. In our design, the only digital part is the latch gate, which takes care of the synchronization of the different FF-MG layers. It limits power consumption by activating the second OHN layer only after the computation of the first OHN layer. However, if all FF-MG inputs are synchronized, for example, if they are part of the same OHN, then input signals to the second OHN should also be synchronized. Thus, our FF-MG layer can adapt to a fully-analog system. We show this novel architecture is capable of performing an image edge detection task. However, it misses edges from usual image edge detection algorithms and requires important latency to compute. Nevertheless, this work introduces a novel OHN layered architecture that can qualify ONN to explore novel applications. The proposed architecture is modular; thus, it can be adapted to include more than two OHN layers, with modularity in input and output data. Furthermore, we only consider an analog MG activation function. However, we can replace the FF-MG with other activations. For example, in [224, 116], authors have shown MG functions to construct different operations such as AND, OR, and XOR. For the moment, we applied our architecture to image edge detection, as commonly explored in the state-of-the-art [143], but applying FF-MG layers is in principle compatible with numerous applications requiring at least two OHN layers. It requires further investigations to train such a system for a specific task. Finally, as already mentioned, scaling is a critical aspect of OHNs, due to the important number of synaptic couplings. In the analog FF-MG layer, scalability will be limited if the number of MG inputs increases. However, if the FF-MG architecture can reduce the OHN size by dividing it into sub-problems, it should be more scalable than single-layer analog OHNs.

4.7 Discussion and conclusion

OHN can perform pattern recognition but is hardly scalable due to the important number of synapses, and has limited precision even after investigating various learning methods. This chapter proposed a novel ONN architecture to go beyond OHN. Usual ANN models are built with multi-layer architectures, to achieve high precision using efficient learning algorithms. Creating layers of oscillators can be challenging, therefore, as a first step, we introduced a novel ONN architecture with feed-forward cascaded OHNs implemented with digital and analog circuits.

First, we proposed to layer two digital OHNs feed-forwardly by cascading them. We used two cascaded OHNs to perform a robotic edge application with obstacle avoidance. We implemented it in different robots considering first a fully digital system implemented on FPGA, and then a SoC system using both a processing system and programmable logic resources implemented on a Zynq. With both systems, we were able to run the obstacle avoidance application in real-time. Also, we combined the obstacle avoidance system with the previous on-chip learning architecture from chapter 3 to showcase the OHN on-chip learning performances on a real application.

Cascading two digital OHNs feed-forwardly is quite simple, while it can be challenging in analog. As ONN is by default an analog computing paradigm, we also propose a solution to cascade two analog OHNs using an additional analog feed-forward majority gate layer. Using two cascaded analog OHNs, we were able to replace convolution filters to detect edges in images, even though precision and latency performances are substandard.

The two solutions, analog and digital, to cascade OHNs are efficient in the proposed applications, and we expect them to also be efficient for other AI tasks. However, it needs to adapt

complex tasks to multi-level pattern recognition tasks, anticipating each OHN-level training pattern to use unsupervised learning rules. Thus, it is not straightforward to apply to different AI problems. A possible solution is to explore novel learning algorithms adapted to cascaded OHN architecture. For example, the equilibrium propagation, introduced for OHN in Chapter 3, could be adapted to cascaded OHNs, but it requires further investigation. Also, we could consider parallel multi-OHN architecture, instead of cascaded, in order to divide a large pattern recognition problem into sub-problems, reducing the OHN size, but keeping the pattern recognition applications. We did not investigate deeply this solution, but it could help increase the OHN scalability for pattern recognition.

Finally, in order to go beyond OHN, we believe it is important to study multi-layer ONN architectures closer to conventional multi-layer ANN models and adapted learning algorithms. We discuss multi-layer ONN architectures in Chapter 5.

The work described in this Chapter resulted in three conference papers [216, 198, 225] and a conference talk [199].

CHAPTER 5

MULTI-LAYER ONN FOR HETERO-ASSOCIATION AND CLASSIFICATION

Contents

5.1 Introduction

In the previous chapters, we introduced a digital ONN implementation on FPGA. We first validated the ONN on FPGA with the state-of-the-art fully connected architecture configured for pattern recognition, thus considering an OHN. The validation and test of the digital OHN showcased its limitation for large-scale edge applications. Chapter 3 shows that using state-of-the-art unsupervised learning to configure the digital OHN limits capacity and does not permit it to excel against other ANN models. We explored alternative learning solutions, starting with unsupervised learning rules implemented off-chip and on-chip for pattern recognition, before proposing a solution with the supervised equilibrium propagation (EP) algorithm to experiment OHN for image classification. However, in both cases, the OHN is not competitive compared with typical multi-layer ANN models. Additionally, the fully connected architecture limits the scale of OHN due to the drastic increase of synaptic connections when the number of neurons increases. Thus, there is a need to explore novel ONN architectures to go beyond OHN.

Usual ANN models are often assembled in a multi-layer architecture to achieve high accuracy as multi-layering creates non-linearity between the input and the output. Creating layers of coupled oscillators can be challenging because coupling is recurrent by default. In Chapter 4, we proposed to build feed-forward layers of OHNs by cascading small-scale OHNs. We used the cascaded OHN architecture on robotics applications, as well as image processing. For both robotics and image processing, the task is divided into sub-pattern recognition tasks to configure each OHN with unsupervised learning. Thus, it is not straightforward to learn and

perform a wide variety of tasks with the cascaded OHN architecture. A solution could study novel learning algorithms compatible with the cascaded OHN architecture [226]. However, we believe it might not enlarge the ONN scope of applications as it mainly considers dividing a typical auto-associative memory (AAM) task into small-scale AAM sub-tasks. Furthermore, analog implementations often require additional circuits to build the feed-forward path between two cascaded OHNs, waiting for the first OHN to stabilize before initializing the second one.

To go beyond OHN and cascaded OHN, we explore alternative multi-layer architectures, considering layers of non-coupled oscillators with coupling between layers, see Figure 5.1. As mentioned, by default, coupling two oscillators creates a bidirectional connection between them, being feed-forward and recurrent at the same time. Thus, as a first step, we explore creating layers of uncoupled oscillators with bidirectional connections between layers. However, with the digital ONN design, we can differentiate both feed-forward and recurrent paths between coupled oscillators. So, later, we modified the digital design to build a multi-layer feed-forward ONN. More than that, we develop methods to apply the multi-layer ONNs to image processing, from image edge detection, to feature extraction, as well as classification tasks.

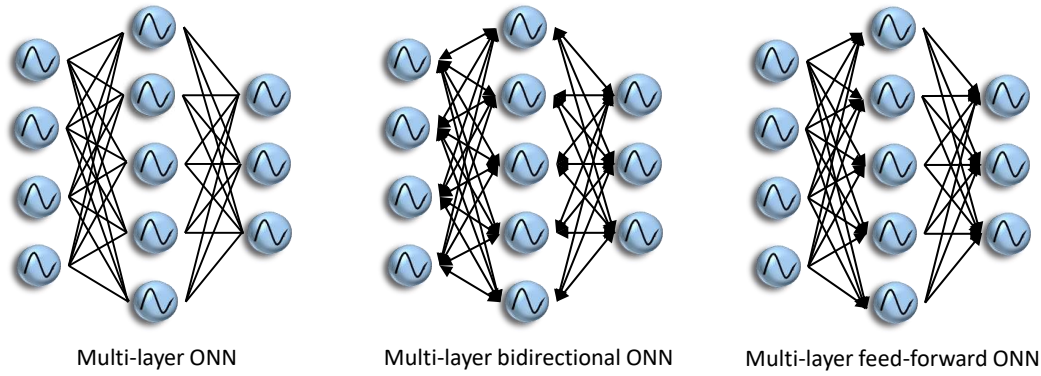


FIGURE 5.1 – *Multi-layer ONN architectures.*

This Chapter starts by presenting the two multi-layer architectures, how they perform inference and learning, and their digital implementation on FPGA. Then, it describes how a 2-layer ONN can be adapted to image edge detection. We also integrate image edge detection with ONN in a more complex system to improve feature extraction latency performances. Finally, we simulate a 3-layer feed-forward ONN applied for classification considering ANN supervised learning solutions.

5.2 Multi-layer architectures

In this thesis, we consider a multi-layer architecture as a network architecture organizing neurons in different layers.

In state-of-the-art, we identified only a few contributions to the study of multi-layer ONN. Authors in [152, 153] proposed to organize neurons in two bidirectionally connected layers. An input layer inside which neurons are not coupled, and a core layer with a fully connected architecture between neurons. Using this architecture in a frequency-computing system, they are able to perform pattern recognition and classification tasks. Alternatively, authors in [227] introduced a neural network containing pairs of frequency-computing oscillators as neurons. Combining these oscillator-based neurons in a multi-layer feed-forward architecture achieved

high accuracy in image classification using back-propagation training. Note, in this case, training configures weights between neurons, not the coupling between a pair of oscillators. More recently, Velichko [151] suggested a frequency-computing 2-layer ONN with an input layer organized with an array topology, connected feed-forwardly to a single output neuron. They highlight that using high-order synchronization enables multi-sate outputs to perform pattern recognition. Finally, considering phase-based computing, Avedillo [228] highlighted that ONN in a feed-forward topology can replace a majority gate using Phase encoded Logic, Shamsi [229] proposed an STDP-based learning rule compatible with feed-forward phase-computing ONNs, and Rudner [202] simulated multi-layer ONNs trained with back-propagation through time (BPTT). To our best knowledge, there are no other works focusing on multi-layer ONN architectures.

Here, we focus on multi-layer ONN architectures, considering feed-forward of bidirectional connections between layers. Note, that we focus on the case where there is no additional coupling between oscillators in the same layer. This Section details the two multi-layer architectures and their implementation with the digital ONN from Chapter 2.

5.2.1 Multi-layer bidirectional ONN

We define a multi-layer bidirectional architecture as a network organizing neurons in multiple layers, at least two, inside which neurons are unconnected but between which neurons are connected bidirectionally. Let's suppose a network with an input layer I of with states S_I, a hidden layer H with states S_H, and an output layer O with states S_O. Layers I and H are connected through bidirectional synapses $W_{IH} = W_{HI}$ and layers H and O are connected through bidirectional synapses $W_{HO} = W_{OH}$ such that for an initial input state S_{IN} the update of each layer follows:

$$\begin{cases} S_I(t+1) = f(W_{HI}S_H(t) + S_{IN}) \\ S_H(t+1) = f(W_{IH}S_I(t) + W_{OH}S_O(t)) \\ S_O(t+1) = f(W_{HO}S_H(t)) \end{cases} \tag{5.1}$$

synchronously until stabilization, with f, the activation function of the neurons.

Before considering oscillators as neurons, multi-layer bidirectional networks were proposed with the so-called bidirectional associative memory (BAM) [208]. BAM networks were introduced to perform heterogeneous associative memory (HAM) [207], associating in-out pairs of data with equal or different dimensions. In BAM, the update of neuron states is sequential, starting with neurons of the input layer, then the hidden layers one by one, before the output layer, and reversely until stabilization. Training a two-layer BAM can simply adapt the Hebbian learning to learn pairs of patterns instead of a single pattern. For larger layers, Kosko [230] also introduced a back-propagation algorithm compatible with BAM, enabling efficient image recall. However, the sequential layer update in BAM is different from the synchronous parallel update of ONN. Alternatively, [58] proposed the energy-based EP learning algorithm and applied it to a multi-layer bidirectional architecture of continuous Hopfield neurons. We expect EP to be compatible with multi-layer bidirectional ONNs and it also motivates our study on this novel architecture.

The synchronous parallel update of the multi-layer bidirectional ONN is both advantageous and restrictive. While parallel oscillation synchronization normally helps for fast infe-

rence, especially for dynamical systems, it also generates an impact of the hidden and output layers over the input layer. If hidden and output layers are not carefully initialized, it can importantly impact the network synchronization latency and precision. Also, apart from EP, to our knowledge, there is no straightforward and efficient learning algorithm adapted for multi-layer bidirectional ONN. Thus, we also study the alternative multi-layer feed-forward ONN.

5.2.2 Multi-layer feed-forward ONN

We define a multi-layer feed-forward architecture as a network organizing neurons in multiple layers, at least two, inside which neurons are unconnected but between which neurons are connected feed-forwardly. Let's suppose a network with an input layer I with states S_I, a hidden layer H with states S_H, and an output layer O with states S_O. Layers I and H are connected through feed-forward synapses $W_{IH} = W_{HI}$ and layers H and O are connected through feed-forward synapses $W_{HO} = W_{OH}$ such that for an initial input state S_{IN} the update of each layer follows:

$$\begin{cases} S_I(t+1) = S_{IN} \\ S_H(t+1) = f(W_{IH} S_I(t)) \\ S_O(t+1) = f(W_{HO} S_H(t)) \end{cases} \tag{5.2}$$

synchronously until stabilization of the output layer, with f, the activation function of the neurons. Note, that the input is constant and the signal propagates only in one direction. Thus, the latency of computation mainly depends on the number of layers and how fast the signal propagates from one layer to the next. Also note, that with HNN and ONN activation functions, a *sign* function is applied on the weighted sum, such that if the weighted sum is equal to zero, the neuron state is not updated and keeps its previous state. Thus, the initialization of hidden and output layers also plays a role in the feed-forward architecture.

Feed-forward network architectures have been widely studied and are still standard nowadays with gradient-base supervised learning to solve complex classification tasks [9, 231]. However, typical oscillator designs cannot differentiate input and output signals, making a coupling bidirectional by default. To overcome this issue, it is possible to include additional circuits between two oscillating layers, as we did in Chapter 4 to connect two analog oscillators feed-forwardly, for example with a majority gate additional circuit [225]. Also, recent analog, digital, and mixed-signal ONNs were designed with the capability to separate inputs and outputs [140, 183]. In particular, the digital design from Chapter 2 has differentiable in-out ports to implement feed-forward architectures. We found only a few contributions in the literature using differentiable in-out oscillators to create feed-forward ONN architectures. First, Velichko [151] proposed a 2-layer ONN with feed-forward coupling between layers and additional neighbor bidirectional coupling in between the input layer. Using frequency-computing and simulated annealing for training, they were able to perform pattern recognition. Then, Avedillo [228] suggested to create phase encoded logic with phase-computing oscillators to perform logic functions, such as a majority gate. Also, Shamsi [229] recently introduced a novel STDP-inspired supervised or unsupervised learning for phase-computing feed-forward ONN. Finally, Delacour [183] and Graber [232] designed analog ONNs with differentiable in-out ports to solve combinatorial optimization problem (COP).

Yet, we considered mainly binary output oscillator phases stabilizing in two 180^o-opposite phases. Thus, Hebbian learning can also be used to train a 2-layer feed-forward ONN for HAM

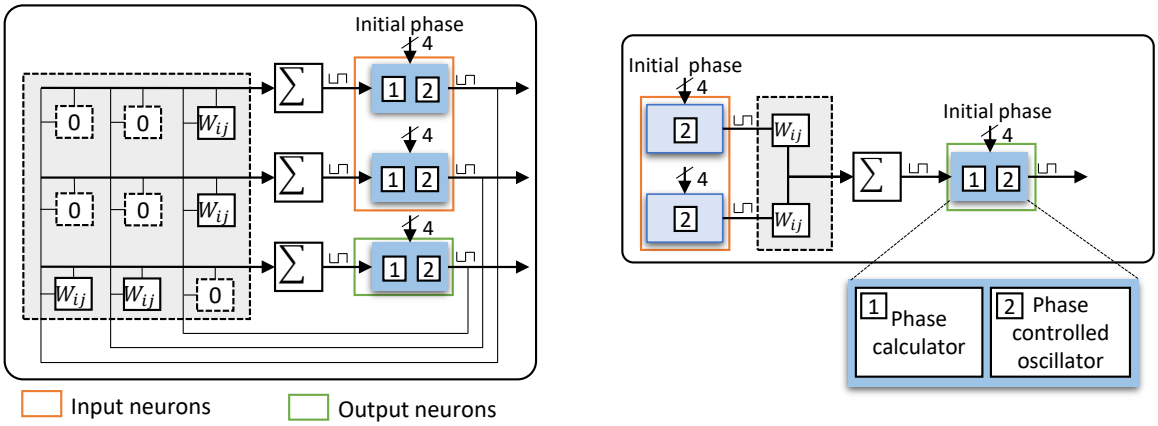
or classification tasks. Otherwise, for more than two layers, usual supervised gradient-based learning could be considered, approximating the neuron activation function to be differentiable, like with SNNs.

5.2.3 Implementation

To fit with the novel architectures, we need to adapt the digital design from Chapter 2. We start by implementing two ONN layers for each bidirectional and feed-forward architecture.

In the digital design, each oscillating neuron is made of a phase calculator to integrate post-synaptic input signals and define a novel phase, and a phase-controlled oscillator to apply the novel phase to the neuron output oscillating signal.

In a bidirectional architecture, neurons in input, hidden, and output layers require both the phase calculator and the phase-controlled oscillator to integrate the neuron input phase and update the neuron output phase. Thus, for a two-layer bidirectional architecture, we reuse the fully-connected ONN design and we constrain synaptic weights between neurons of the same layer to be zero, representing no coupling, see Figure 5.2a. To increase the number of layers, we can also consider a fully connected ONN arranging weights to not couple neurons in the same layer but it can be challenging and it will utilize more digital resources than necessary, or we need to modify the digital design to divide the weight matrix and organize oscillators in layers.



(a) *2-layer bidirectional ONN.* **(b)** *2-layer feed-forward ONN.*

FIGURE 5.2 – *2-layer digital ONNs with 2 input neurons and 1 output neuron.*

In comparison, in a feed-forward architecture, the input layer receives an input phase and generates an oscillation aligned in this phase, so a neuron from the input layer only contains a phase-controlled oscillator. Also, we arrange neurons in layers and we modify the weight matrix to create different matrices for the different synaptic layers, see Figure 5.2b.

For both bidirectional and feed-forward ONNs, we keep the 5-bit synapse representation for each weight, as well as the 16-phase stages of the phase-controlled oscillators. Note, that we modified the design to improve latency performances, performing parallel initialization of oscillators' phases instead of sequential initialization. We simulate the multi-layer ONN architectures using the XC7Z020-1CLG400C FPGA as the target device and implement them in the Zybo-Z7 development board equipped with the XC7Z020-1CLG400C FPGA.

Finally, in order to easily validate ONN architectures and their configuration for image

edge detection, we first test our methods using an HNN emulator developed on Matlab for each architecture. The HNN Matlab emulators model the two ONN architectures using bipolar spins with *sign* activation functions as neurons.

5.3 2-layer ONN for image edge detection

Image edge detection detects and extracts contrast in images. Typically, it uses small-size convolution filters to scan images and detect brightness and color changes. State-of-the-art Sobel [223], and Canny [222] algorithms use small convolution kernels (commonly 3x3, 5x5, and 7x7) to scan the image and detect edges in the different areas of the image. For example, Sobel commonly uses two 3x3 convolution kernels associated with vertical and horizontal edges. Scanning the image applies the convolution kernels' parameters on windows of the image sequentially and the convolution result is used to calculate a global gradient for the central pixel of the window. The gradient, which can be binarized with a threshold, indicates the detection of an edge, see Figure 5.3. 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.

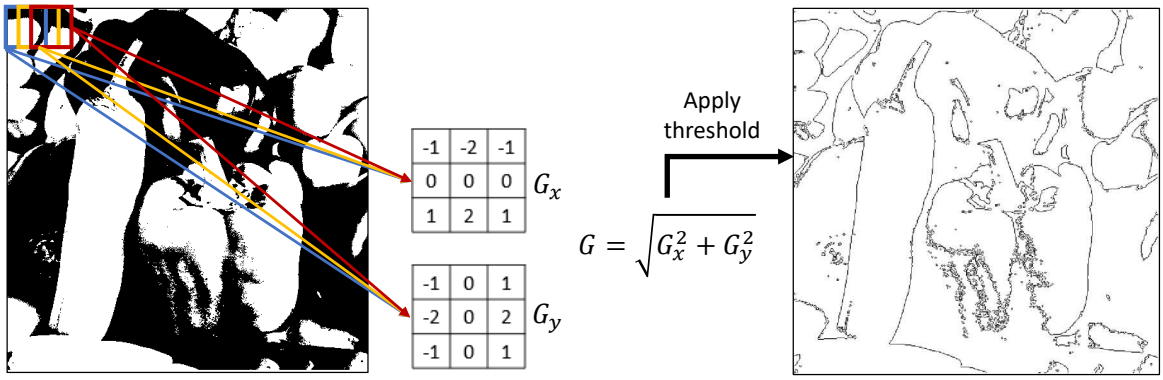


FIGURE 5.3 – *Sobel image edge detection algorithm.*

A convolution operation resembles the activation of the output layer in a 2-layer ONN, motivating the exploration of how to replace convolution kernels with the 2-layer ONN architectures. We studied both bidirectional and feed-forward architectures and were able to replace Sobel convolution filters with ONN to perform efficient image edge detection.

5.3.1 Evaluation of the image edge detection application

We develop two efficient solutions to perform image edge detection first with a 2-layer bidirectional ONN before adapting it to a 2-layer feed-forward ONN. In both cases, we focus on replacing the usual convolution kernels with a 2-layer ONN. To do so, we consider first a 3x3 ONN input layer that scans the image, using 3x3 windows of the image as input of the ONN, before generalizing it to larger input data with 5x5 or 7x7 pixels. We first validate the methods on gray-scale images and evaluate precision performances compared to state-of-the-art algorithms.

Evaluation of image edge detection algorithms is not an easy task as, to the best of our knowledge, there is no state-of-the-art evaluation metric, as well as no ground truth (GT) avai-

lable. To first visually evaluate the edge detection solutions with the Matlab emulator, we use a method from the literature [233] which uses hexagonal forms with various gray levels on a white background, see Figure 5.4. It first assesses if the solution is able to retrieve simple edges from a basic image.

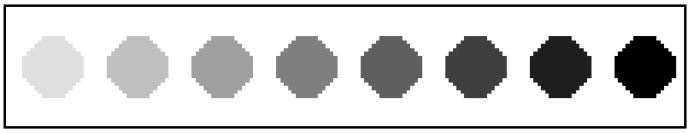


FIGURE 5.4 – *Test image for image edge detection validation using hexagonal forms with various gray levels on a white background.*

However, a more precise assessment of the performance is necessary to evaluate the ONN image edge detection solutions and to compare them with state-of-the-art algorithms. In literature, similarity coefficients are often used to assess the precision of object detection algorithms. Here, we propose to use the Jaccard similarity coefficient (JSC) [234] to evaluate our solutions. JSC metric takes a GT and calculates the similarity between the GT and the obtained output information making a ratio between overlapping and union areas. To apply it to edge detection algorithms, we consider the overlapping area as the number of overlapping edges, and the union area as 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:

$$JSC = \frac{Overlap}{Union} \tag{5.3}$$

As already mentioned, to the best of our knowledge, there is no benchmarking data set with GT for edge detection algorithms. 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 ONN solutions using the JSC metric. We generate Sobel and Canny outputs using built-in Matlab functions.

5.3.2 2-layer bidirectional ONN

As a first development step, we adopt a 2-layer bidirectional ONN to perform image edge detection by using it as a 3x3 filter. We define the ONN input layer with a 3x3 image window and the ONN output layer with the detected edge.

However, as mentioned in Section 5.2.1, in ONNs, all coupled oscillators interact simultaneously. Thus, if only neurons of the input layer are initialized, neurons of the output layer will also impact the input layer due to coupling. A solution to counter this effect is to initialize output oscillators with neutral values, such that they do not influence the dynamic. Presently, we only use bipolar values, $\{-1, +1\}$ with Hopfield neurons or $\{0^o, 180^o\}$ with oscillators. Consequently, to avoid the influence of one of the two values, we consider two neurons for each output information initialized with different values, one with $\{-1\}$ or $\{0^o\}$ and the other with $\{+1\}$ or $\{180^o\}$. Hence, one output has the same influence for each possible value. Note, that we interconnect output neurons to let them interact, as they represent the same information.

The first solution developed with the bidirectional ONN configures the ONN synaptic

weights using the Sobel kernels. Especially, it associates the Sobel coefficients with output neuron combinations to create training patterns, so we can use the usual Hebbian learning algorithm, see Figure 5.5. One pattern associates the horizontal Sobel kernel with white pixels output corresponding to {-1, -1} for HNN, or {0^o,0^o} for ONN, and the second pattern associates the vertical Sobel kernel with black pixels output corresponding to {+1, +1} for HNN, or {180^o,180^o} for ONN. A no-edge case is detected when the ONN stabilizes to none of the trained output patterns. Note, that 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. Also note, that the learning algorithm uses patterns with doubled output information.

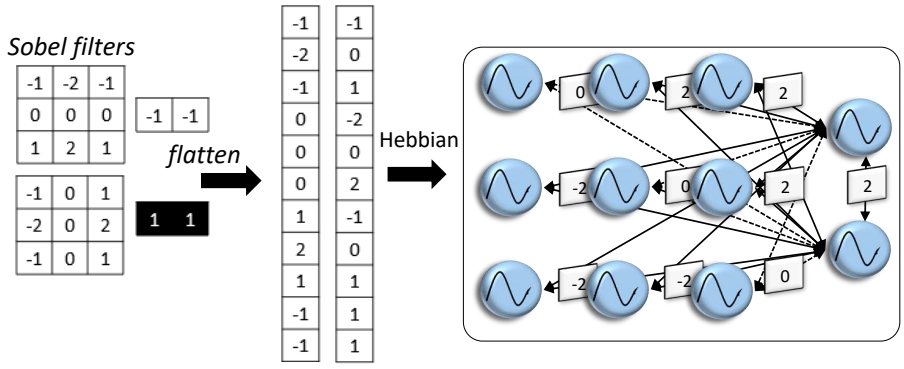


FIGURE 5.5 – *First 2-layer bidirectional ONN configuration for image edge detection.*

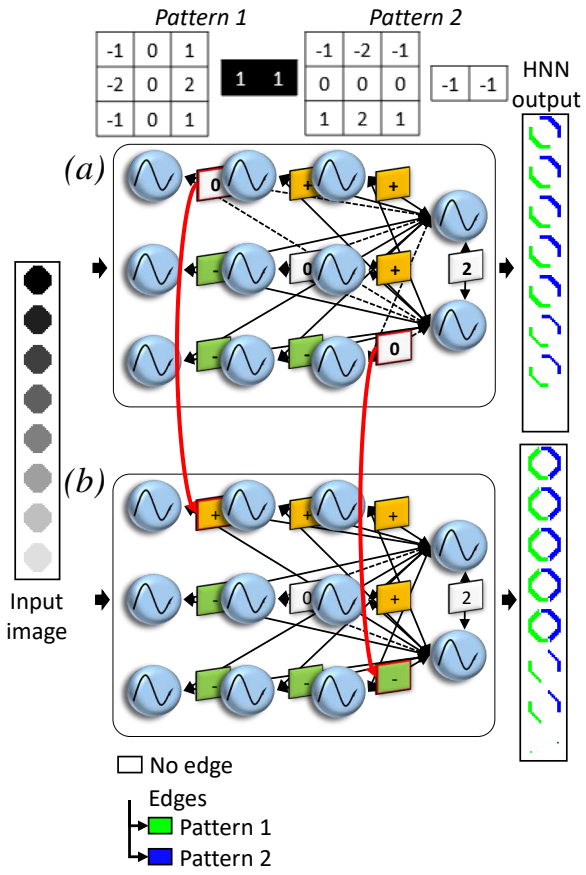


FIGURE 5.6 – *Evolution of the 2-layer bidirectional ONN configuration for image edge detection, from (a) to (b).*

Using this simple method, we obtain encouraging results meanwhile the ONN was mis-

sing some edges, as shown in Figure 5.6.a). Thus, we customize the synaptic weights to 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 the top right input neuron to output neurons are $\{-2\}$, while weights from the bottom left 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 edge, 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, see Figure 5.6.b).

5.3.3 2-layer feed-forward ONN

We expect the bidirectional layered ONN to be slower to compute than a feed-forward layered ONN due to the parallel computation between input and output layers. Thus, we adapt the previous ONN image edge detection method to a feed-forward ONN architecture.

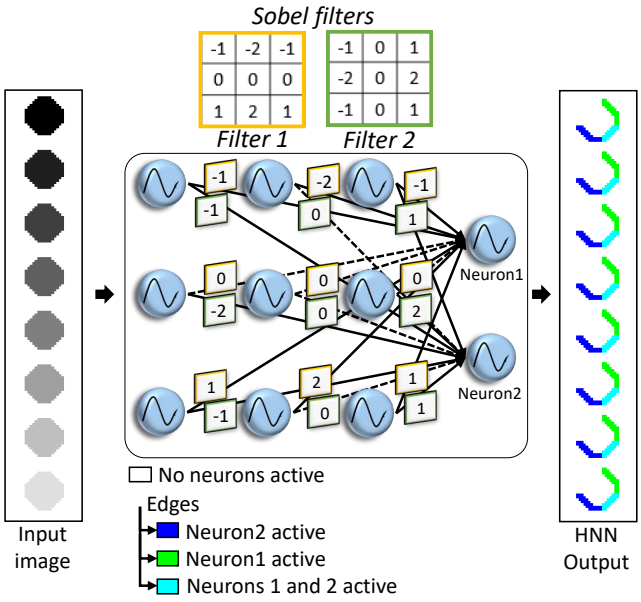


FIGURE 5.7 – *Configuration of the feed-forward ONN with 3x3 input size, using Sobel kernel coefficients as weights, and results of the Matlab emulator on the gray-scale octagon map.*

In the mathematical concept, the weighted sum computed before the neuron activation function is equivalent to a convolution operation, as used in classical image edge detection algorithms. Thus, our first approach to use the feed-forward ONN for edge detection consisted of applying the Sobel kernel coefficients as weights between the 3x3 input neurons and two output neurons, one for each Sobel kernel, as shown in Figure 5.7. In this case, output neurons are initialized with $\{-1\}$ for HNN or $\{0^o\}$ phase for ONN that we associate with no-edge information. Then, we expect the feed-forward ONN to change the output information to $\{+1\}$ for HNN, or $\{180^o\}$ phase for ONN if an edge is detected. However, Figure 5.7 highlights that not all edges are correctly retrieved. It can be explained by the difference between the Sobel gradient calculation and the Hopfield or ONN activation functions. In Sobel, a large negative intensity and a large positive intensity of gradient will correspond to an edge. However, with

Hopfield or ONN, if the weighted sum is negative, the states will evolve to $\{-1\}$ for HNN, and $\{0^o\}$ phase for ONN, corresponding to 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 changes 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.

Thus, we studied alternative solutions to use our feed-forward ONN for edge detection. First, we tried to reproduce a configuration similar to the bidirectional architecture. In this case, 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 $\{0^o,0^o\}$ ONN phase states. We set weights from input neurons corresponding to the opposite side of each edge with opposite weight values, as displayed in Figure 5.8(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 value from the previous configuration, so positive weights become negative, and negative weights become positive, see Figure 5.8(b). In this case, we only detect edges that were missing in the previous configuration. Thus, by using one of the described configurations to connect one output neuron, and the other configuration to connect the second output neuron, the feed-forward ONN is able to retrieve all edges of the gray-scale octagon map, see Figure 5.8(c).

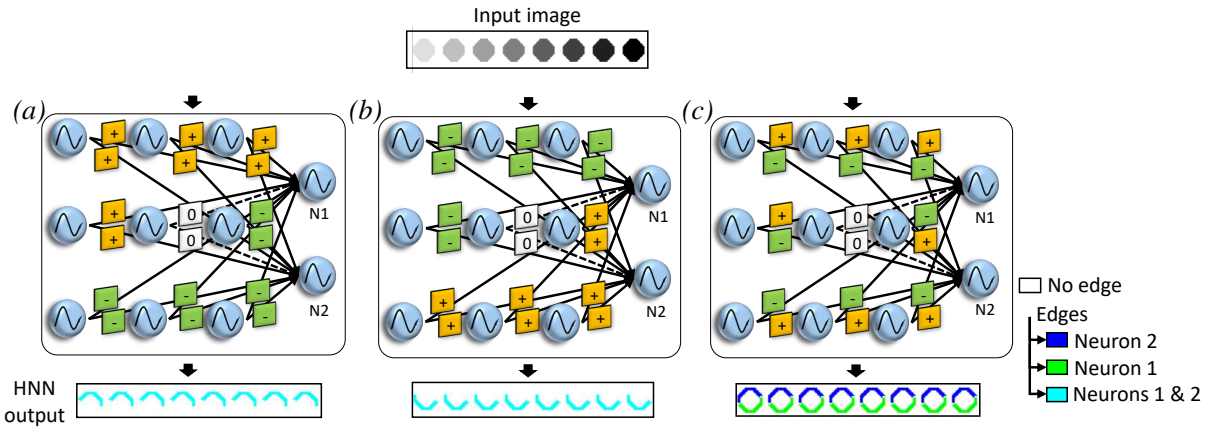


FIGURE 5.8 – *Feed-forward ONN configuration with 3x3 input size, with (a) first option tried from the bidirectional ONN configuration, (b) second option by reversing the weights of the first option, and (c) final option which combines (a) and (b). Also, the results of the Matlab emulator on the gray-scale octagon map are presented for each configuration.*

5.3.4 Extension to 5x5 and 7x7 input kernels

Convolution filters configured for image edge detection are often 3x3 kernels but can also be larger, with 5x5 kernels, or even 7x7 kernels. Increasing the filter size can detect edges in larger windows, allowing a larger stride for scanning depending on the application, and reducing the scanning latency. Thus, we also study if the image edge detection methods with 2-layer bidirectional and feed-forward ONNs can scale for larger input sizes. In particular, we extend the methods to perform image edge detection with ONN filters using 5x5 and 7x7 input kernels, see Figure 5.9. 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 correspond to the same edge, they have opposite values. For the feed-forward ONN, an additional rule is necessary: weights from the same input neuron to the two output

neurons, need to have opposite values, see Figure 5.9. When we apply the 5x5 or 7x7 filters, we use the same method as with 3x3, we select a small part of the image (5x5 or 7x7), we apply the ONN for edge detection (bidirectional or feed-forward architecture), and we apply the output information to the central pixel. Then, we move the filter with a 1-pixel stride to select the next part of the image, and so on, see Figure 5.9. Note, that using larger filters detects multiple times each edge as the scanning window is larger.

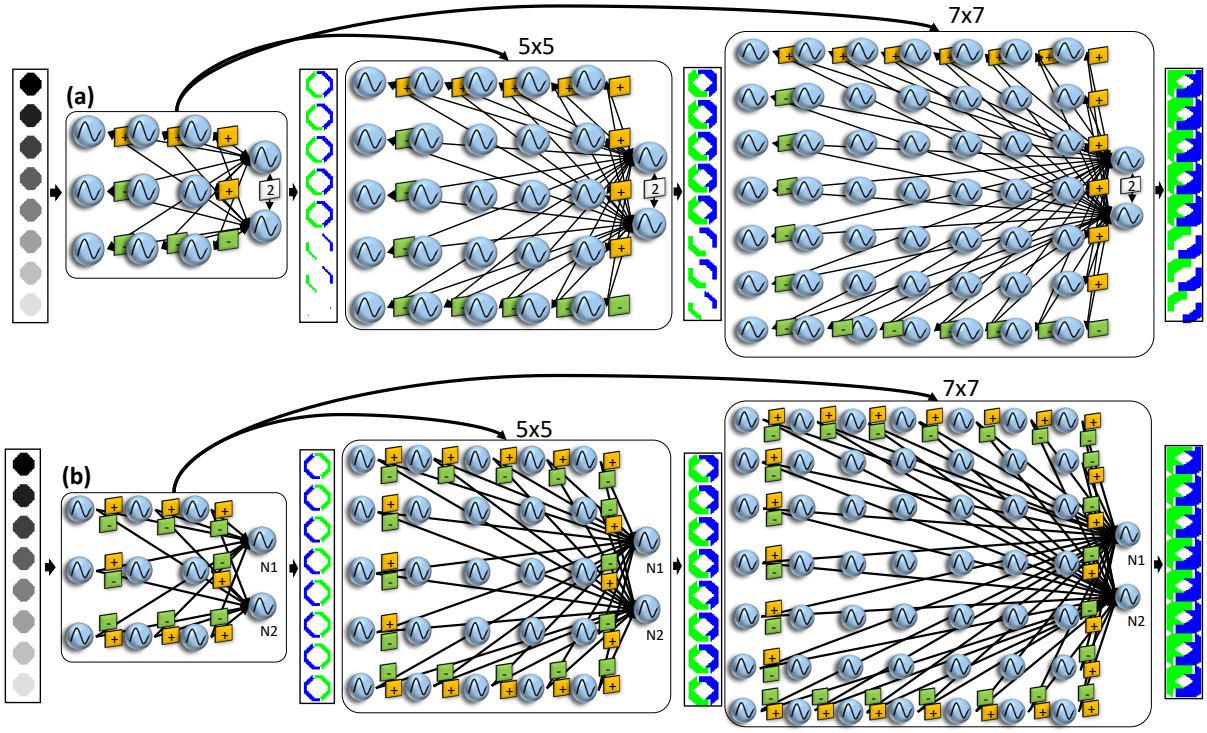


FIGURE 5.9 – *ONN configuration with 3x3, 5x5, and 7x7 input size, for (a) the bidirectional ONN architecture, and (b) the feed-forward ONN architecture. Also, the results of the Matlab emulator on the gray-scale octagon map are presented for each configuration.*

5.3.5 Results and benchmarking

After exploring and validating methods to perform image edge detection using the 2-layer ONNs with the Matlab emulator, we validate and assess the performances of the methods with the digital ONN design. We test both architectures on black-and-white 28x28 MNIST images, and gray-scale 28x28 MNIST images, as well as on large-scale black-and-white images. For each architecture, we perform image edge detection using 3x3, 5x5, and 7x7 input filters, we evaluate the precision and compare them with state-of-the-art algorithms. Finally, we extract resource utilization and latency to assess the real-time performances of the multi-layer ONN architectures compared to the previous analog cascaded OHNs configured for image edge detection and to other image edge detection implementations.

Black and White 28x28 MNIST Images

The MNIST database [9] contains 28x28 gray-scale images that we binarize to obtain black and white images. We simulate the two architectures with their respective digital designs performing a sequential scanning on black-and-white MNIST images, see Figure 5.10. It highlights that both 3x3 bidirectional and 3x3 feed-forward architectures have equal results. Compared 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 feed-forward architectures, edges are detected even more times as we scan the image with a 1-pixel stride. This creates large edge shapes, making it hard to visualize edges on a small 28x28 image. Figure 5.11 corroborates this visual assessment by comparing numerically the feed-forward and bidirectional ONN filters with the Sobel filter considering Canny as GT. For black and white MNIST images, the number of overlapping edges between our solutions and Canny is close to the number of overlapping edges between Sobel and Canny. However, the number of union edges is larger for the ONN solutions, increasing with the filter size. It confirms that ONNs detect more edges than Sobel, which slightly impacts negatively the JSC.

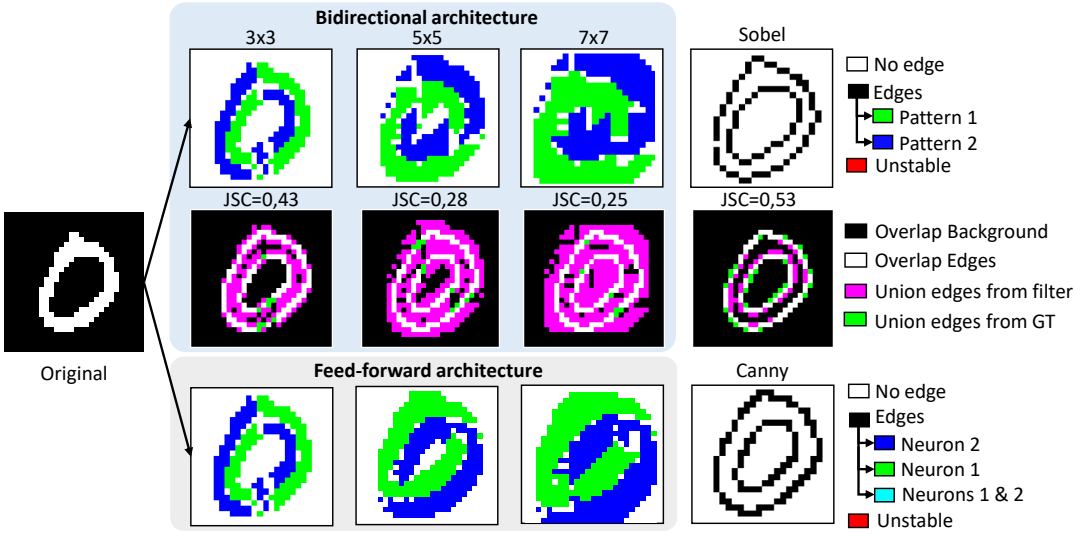


FIGURE 5.10 – *Edge detection results of the digital ONN designs for the bidirectional architecture with input size 3x3, 5x5, and 7x7, and for the feed-forward architecture with input size 3x3, 5x5, and 7x7, and state of the art Sobel and Canny on a black and white 28x28 MNIST image.*

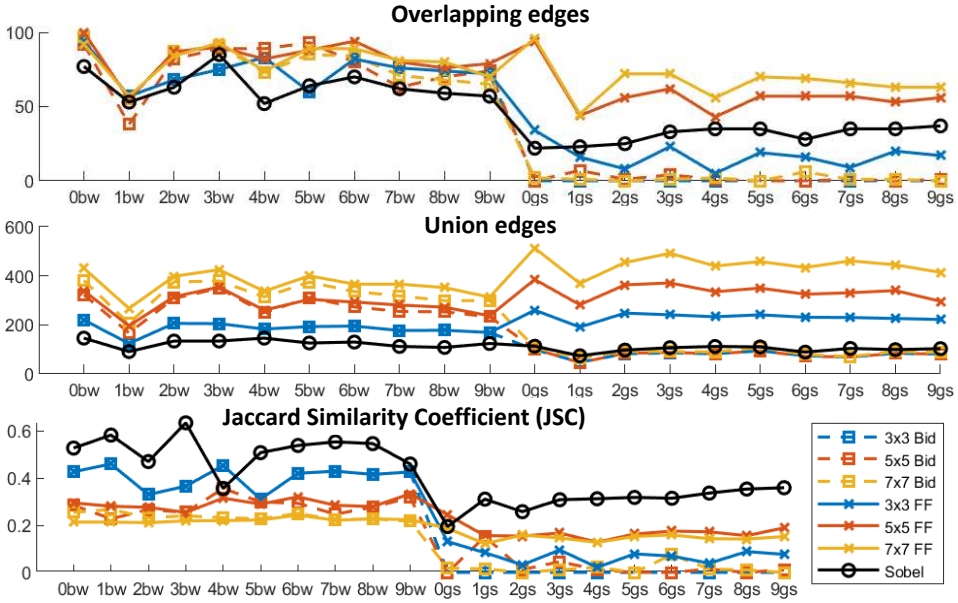


FIGURE 5.11 – *Number of overlapping edges, union edges, and similarity coefficient between the digital ONN solutions and Canny reference, and between Sobel state-of-the-art and Canny reference on MNIST black and white (bw) and grayscale (gs) digit images.*

Gray Scale 28x28 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 5.12 shows ONN output on a 28x28 gray-scale MNIST image for the various architecture configurations. It highlights that the bidirectional architecture is not able to perform image edge detection on realistic gray-scale images. The Matlab emulator was able to detect edges on a gray-scale octagon map, however, with the digital ONN design, most of the inference cycles output unstable states, never reaching stabilization. The feed-forward ONN architecture correctly detects edges on gray-scale images. In comparison to state-of-the-art Sobel and Canny, visually the feed-forward ONN seems to perform better than Sobel, but worse than Canny. However, when computing the JSC, Sobel gets better results than the 3x3 ONN filter. Figure 5.12 showcases that the feed-forward ONN detects more edges than Sobel, however, edges do not overlap 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 behavior increases the JSC parameter as the number of overlapping edges increases with the number of detected edges, however, visually, large-scale ONN filters do not seem relevant for small-size images.

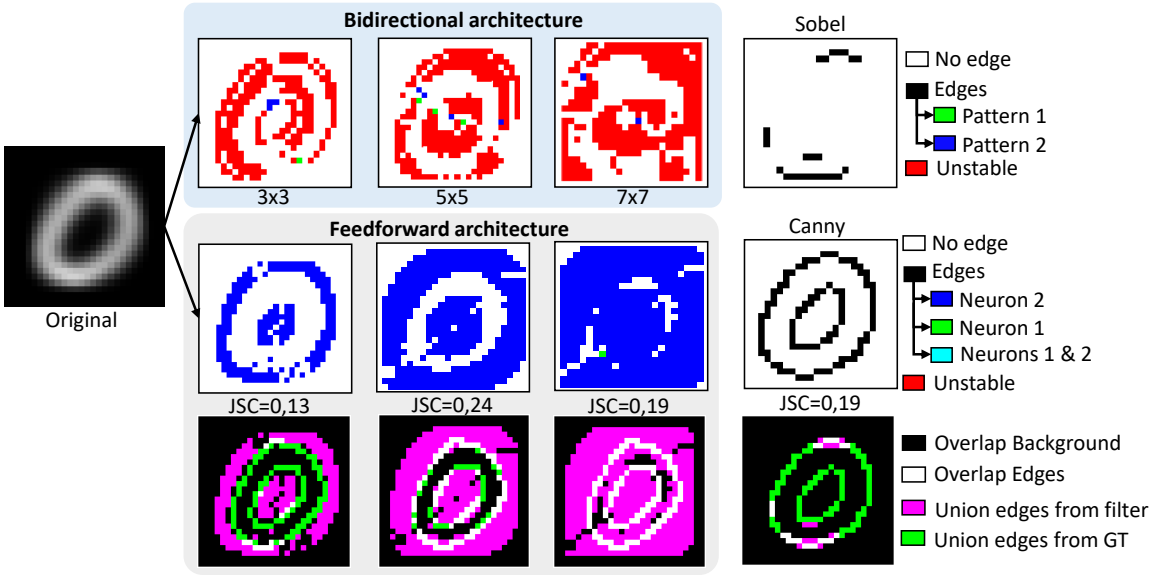


FIGURE 5.12 – *Edge detection results of the digital ONN designs for the feed-forward architecture with input size 3x3, 5x5, and 7x7, and for the feed-forward architecture with input sizes 3x3, 5x5, and 7x7, compared with state-of-the-art Sobel and Canny on a gray scale 28x28 MNIST image.*

512x512 Standard Black and White Images

Finally, we test the two ONN architectures configured for image edge detection on 512x512 large-scale standard black-and-white images. The digital ONN simulation process takes time, and simulating 512x512 images is equivalent to the simulation of more than 260k times a single ONN for each part of the image. So, for the 3x3 filters, we simulate all 3x3 black and white options (512 possible inputs) using the digital design, and we associate each part of the image with the corresponding ONN output. However, with 5x5 ONN input, the number of possibilities increases to more than 33 million, and for 7x7 it is even larger, becoming a challenge to simulate. For both the 5x5 and 7x7 options, we select a small part of the image and apply it as input for the multi-layer ONNs.

Figure 5.13 visually confirms the efficiency of the two ONN solutions with bidirectional and feed-forward architecture to perform image edge detection on black and white images.

Figure 5.14 supports the previous results showing an increase in the number of detected edges when increasing the input layer size. It also corroborates that the ONN edge detection filter is close to the Sobel state-of-the-art algorithm as it detects a similar number of overlapping edges, even if the higher number of union edges, increasing with the filter size, decreases the JSC when comparing similarity with the Canny algorithm. In comparison with the previous analog cascaded OHN solution for image edge detection presented in Chapter4, the digital 2-layer bidirectional and feed-forward ONNs achieve better similarity with Canny, going from a JSC of 0.18 with the analog solution to 0.37 with the digital solution on large-scale black-and-white images.

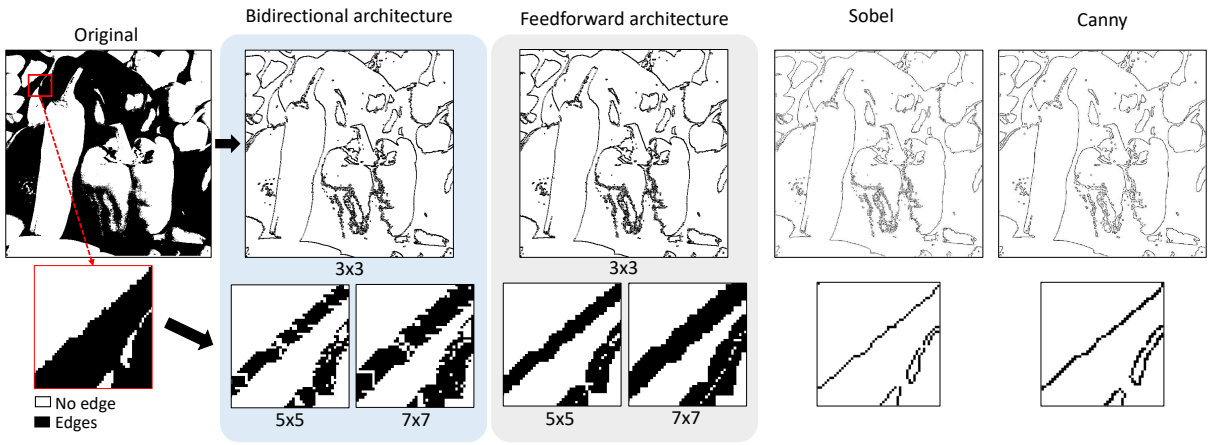


FIGURE 5.13 – *Edge detection results of the digital ONN designs for the bidirectional architecture with input size 3x3, 5x5, and 7x7, and for the feed-forward architecture with input size 3x3, 5x5, and 7x7, and state of the art Sobel and Canny on the black and white 512x512 standard "pepper" image.*

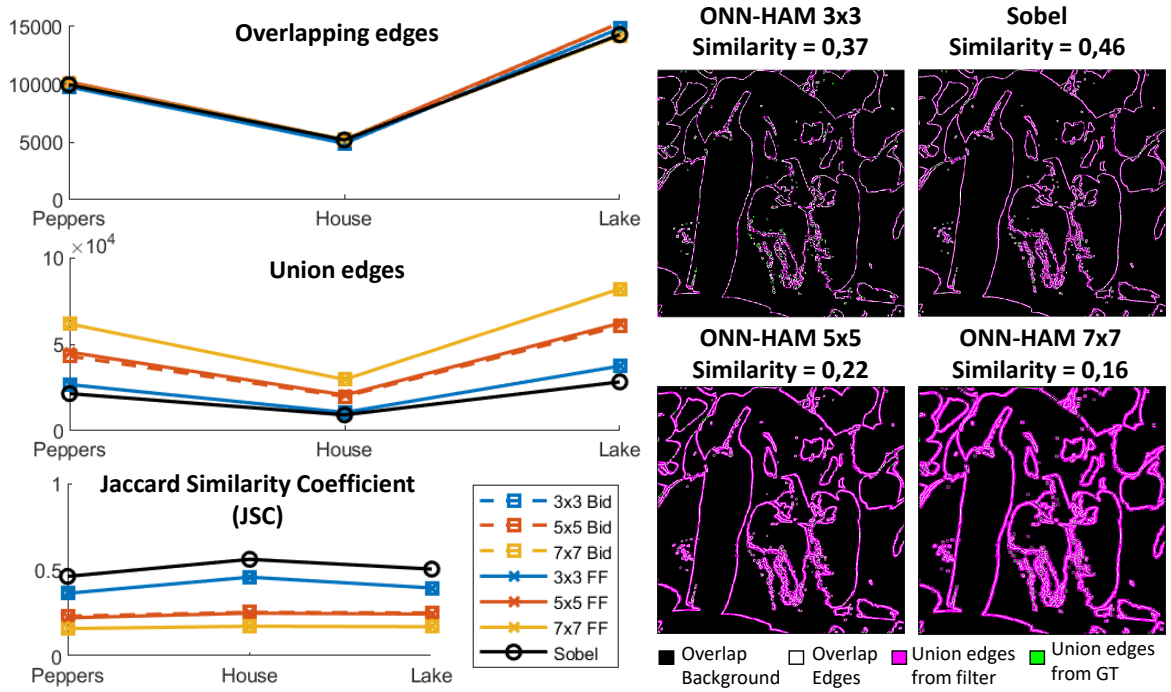


FIGURE 5.14 – *Number of overlapping edges, union edges, and similarity coefficient between the ONN solutions simulated on Matlab, Sobel state-of-the-art and Canny GT reference on large scale black and white images Peppers, House, and Lake. Similarity outputs of the bidirectional ONN solutions simulation on Matlab and Sobel solution compared to Canny on black and white Peppers image.*

ONN performances

After validation of the ONN efficiency to perform image edge detection using simulation, we also extract ONN characteristics from simulations. Table 5.1 presents the latency performances of the different ONN solutions (feed-forward and bidirectional with various input sizes). Each ONN operates with a system frequency set to $F_{sys} = 166MHz$, equivalent to an oscillation frequency of $F_{osc} = 2.7MHz$. It highlights that a 3x3 bidirectional ONN needs $1.42\mu s$ on average to stabilize. Note, that 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 feed-forward ONN designs drastically accelerates the initialization process compared to the original fully-connected ONN design described in Chapter 2. Here, 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 feed-forward architecture than with the bidirectional architecture. The difference is minimal when considering only one ONN computation, but as shown in Table 5.2 for a full image edge detection process, the difference becomes non-negligible for an entire image scanning. For example, considering a 5x5 input size, the feed-forward ONN can treat images up to 170x170 pixels in real-time (considering 30 images per second), while the bidirectional ONN can handle up to 150x150 pixels. In comparison with the previous analog cascaded OHN solution from Chapter 4, the digital 2-layer ONNs compute faster for a similar oscillating frequency, see Table 5.2. Thus, the digital ONN-based image edge detection is more advantageous compared to the analog solution both in terms of accuracy and latency.

TABLE 5.1 – *ONN latency performances and resource utilization, depending on the architecture and the size. The system is configured with 166MHz frequency, allowing 2.7MHz oscillation frequency.*

ONN architecture	Bidirectional			Feed-forward		
ONN input size	3x3	5x5	7x7	3x3	5x5	7x7
Initialisation	24ns	24ns	24ns	24ns	24ns	24ns
Computation	1.42μs	1.42μs	1.42μs	1.15μs	1.15μs	1.15μs
LUTs (53200)	484	1214	2369	211	302	457
Flip-Flops (106400)	458	1026	1626	277	437	597

TABLE 5.2 – *Estimation of full image edge detection using ONN architectures. The estimation multiplies the number of pixels to treat in each image by the time to initialize and compute each ONN design.*

ONN architecture ONN input size	Bidirectional 3x3 / 5x5 / 7x7	Feed-forward 3x3 / 5x5 / 7x7	Analog cascaded OHN 3x3 ($F_{osc} = 3MHz$)
3x3 image	1.42μs	1.15μs	2.34μs
28x28 image	1.11ms	0.78ms	1.83ms
100x100 image	14.2ms	11.5ms	-
128x128 image	23.3ms	18.8ms	-
512x512 image	372ms	301ms	613ms

Table 5.1 also points out that the bidirectional architecture requires more resources than the feed-forward 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 2369 LUTs, and 1626 Flip-Flops for the 7x7 bidirectional ONN. Thus, both ONN architectures can easily be integrated into larger systems with minimal resource availability.

Benchmarking

Benchmarking image edge detection algorithms is challenging and a common GT is needed for comparison. Here, we evaluate the ONN image edge detection algorithms by comparing the outputs with other state-of-the-art edge detection algorithms. In particular, we consider Canny as GT for the JSC evaluation and compare ONN with Sobel. We show that both the bidirectional and feed-forward ONNs can detect most edges detected by Sobel and Canny algorithms on black-and-white images, even if edges are detected multiple times. We also point out that the efficiency of the bidirectional ONN is limited to black-and-white images, like Sobel, while the feed-forward ONN can also address gray-scale images, like Canny.

The latency is a critical point in image edge detection as it is often integrated into more complex systems. The main drawback comes from the sequential scanning necessary to detect edges in the entire image. In the literature, there are various references to FPGA implementation of Sobel and Canny algorithms, with mainly two options considered to accelerate the image scanning process. One is more adapted to Sobel, and the other is more adapted to Canny. In [235, 236, 237], authors propose to simplify the Sobel convolution 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 [235] can process each pixel gradient in a single clock cycle. Using a clock at $50MHz$, they are able to process a 512x512 image in around $5ms$. Using faster frequency, authors in [236] and in [237] process the 512x512 image in more or less $1ms$, see Table 5.3. For Canny, additional parallelization is necessary to achieve short processing. Authors in [238] propose to parallelize the process by blocks. They divide the image into numerous 64x64 non-overlapping blocks and process edge detection inside each block sequentially, meanwhile, the different blocks compute in parallel. In this way, with a system running at $100MHz$, they can process a 512x512 image in less than $1ms$. However, as shown in Table 5.3, it requires much more resources than the previous Sobel implementations.

TABLE 5.3 – *Performances of FPGA implementation of edge detection algorithms from the literature.*

	Filter size	Resources LUTs	Resources Flip-Flops	Hardware	Frequency	Latency 512x512
Sobel [235]	3x3	346	289	Xilinx Spartan 3 XC3S200	50 MHz	$5.25ms$
Sobel [236]	3x3	47	107	Xilinx Spartan 3 XC3S50-5PQ20	204 MHz	$1.28ms$
Canny [238]	3x3	82496	40640	Xilinx Virtex 5 XC5VSX240T	100 MHz	$0.721ms$
Sobel [237]	3x3	0	114	Xilinx Spartan 6 XC6SLX43TQG144	504 MHz	$0.52ms$
ONN-Bid	3x3	484	458	Xilinx Zynq-7000 XC7Z020-1CLG400C	2.7 MHz	$372ms$
ONN-Bid Parallel	3x3	48525	45577	Xilinx Zynq-7000 XC7Z020-1CLG400C	2.7 MHz	$3.7ms$
ONN-FF	3x3	211	277	Xilinx Zynq-7000 XC7Z020-1CLG400C	2.7 MHz	$301ms$
ONN-FF Parallel	3x3	53125	82727	Xilinx Zynq-7000 XC7Z020-1CLG400C	2.7 MHz	$0.52ms$

Table 5.3 shows that the sequential ONN is much slower than the FPGA implementations of 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. However, ONN needs to wait for phase synchronization and the oscillation frequency is much slower than the system frequency of Sobel FPGA implementations. Thus, as Canny, we need to include some parallelization to be competitive in terms of Latency. Additionally, as mentioned in [238], scanning the image with a stride larger than one, which changes

the overlap between two neighboured windows, can accelerate the image edge detection process even if it also impacts the edge detection precision. Table 5.4 shows that using a single ONN to sequentially scan an image takes more than $300ms$, which is faster than the analog cascaded OHN solution from Chapter 4 requiring more than $600ms$ but hardly competitive against Sobel and Canny FPGA implementations. However, it is interesting to note that using parallel ONNs, with overlapping, the bidirectional ONN can process the entire 512x512 image in less than $10ms$, in particular, using 100 3x3 bidirectional ONNs in parallel, we can process the image in $3.7ms$, which is in the same range as Sobel and Canny FPGA implementations. Furthermore, the feed-forward ONN is even faster and requires fewer resources so we can implement more feed-forward ONNs. Thus, using 290 3x3 feed-forward ONNs in parallel, we can scan the 512x512 image in around $1ms$ which is faster than some Sobel FPGA implementations, but slightly slower than the Canny FPGA implementation, see Table 5.3. Combining both non-overlapping options 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 and comparison with state-of-the-art does not provide precise information on the impact of this overlapping parameter over precision. Additional study is necessary to better assess the precision of the overlapping parameter.

TABLE 5.4 – *Estimation of resource utilization, latency, and precision for various parallel and overlapping parameters for the two ONN architectures, bidirectional (Bid) or feed-forward (FF), and various sizes. 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. We also consider the number of parallel ONNs as the maximum number of parallel ONNs that can be implemented in the XC7Z020-1CLG400C FPGA. The precision is represented by the mean JSC between the ONN simulated with the Matlab emulator and the Canny algorithm on three large-scale images (peppers, house, lake).*

	Filter size	Overlap	Parallel ONNs	Resources		Latency 512x512	JSC Canny GT
				LUTs	Flip-Flops		
Bid	3x3	Y	1	484	458	$372ms$	0.41
Bid	5x5	Y	1	1214	1026	$372ms$	0.25
Bid	7x7	Y	1	2369	1626	$372ms$	0.17
Bid	3x3	N	1	484	458	$41.5ms$	0.23
Bid	5x5	N	1	1214	1026	$14.8ms$	0.16
Bid	7x7	N	1	2369	1626	$7.6ms$	0.13
Bid	3x3	Y	100	48525	45577	$3.7ms$	0.41
Bid	5x5	Y	40	49625	38520	$9.3ms$	0.25
Bid	7x7	Y	20	47980	27778	$18.6ms$	0.17
Bid	3x3	N	100	48525	45577	$0.42ms$	0.23
Bid	5x5	N	40	49625	38520	$0.37ms$	0.16
Bid	7x7	N	20	47980	27778	$0.38ms$	0.13
FF	3x3	Y	1	211	277	$301ms$	0.41
FF	5x5	Y	1	302	437	$301ms$	0.24
FF	7x7	Y	1	457	597	$301ms$	0.17
FF	3x3	N	1	211	277	$33.6ms$	0.23
FF	5x5	N	1	302	437	$11.9ms$	0.16
FF	7x7	N	1	457	597	$6.1ms$	0.13
FF	3x3	Y	290	53125	82727	$1.04ms$	0.41
FF	5x5	Y	170	51665	74297	$1.77ms$	0.24
FF	7x7	Y	110	51724	61616	$2.74ms$	0.17
FF	3x3	N	290	53125	82727	$0.12ms$	0.23
FF	5x5	N	170	51665	74297	$0.07ms$	0.16
FF	7x7	N	110	51724	61616	$0.06ms$	0.13

The other solution to evaluate the performances of the various image edge detection algorithms is to integrate them into a larger system to evaluate the larger system's performances. For example, image edge detection is often used as part of image classification applications, image segmentation applications [219], or in feature extraction applications [218]. Thus, we can modify those algorithms to replace the original image edge detection solution with one of the 2-layer ONNs configured for image edge detection in order to compare the final performances. In particular, in the next section, we set up a demonstrator integrating the ONN for image edge detection inside the scale-invariant feature transform (SIFT) feature detection algorithm.

5.4 ONN image edge detection for feature extraction

Feature detection algorithms are applied on large-scale images to detect important shapes or patterns to be used for example for object tracking in robotics navigation. Navigation is a complex and pervasive problem that allows autonomous robots to navigate safely in an environment without human interaction. Robot navigation is typically divided into various tasks, such as localization, obstacle avoidance, or mapping, among others [239, 240].

Simultaneous Localization And Mapping (SLAM) is the most widely applied algorithm for navigation [241, 242]. SLAM uses sensor data from the robot to estimate the robot's current location and to create an environment map around the robot. SLAM can use various types of sensors, for example, some SLAM algorithms use cameras as sensors and use captured images to estimate robot position and environment. Another widely applied solution is to combine feature-based object tracking from images with SLAM algorithm [243, 244, 245]. Feature-based object tracking consists of detecting and describing features from two following image frames before matching the corresponding features to compute transformation between two frames. In state-of-the-art, SLAM algorithm is combined with oriented FAST and rotated BRIEF (ORB) or speeded up robust features (SURF) algorithms [246, 244, 245, 247] because they perform fast feature detection and description. In particular, ORB is really attractive for real-time feature detection while it has limited precision. Other feature detection and description algorithms can reach better precision but are too slow to compute for real-time navigation applications. For example, the SIFT [218, 248] is one of the best-reported feature detection and description algorithms in terms of precision but it has a large computation latency.

We choose to investigate SIFT as it is often used as a baseline in various feature detection algorithms, and the first stage in the SIFT algorithm computes a difference of Gaussian (DoG), in which the output resembles an image edge detection algorithm. Thus, we propose to replace the DoG with the ONN-based image edge detection described in Section 5.3, first to evaluate the ONN-based image edge detection on a larger application and then to possibly accelerate the SIFT algorithm meanwhile maintaining a reasonable precision. In particular, we use the fast and efficient 2-layer digital feed-forward ONN configured for image edge detection to replace the SIFT-DoG. Note, we do not consider the 2-layer bidirectional ONN as it utilizes more resources and computes slower than the feed-forward ONN.

5.4.1 Feature detection and description with SIFT

Over the last twenty years, various feature detection and description algorithms have been proposed. Feature detection consists of detecting important attributes in images and is often based on edge or corner detection algorithms. After feature detection, it is necessary to describe

each frame feature in order to match the corresponding features between two following frames to correctly compute transformation, see Figure 5.15.

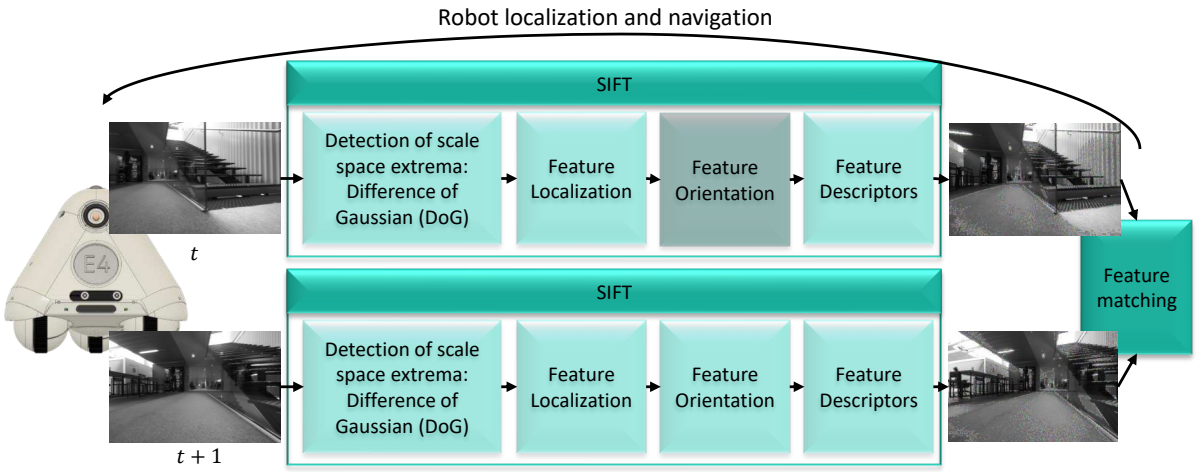


FIGURE 5.15 – *Robot movement computation based on SIFT feature detection, description, and matching.*

The differences between two feature detection and description algorithms are mainly in the mathematical approaches to detect features, and in the methods to define feature descriptors. Important parameters are scale and rotation invariance, which ensure the correct detection and description of features even if they are rotated with different sizes and scales. SIFT is one of the first feature detection and description algorithms introduced with scale and rotation invariance [218], obtaining high precision even if it necessitates large computing resources and long computation time. Since SIFT, other solutions have been proposed to reduce computation latency depending on the target applications [249, 250, 251, 246]. For example, in robotics, SIFT has been successively replaced by SURF [251] and ORB [246], which are faster and, therefore, more suitable for real-time constrained applications, see Figure 5.16.

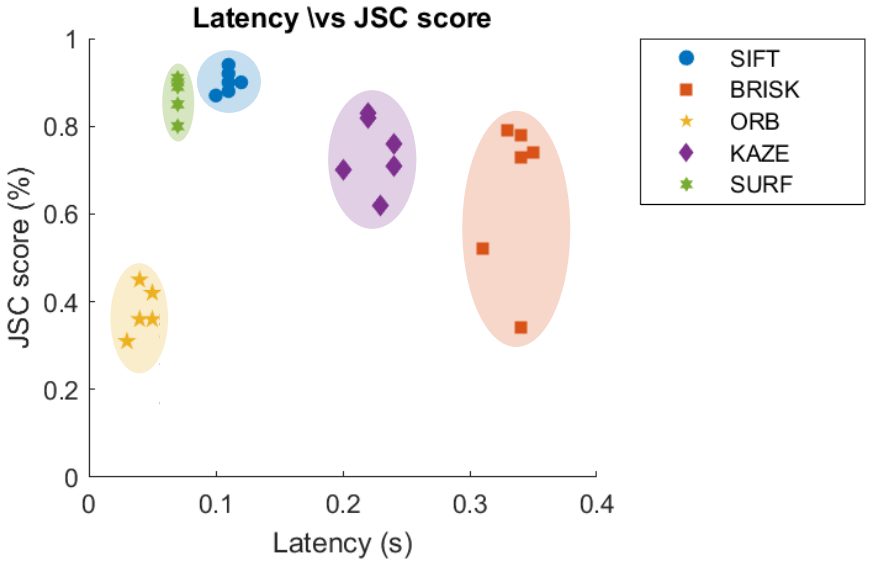


FIGURE 5.16 – *Precision over latency of state-of-the-art feature detection and description algorithms.*

SIFT can be divided into four main stages, as shown in Figure 5.15. The first stage detects extrema in various scale spaces, using DoG to detect edges in images smoothed by Gaussian blur filters with different smoothing scales. For each pixel at position (x, y) in the image, it computes:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{5.4}$$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{5.5}$$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \tag{5.6}$$

where D is the DoG output, L is the output image after the Gaussian blur is applied, G is the Gaussian blur, and σ is the smoothing scale, see Figure 5.17. Then, extrema are extracted from the DoG outputs. For each octave, each pixel of a scale space is compared with its 8 neighboring pixels from the same scale space, and with its 9 neighboring pixels from lower and upper scale spaces. Extrema are pixels with the highest or smallest values compared to all 26 neighbors, see Figure 5.17.

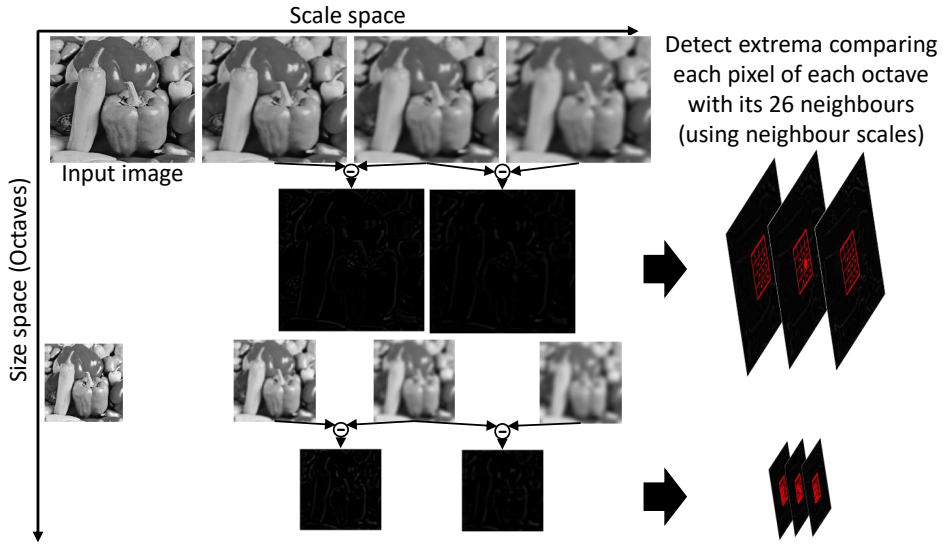


FIGURE 5.17 – *Detection of scale space extrema based on Difference of Gaussian.*

The second stage of SIFT selects key points from the extracted extrema by filtering low extrema to obtain only the strong key points that are reproducible in different images. Some extrema on edges are also filtered because edges are sensitive to noise and DoG is highly sensitive to edges. After localizing the key points, SIFT computes for each key point the magnitude and orientation on a 16x16 window. Then, it groups pixels in 4x4 windows to create orientation histograms by combining orientation and magnitude in each 4x4 window, and the main orientation from each 4x4 histogram is extracted, see Figure 5.18. The final stage creates the descriptors, which are vectors containing main orientations with magnitude around each key point, making descriptors scale and rotation invariant. Once SIFT is applied to two images, descriptors from both images are matched to determine the transformation between the two images. For example, in the case of two following images from a moving robot, the transformation can be associated with the movement of the robot. Note, that it is possible to filter the number of matches used to compute the transformation to impact the obtained transformation value.

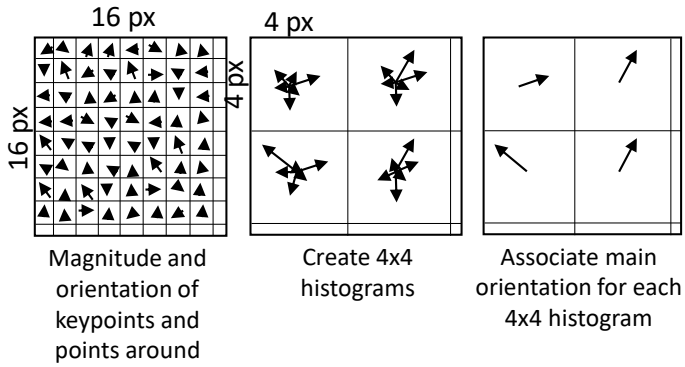


FIGURE 5.18 – *Feature orientation.*

5.4.2 SIFT-ONN adaptation

In the SIFT algorithm, the DoG computation can be approximated to an image edge detection computation. Thus, in this work, we aim to replace the SIFT DoG stage with 2-layer feed-forward ONNs configured for image edge detection with 3x3, 5x5, or 7x7 inputs, creating a hybrid SIFT-ONN algorithm, see Figure 5.19.

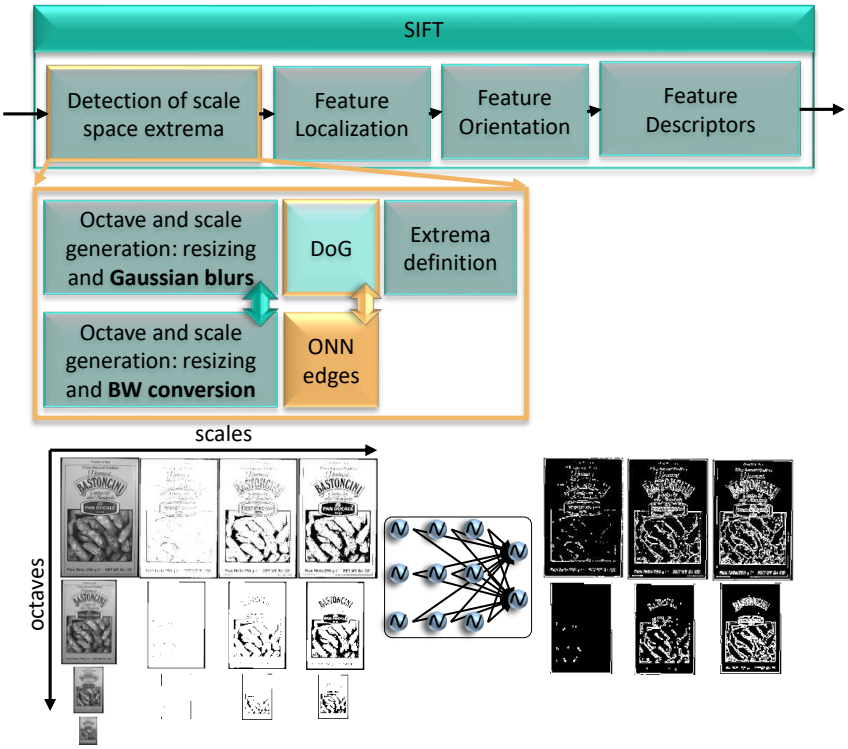
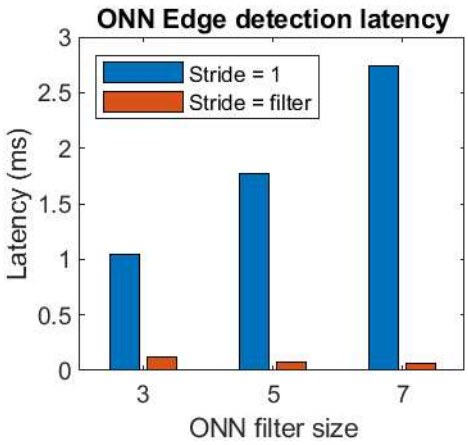


FIGURE 5.19 – *SIFT-ONN algorithm process.*

In the first stage of SIFT, a first pyramid is generated from the input image by resizing it to create various octaves. The first octave is composed of the input image up-sampled by two, and the following octaves are generated by sub-sampling by two the previous octave until the size is too small to make an Euclidean division by two. Then, a second pyramid is created by applying various Gaussian blurs on the image from each octave to create different scales of the same image, see Figure 5.17. Next, the DoG outputs are computed for each octave between neighboured Gaussian blurs to obtain images with highlighted edges. In this work, the SIFT-ONN keeps the generation of the first pyramid, creating octaves by sub-sampling the input

image, and the Gaussian blurs are replaced with image binarization using various black-and-white thresholds. The 2-layer feed-forward ONNs for image edge detection are applied to each generated black-and-white image to obtain images with highlighted edges. Finally, the images generated by the original SIFT DoG are replaced with the ones generated by the ONN for the rest of the SIFT algorithm, see Figure 5.19.

Figure 5.20 confirms that using a stride of the size of the filter really improves latency, as pointed out in Section 5.3.5. Also, in Section 5.3 the ONN output was assigned to the full selected window to avoid discontinuities. However, in SIFT-ONN, tests showed the precision is higher if only the central pixel is assigned, even though it induces discontinuities. Thus, in this Section, the ONN output is assigned to the central pixel of the scanning window. For example, for a 3x3 ONN filter, the stride is equal to 3, and the ONN output is assigned to the central pixel of the 3x3 window.



ONN filter size	#Parallel ONNs
3x3	290
5x5	170
7x7	110

FIGURE 5.20 – *Feed-forward ONN edge detection latency estimation for SIFT-ONN 512x512 image process. Note, that the number of parallel ONNs is defined from the maximum resource utilization of the Zybo Z7 board, which is equipped with a Xilinx Series-7 FPGA.*

5.4.3 Validation and evaluation methods

The feed-forward ONN was validated and evaluated for image edge detection in Section 5.3.5 using the digital ONN implemented on FPGA. Here, the feed-forward ONN configured for image edge detection is first emulated in Python with a Hopfield-based feed-forward network equivalent to the Matlab emulator used in Section 5.3 in order to integrate the ONN edge detection with SIFT built-in functions from Python libraries [252].

We evaluate the SIFT-ONN on a custom dataset containing 36 standard gray scale images with sizes 256x256 and 512x512 from [253]. Each image is resized and normalized to 512x512 pixels to become a base image. Then, from each 512x512 base image, 5 sub-images are generated by applying rotation, perspective, and size transformations. Thus, the full dataset contains 36 base images, and 180 transformed images.

SIFT-ONN is validated in Python using a Hopfield-based emulator of the digital feed-forward ONN configured for image edge detection combined with built-in SIFT Python functions [252]. Precision is obtained using images from the dataset applied to the image relocation application. In feature detection and description algorithms, precision is evaluated by checking repeatability, meaning that features can be correctly detected in images before and after transformation. In this work, to assess SIFT-ONN repeatability, the goal is to retrieve the position of a sub-image from the dataset in the original base image. To do so, feature detectors and

descriptors are first generated from all base and sub-images using SIFT-ONN, and brute force matching is performed between the sub-image descriptors and the base image descriptors. Depending on the matching, a transformation is computed to position the sub-image in the base image before comparing the obtained position with the real position of the sub-image. We compare the two images using the JSC applied to feature detection and description [234]. In this case, the JSC computes the ratio of intersection over union pixels between the real sub-image position, and the computed sub-image position, such as:

$$JSC = \frac{Intersection}{Union} \tag{5.7}$$

The final JSC score computes the average of the JSC scores of all sub-images. Moreover, we use the same method with other state-of-the-art feature detection and description algorithms to have a consistent benchmark of the SIFT-ONN solution.

After assessing the precision and repeatability of the SIFT-ONN using only Python software, we estimate the latency of the SIFT-ONN combining the digital ONN latency for the image edge detection stage with the Python SIFT libraries for the next stages. In particular, we implement the digital ONN with various sizes to measure the latency of one computation and use it to estimate the latency of one full-image scanning. Then, we extract post-place and route implementation resource utilization to deduce the maximum number of parallel ONNs that can be implemented in the Zybo Z7 board. In parallel, we measure the computation time of the other steps of SIFT-ONN from SIFT Python libraries based on the GeForce GTX 1050 Mobile GPU hardware, and we combine both results to estimate the entire SIFT-ONN latency. The latency estimated can be associated with a system based on GPU combined with an ONN-based FPGA accelerator, without taking into consideration additional data transmission latency between GPU and FPGA. We compare with state-of-the-art algorithms computed in Python based on the GeForce GTX 1050 Mobile GPU.

5.4.4 Results and benchmarking

Figure 5.21 highlights the JSC mean score obtained for each tested algorithm depending on the percentage of matches used for transformation. Note, that SIFT-ONN is tested with the three ONN filter sizes: 3x3, 5x5, and 7x7, and in each case, the stride of image scanning is equal to the filter size, for example, a stride of 3 for the 3x3 filter. Also note that when scanning with a stride of the size of the filter, the ONN output is applied only to the central pixel of the 3x3 window scanned. Table 5.5 highlights that scanning with a stride of the size of the filter does not diminish the precision score, it even increases it. However, it really decreases the real latency to test on Python, as well as the real estimated latency. Thus, we do not consider small strides due to the long computation time.

TABLE 5.5 – *ONN-SIFT real computation time obtained with the sequential process of the 3x3 Hopfield emulator in Python, estimated computation time obtained with the digital 3x3 ONN implemented on FPGA, and precision JSC score for a 3x3 ONN filter size for scanning strides 1 and 3.*

Stride	1	3
Real comp. time (s)	3712	563
Est. comp. time (s)	0.0878	0.0632
JSC score	0.16	0.28

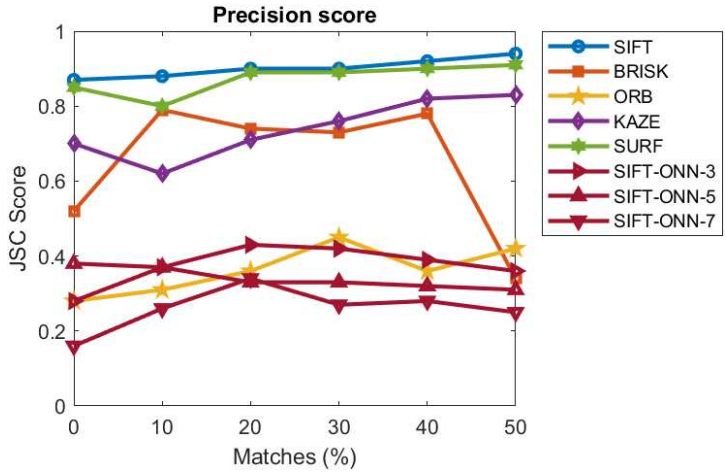


FIGURE 5.21 – *Precision scores of feature extraction algorithms compared with SIFT-ONN solution.*

On one side, Figure 5.21 shows that SIFT-ONN reduces the SIFT precision score by a factor larger than two. SIFT-ONN precision score is also lower than BRISK, SURF, and KAZE results, but similar to ORB for most of the cases. SIFT-ONN with a 3x3 filter gives the best precision score compared with larger ONN filter sizes. Note, that during tests, some divergences in the precision results were observed between two Python simulations with equal parameters. Thus, we believe the generated dataset may not be large enough to clearly assess the precision of the SIFT-ONN algorithm and further tests are necessary. However, due to the large computation time, it was not possible to enlarge the dataset. Also, precision is tested on the image relocation application. However, it is also important to test feature detection and description algorithms on other applications, like image occlusion. Additional tests on a larger dataset and on additional applications are also necessary to have a better assessment of the SIFT-ONN precision.

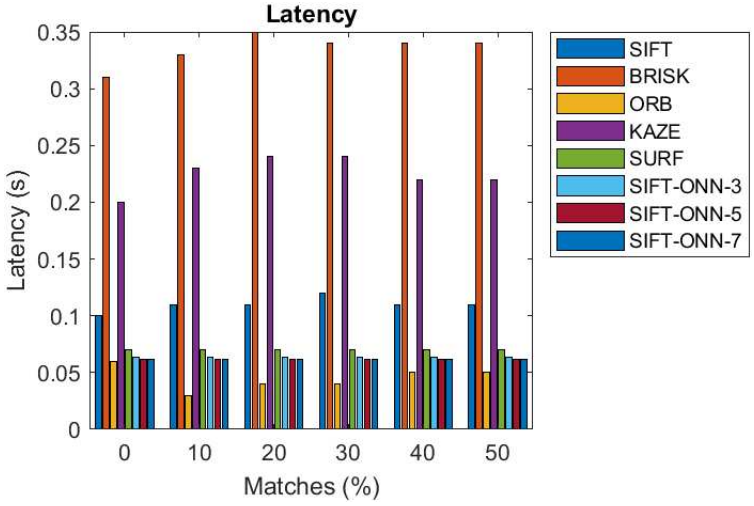


FIGURE 5.22 – *Latency estimations of ONN-SIFT for various ONN filter sizes, considering the digital ONN design, compared with the latency of state-of-the-art algorithms computed in Python based on the GeForce GTX 1050 Mobile GPU hardware.*

On the other side, Figure 5.22 shows the ONN filter size does not affect much the global SIFT-ONN latency. SIFT-ONN improves latency from the original SIFT algorithm by reducing it approximately by a factor of two. Figure 5.23 combines both precision score and latency results to highlight SIFT-ONN moves the SIFT algorithm in the same precision and latency ranges as ORB with a lower precision but a faster computation time than the original SIFT.

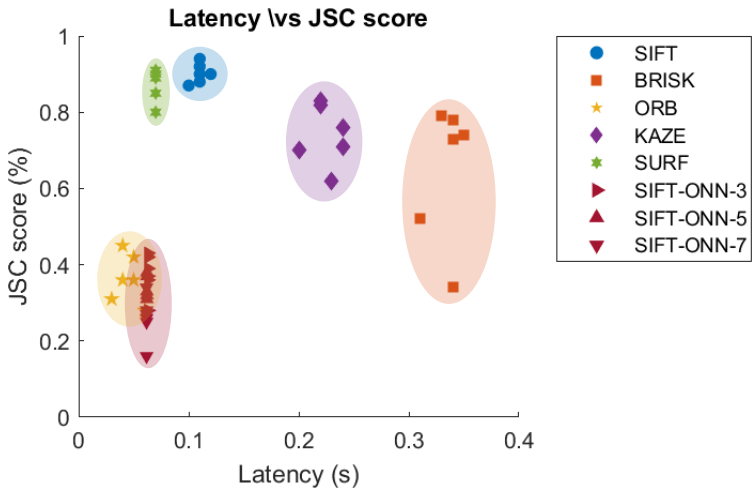


FIGURE 5.23 – *Combination of precision score and latency of SIFT-ONN compared with state-of-the-art feature detection and description algorithms.*

The aim of SIFT-ONN was to reduce SIFT computation time while keeping high precision to allow feature detection and description on edge devices. Yet, the results show the SIFT-ONN solution induces a drastic reduction of precision in comparison with SIFT, by a factor of 2. We believe this decrease is mainly due to the ONN edge detection binary output. The DoG from SIFT algorithm outputs gray-scale images with various edge strengths such as strong or weak edges, while the ONN edge detection outputs black-and-white images with binary edge information. The binary ONN output can have a negative consequence in the extrema definition such that each edge can become a maximum and each background can become a minimum depending on its position. However, in principle, ONN can stabilize to non-binary phases among the $0^o - 360^o$ range. Thus, a solution to improve ONN-SIFT precision is to investigate how to perform ONN image edge detection with non-binary ONN outputs. But, as mentioned in Chapter 3, allowing multi-state ONN outputs is not straightforward and requires further investigation. Also, this work does not compare SIFT-ONN with SIFT adapted with other image edge detection algorithms, such as state-of-the-art Sobel [223] or Canny [222]. Both algorithms can generate gray-scale images with various edge significance, thus it can help to first assess if a gray-scale ONN edge detection could solve feature detection and description with higher precision.

Still, SIFT-ONN is close to ORB performances both in terms of precision and estimated latency. ORB is currently a standard to perform feature detection and description on embedded devices due to its fast computation time, even though precision is lower than other state-of-the-art algorithms. Thus, the SIFT-ONN can become an alternative to ORB for feature detection and description, for example, in robotic applications.

It is also important to highlight that SIFT-ONN adaptation does not require important changes. Only the SIFT scale generation and DoG stages are replaced with the ONN image edge detection. However, the following SIFT stages do not require any change. Note, that we do not include an energy assessment of our solution as we do not have a full implementation of the system yet, and the digital ONN implementation is a proof of concept of the ONN paradigm which is not designed for low-power computation. However, analog ONN implementations show promising low-power computation properties [221], which can be advantageous for edge computing in robotic applications.

Finally, we validated and evaluated the SIFT-ONN solution, but up to now, there is no hardware demonstrator implemented. In literature, it is reported various FPGA implementations of

SIFT [254, 255], so in the future, we could consider developing a real-time fully digital FPGA implementation of the SIFT-ONN algorithm.

5.5 3-layer feed-forward ONN for classification

With the image edge detection application, we showcased multi-layer ONN with two layers trained with the unsupervised Hebbian learning rule.

Multi-layer ANN models are often used for classification tasks, considering more than two layers. More particularly feed-forward multi-layer models are usually trained with supervised learning algorithms for image classification such as MNIST or CIFAR-10. However, in Chapter 3 we pointed out the large model scale necessary to solve those image classification tasks, unsuitable with the limited digital ONN scale. Even if we showcased preliminary results with OHN to solve a simplified 10x10 MNIST classification, we believe alternative classification datasets are more adapted to the ONN's limited size.

The Yin-Yang dataset [256] was proposed recently in 2021 as a narrow dataset for fast training of small-scale network models. Additionally, it performs non-linear transformations between input and output data, showing better performances with deep networks compared to shallow networks. Thus, it is an interesting dataset to demonstrate the non-linear behavior of small-scale networks.

In this section, we propose to study if we can build a multi-layer feed-forward ONN with the digital design, applying it to the Yin-Yang classification task. Note, that we focus on feed-forward architecture to resemble the conventional ANN models used for classification tasks. Also note, that to combine more than two layers, alternative learning algorithms should be considered. In this section, we do not focus on novel learning solutions for multi-layer ONNs, but we aim to showcase a first 3-layer ONN computing model.

5.5.1 Yin-Yang classification dataset

The Yin-Yang dataset classifies data points into three classes based on their coordinates in a graph representing a Yin-Yang drawing, see Figure 5.24. In [256], authors used a 3-layer ANN model built with a 4-neuron input layer containing 4 coordinates $\{x, y, 1-x, 1-y\}$, a 30-neuron hidden layer, and a 3-neuron output layer representing the three classes. Each neuron has a $ReLU$ activation function with additional bias parameters. The small size of the network motivated us to test it with ONN, especially with a 3-layer feed-forward architecture simulated with the digital ONN design.

5.5.2 Training ONN for Yin-Yang classification

This work aims to validate the ONN computation with a 3-layer feed-forward architecture, studying its performances for the Yin-Yang classification task. Because it is a preliminary work on multi-layer ONN, we do not study multiple learning algorithms and prefer to validate the ONN computation first. To do so, we train an ANN with equal architecture and similar activation function using a typical back-propagation algorithm, before re-scaling and transferring the obtained weight parameters inside the digital ONN design.

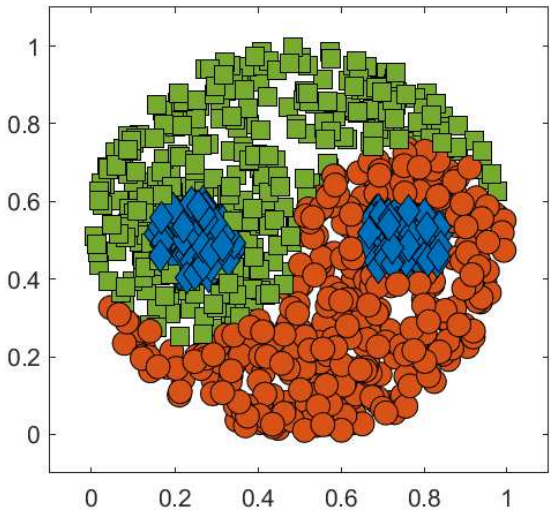


FIGURE 5.24 – *Yin-Yang test dataset used for the 3-layer feed-forward ONN classification.*

With the fully connected architecture, ONN showcased similar behavior as HNN for pattern recognition. Thus, we define the ANN activation function for training with a differentiable approximation of the original $sign$ activation function, a $hardtanh$. Also, we do not train additional bias parameters to only consider weights. We start by training the ANN with back-propagation and we test the obtained weights in various networks, close to the digital ONN design, to ensure the correct configuration and compare with the digital ONN afterward. The steps are:

1. We test the 3-layer ANN using a $hardtanh$ activation and full-precision weights.
2. We test the 3-layer ANN using a $hardtanh$ activation and 5-bit weights.
3. We test the 3-layer ANN using a $sign$ activation and full-precision weights.
4. We test the 3-layer ANN using a $sign$ activation and 5-bit weights.

Finally, we integrate the weights reduced to 5-bits signed precision in the digital ONN design to simulate and test it. It is also necessary to adapt the dataset to fit with the phase-based ONN computing. In particular, we need to define how to encode input data into phases, and how to decode phases into classes. In the original dataset, the four input neurons correspond to the four coordinates $\{x, y, 1-x, 1-y\}$ that are floating point numbers comprised between 0 and 1. The three output neurons are associated with the three classes, one for each class, and they represent the probability of the three classes comprised between 0 and 1. The highest probability is associated with the corresponding class. Considering the digital ONN, information is encoded in each oscillator phase that is limited to 16 possible stages, between $[0, 15]$, with $\{0\}$ corresponding to $\{0^o\}$ and $\{8\}$ corresponding to $\{180^o\}$. We propose two options to encode input data and decode output data for classification:

1. We encode input data between $[0^o, 180^o]$ and classify phases around $\{0^o\}$ to be a logic $\{'0'\}$, and phases around $\{180^o\}$ to be a logic $\{'1'\}$ using the trigonometric circle, see Figure 5.25.
2. We encode input data between $[0^o, 360^o]$ and classify phases between $\{0^o\}$ and $\{180^o\}$ to be a logic $\{'0'\}$, and phases between $\{180^o\}$ and $\{360^o\}$ to be a logic $\{'1'\}$ using the trigonometric circle, see Figure 5.25.

With these two methods, it is possible to classify one data input into two different classes. Thus, to differentiate, we consider the closest phase value to the original $\{'1'\}$ defined by the input data encoding. Finally, we know from Section 5.2.2 that the initialization of hidden and output layers also impacts the full ONN computation. Thus, we test various initialization parameters depending on the input data encoding.

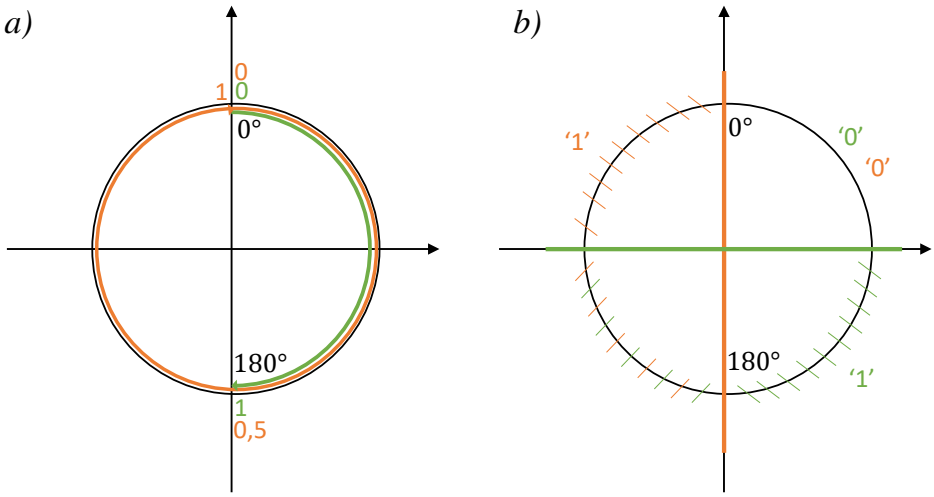


FIGURE 5.25 – *a) Input data encoding associated with b) output data encoding.*

5.5.3 Results and benchmarking

Table 5.6 presents the results of the 3-layer ONN configured for Yin-Yang classification. It highlights that training a 3-layer feed-forward ANN with $hardtanh$ activation instead of the original $ReLU$ function still achieves high accuracy on the Yin-Yang dataset. However, when reducing weight precision to the signed 5-bit precision, accuracy reduces drastically and when transferring the weights to the network with a $sign$ activation, accuracy drops even more, no matter the weight precision. Due to the binary output states of $sign$, in a lot of cases, the network activates more than one output neuron, making it impossible to determine the output class. Nevertheless, we point out that for each sample, the 3-layer digital ONN output converges to a final phase state, making it possible to compute with a 3-layer feed-forward ONN. More than that, it stabilizes to multi-phase values, not only in- or out-of-phase, allowing to better differentiate classes compared to Hopfield neurons with a $sign$ activation function. The behavior with activation of multiple classes for a single sample also appears sometimes with the digital ONN when two output neurons have equal phase distance with the phase equivalent to the $\{'1'\}$ output, but less frequently than with the binary Hopfield activation.

With the current solution, the digital ONN achieves better precision than an ANN with binary output states but a slightly lower precision than the continuous network with $hardtanh$ activation with 5-bit weight precision and a lower precision than a shallow network tested in [256]. However, these results are preliminary results, and there are many possible improvements to investigate with the 3-layer feed-forward ONN. First, we believe that increasing the weight precision inside the digital ONN design could improve the ONN classification accuracy. Additionally, the current oscillation period offers 16 possible phase values, and it might be necessary to increase the number of possible phase values to better differentiate output classes. Then, from the original network presented in [256] with $ReLU$ function to the $hardtanh$ version, accuracy reduced from 97.6% to 81.7%. So, we believe we could find a better approxima-

tion of the ONN activation function before training the ANN to obtain a better configuration of the weight parameters. Also, alternative learning algorithms, more adapted to ONN computing paradigms need to be explored, such as the Oscillatory Hebbian Rule (OHR) [229] or the equilibrium propagation (EP) [58]. Finally, in [256], they showcase an increase in accuracy when increasing the number of hidden neurons from 20 to 30. The current 3-layer digital ONN utilizes a low amount of resources, see Table 5.7, so we could use a larger hidden layer. Table 5.7 also presents the latency of computation of the 3-layer ONN using digital resources with 23.72μs per inference, which is fast and can be used for many real-time applications.

TABLE 5.6 – *Accuracy of the 3-layer feed-forward ANNs (4x30x3) for Yin-Yang classification.*

Network	Input data encoding	Hidden/Output input phase	Accuracy
1. $hardtanh$ FP	$[-1, 1]$	0	81.7%
2. $hardtanh$ 5-bit	$[-1, 1]$	0	56%
3. $sign$ FP	$[-1, 1]$	0	27.3%
4. $sign$ 5-bit	$[-1, 1]$	0	19.9%
ONN	$[0^o, 360^o]$	0^o	48.2%
		180^o	46.4%
ONN	$[0^o, 180^o]$	0^o	22.3%
		90^o	34.9%
Shallow[256]	$[0, 1]$	random	63.8%
ReLU[256]	$[0, 1]$	random	97.6%

TABLE 5.7 – *3-layer feed-forward ONN characteristics.*

F_{sys}	F_{osc}	Inference latency	LUTs	Flip-Flops
16.67MHz	260 KHz	23.72μs	643	762

5.6 Discussion and conclusion

This chapter proposed novel architectures to go beyond OHN and cascaded OHN architectures and applications. OHN limits the system scalability due to the fully-connected architecture and also constrains the ONN applications to pattern recognition with limited precision. ANN models are usually organized in layered architectures to perform a wide variety of tasks with high precision. In Chapter 4 we proposed to create layers of OHNs by cascading them and highlighted that the novel cascaded OHN architecture helps to partially solve the scalability issue by limiting the number of synaptic elements meanwhile it still limits ONN to pattern recognition tasks. So, in this chapter, we studied multi-layer ONN architectures considering layers of unconnected oscillators to further resemble conventional ANN multi-layer architectures.

In particular, we proposed two different architectures considering either bidirectional or feed-forward synaptic connections between layers. By default, coupling two oscillators creates a bidirectional connection between them, being feed-forward and recurrent at the same time, and most ONN analog or mixed-signal implementations can not separate the two paths. However, novel ONN designs [140, 183, 232], including the digital ONN design from Chapter 2, can differentiate both feed-forward and recurrent paths between coupled oscillators and create feed-forward synaptic connections. So, we first proposed the multi-layer bidirectional ONN architecture to fit with all ONN implementations, before testing the multi-layer feed-forward architecture with the digital ONN design.

We first demonstrated the multi-layer ONN capabilities considering two layers to efficiently perform image edge detection by replacing two convolution filters with either one 2-layer bidirectional ONN or one 2-layer feed-forward ONN. We validated both architectures for the image edge detection application simulating and implementing the digital ONN design to compare with state-of-the-art Sobel [223] and Canny [222] edge detection algorithms. Then, for a better assessment of the ONN-based image edge detection performances, we integrated it into the SIFT feature detection algorithm, replacing the equivalent DoG to create the SIFT-ONN. The SIFT-ONN reduced the SIFT precision but accelerated the SIFT computation, bringing SIFT-ONN to similar performances as ORB's state-of-the-art feature detection solution for fast embedded systems [246].

With the 2-layer ONN, the main limitation is the binary output states which limit the precision for both the simple image edge detection and the larger feature detection and extraction applications. However, with the current binary limitation, we can already propose alternative applications to image edge detection. In particular, in the literature, others proposed to replace convolution filters with ONN [150, 143] but not only for image edge detection. We believe the 2-layer ONNs can also replace other convolution filters than the Sobel kernels. For example, 2-layer ONNs could be used as convolution filter accelerators for CNN models, as proposed in [143].

Furthermore, we showcased that ONN can stabilize to multi-state output phases with a feed-forward 3-layer architecture. To go beyond two layers and enlarge the ONN scope of application, we introduced a 3-layer feed-forward ONN and applied it to the small-scale Yin-Yang classification task considering the digital ONN design. We performed back-propagation learning on an ANN with equal architecture and an approximated activation function before re-scaling and transferring weights in the digital ONN design. It highlighted the 3-layer feed-forward ONN can stabilize to multi-state output phases, achieving better accuracy than the equivalent ANN with a $sign$ activation, even though accuracy is much lower than the equivalent ANN with continuous $hardtanh$ activation. This preliminary study of multi-layer ONN for classification encourages further investigations for a better network configuration and novel ONN-based learning algorithms, to improve ONN accuracy. Especially, the recent emergence of hardware-aware learning algorithms, such as the ONN-specific oscillatory Hebbian rule [229], and the more general equilibrium propagation [58], could provide efficient learning for multi-layer feed-forward or bidirectional ONNs.

The work described in this Chapter resulted in one journal paper [206] and two conference papers [257, 258].

CHAPTER 6

DISCUSSION AND CONCLUSION

Neuromorphic computing is a promising area to bring powerful computing algorithms to the edge. Neuromorphic computing paradigms take inspiration first from the biological neural network structure to distribute memory and processing units, and then from the brain representation of information to compute in time. In particular, the inherent phase synchronization of coupled oscillators using temporal encoding has been used to produce low-power analog oscillatory neural networks (ONNs). State-of-the-art ONN, called oscillatory Hopfield network (OHN), creates a fully connected recurrent architecture configured with unsupervised learning to perform pattern recognition. The OHN architecture first necessitates a large number of synapses that limit the scalability of ONN implementation and restrict ONN to binary pattern recognition tasks for which OHN has limited precision. The recent investigation of low-power compact devices for large-scale analog ONN implementation motivated this thesis with the exploration of novel beyond-OHN ONN networks to assess ONN performances for alternative edge applications and better benchmark ONN with more conventional models.

To facilitate the exploration of ONN beyond OHN, we considered a reconfigurable digital ONN implementation on FPGA. The digital ONN design does not behave identically to analog ONN designs but provides a proof of concept of the ONN computing paradigm to demonstrate novel learning algorithms, architectures, and applications.

6.1 Contributions

In this manuscript, we investigated how to go beyond OHN with the phase-computing ONN paradigm using a digital proof of concept of the ONN on FPGA. We first studied learning algorithms to improve current OHN pattern recognition accuracy, before investigating novel architectures to go beyond OHN. We proposed to benchmark ONNs with conventional artificial neural network (ANN) models, applying them to image processing or classification edge applications.

1. How to improve OHN accuracy with alternative learning?

In chapter 3, we studied how to efficiently learn with OHN, from unsupervised to supervised learning algorithms, to assess OHN performances on pattern recognition. We also provided the first solution to perform OHN on-chip unsupervised learning using the digital design. Even if the supervised learning experiments increased OHN accuracy, OHN trained with unsupervised or supervised learning, with or without on-chip learning, did not overcome other ANN models. We believe the main accuracy limitations are

- the **binary OHN output state** and
- the **single-layer OHN architecture** which also limits the OHN scalability, restricting the digital off-chip learning OHN to 120 neurons and the digital on-chip learning OHN to 35 neurons.

2. How to create novel ONN architectures to go beyond OHN?

To avoid the accuracy and scalability limitations of OHN, we studied novel ONN architectures. First, in chapter 4, we presented an architecture with feed-forward layers of small-scale OHNs, cascading them. We divided a pattern recognition task into sub-tasks to reduce the number of synaptic elements. Even if we were able to perform edge applications, it was not straightforward to train and it still restricted ONN to pattern recognition tasks.

So, in chapter 5, we proposed multi-layer bidirectional and feed-forward ONN architectures to resemble and benchmark with conventional ANNs, going beyond pattern recognition tasks. We first validated a 2-layer ONN trained with adapted unsupervised Hebbian learning for hetero-association and highlighted its limitation to binary output phases, like in OHN. However, we demonstrated a 3-layer feed-forward ONN configured for classification with supervised learning could stabilize to multi-phase outputs. We used supervised back-propagation learning on a conventional ANN model approximating ONN and transferred weights to the digital design. However, the weight transfer from the conventional ANN model to the digital ONN drastically reduces the classification accuracy, making multi-layer ONN noncompetitive with other models. We believe the main accuracy limitations come from

- the **digital ONN design** with its limited 5-bit weight precision, and with its 16-phase oscillating periods which reduces the possible outputs when using multi-phase values for classification, making classes hardly differentiable, and
- the **back-propagation learning algorithm** applied on a conventional model which may not approximate correctly the ONN model.

3. What are the possible edge applications for ONN?

During this thesis, we demonstrated the various ONN architectures and learning algorithms, mainly with the digital ONN design, performing various edge applications. We first validated the digital OHN design in chapter 2 performing real-time digit recognition from a camera stream. Then, in chapter 4, we showcased the digital cascaded OHN architecture for a real-time obstacle avoidance application using proximity sensor input data on mobile robots, such as the industrial E4 robot from A.I.Mergence. We also proposed an analog design of the cascaded OHN architecture and used it to perform image edge detection, replacing typical convolution filters. Even if this first image edge detection achieved interesting precision, the computation latency was too long. So, in chapter 5 we implemented digitally a 2-layer ONN configured for image edge detection. We pointed out the real-time performances when taking advantage of the FPGA parallelism, notably being able to accelerate the SIFT feature detection and extraction algorithm. Finally, we studied how to solve classification tasks with ONN. First, in chapter 3, we used OHN to solve a simplified MNIST set, transforming an image classification into a pattern recognition task. Then, in chapter 5, we configured a 3-layer feed-forward ONN to solve the Yin-Yang classification task. Currently, the ONN classification accuracy does not outperform conventional ANN models certainly because of the limited precision of the digital design and the non-adapted learning solutions.

6.2 Future work

The proposed ONN architectures trained with the different learning algorithms implemented digitally did not achieve competitive precision compared with conventional ANN models or with neuromorphic spiking neural networks (SNNs). There are several areas of research to benchmark ONNs with competitive ANNs or SNNs.

1. Multi-state ONN stabilization

Currently, both single-layer and multi-layer ONNs achieve lower accuracy than conventional ANNs for image processing tasks, from pattern recognition to image classification. The phase-computing ONN provides a natural continuous activation function, however, it is often limited to binary output phases by the network configuration with binary learning patterns or algorithms. To consider multi-phase learning solutions, some proposed to take inspiration from complex Hopfield networks, considering complex weights with the intrinsic complex ONN activation function using existing complex unsupervised and supervised learning algorithms. Alternatively, our experiments on the 3-layer feed-forward ONN showcased stabilization to multi-phase outputs after configuration with back-propagation on an approximated continuous ANN. Even if ONN was able to stabilize to multi-phase outputs, the accuracy obtained was much lower than with the approximated 3-layer ANN. As a first study, future work could investigate alternative ANN continuous activation functions for a better ONN approximation before training. Then, it might also be necessary to increase the ONN implementation precision, either modifying the present digital design or considering an alternative analog implementation. Finally, future research should also focus on alternative learning algorithms compatible with multi-layer multi-phase ONNs.

2. Multi-layer ONN learning

The best-in-class supervised learning algorithm for conventional ANN models is the back-propagation algorithm. It computes the gradient of a loss function which retro-propagates through layers using the derivative of the neuronal activation function. However, it is computationally intensive, and not applicable to ONN as the precise activation function of the phase-computing ONN paradigm is still unknown. The solution to use an approximation of the activation function considered in chapter 5 did not achieve yet high precision with ONN. Recently, novel hardware-based supervised learning algorithms were introduced for physical neuromorphic computing. For example, the supervised equilibrium propagation learning algorithm was introduced for hardware-based multi-layer recurrent networks. Even if it did not significantly increase the accuracy of a single-layer ONN, we believe it requires further examination for multi-layer feed-forward or bidirectional ONN learning. The equilibrium propagation is one example and we assume there are other options to consider. In our opinion, one of the main research topics to improve ONN accuracy performances is to concentrate on novel efficient hardware-aware learning algorithms.

3. ONN implementation

In this thesis, we consider a digital ONN design as a proof of concept of the ONN computing paradigm to easily test and validate novel architectures and learning algorithms for edge computing. One of the main interests of the ONN computing paradigm comes from its possible analog implementations. Analog computing takes advantage of the physical computing properties of analog circuits for low-power fast computation, providing natural continuous activation functions. However, building large-scale analog ONNs with re-configurable synaptic architecture and weights is challenging. For example, to give synaptic architecture flexibility,

the system needs to provide possible all-to-all non-symmetric oscillator couplings with high precision, requiring a large space. Thus, the investigation of novel compact, fast, and low-power devices for analog ONN designs is another important research area. Alternatively, to keep the re-configurable advantage of the digital ONN design, we could try to bring analog properties to the digital ONN design such as stochasticity or noise.

4. ONN alternative architectures and applications beyond OHN and ANN

There are still many topics to investigate in order for ONNs to compete and benchmark with ANNs and SNNs. However, usual ANN benchmark tasks might not take advantage of the ONN physical properties, and might never achieve competitive accuracy results compared with conventional and alternative ANNs. First, accuracy might not be the most attractive criterion of the ONN computing paradigm when benchmarking with ANNs or SNNs, and in the future, it could be interesting to also benchmark ONN in terms of energy consumption, and latency of computation, among others. Then, it is also important to propose alternative beyond-conventional ANN architectures and applications, such as

- **Multimodal data processing**, considering modular ONN architectures, could use a combination of small-scale application-specific ONN accelerators in larger-scale multimodal sensory platforms, for example in robotics.
- **Dynamic data processing** could take advantage of the high non-linearity and dynamic computing of coupled oscillators. For example, frequency-computing ONNs have already been studied to build reservoir networks for reservoir computing. The first experiments are promising and motivate further analysis for temporal data processing with phase-computing ONN.
- Solving **combinatorial optimization problems (COP)** with ONN. COP can be represented with a coupling graph governed by an Ising Hamiltonian function, that can be associated with the intrinsic ONN energy function when creating graphs of coupled oscillators to physically solve COP. Recent studies showcased significant performances when using ONN for COP compared to state-of-the-art approaches, encouraging further experimentation.

This thesis manuscript explored beyond-OHN architectures, applications, and learning algorithms, taking inspiration from ANN for benchmarking. Current performances do not outperform classical ANN models yet but still encourage research of ONN for edge computation. In the future, some could explore alternative learning solutions to improve ONN as ANN performances meanwhile others could focus on alternative ONN computing architectures and applications taking more advantage of the physical ONN computing paradigm.

BIBLIOGRAPHIE

[1] Gordon E Moore. Cramming More Components onto Integrated Circuits. *PROCEEDINGS OF THE IEEE*, 86(1), 1998. 1

[2] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into Deep Learning, February 2023. 2, 139

[3] Conrad D. James, James B. Aimone, Nadine E. Miner, et al. A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications. *Biologically Inspired Cognitive Architectures*, 19:49–64, January 2017, doi:10.1016/j.bica.2016.11.002. 2, 139

[4] Vijay Balasubramanian. Brain power. *Proceedings of the National Academy of Sciences*, 118(32):e2107022118, August 2021, doi:10.1073/pnas.2107022118. Publisher: Proceedings of the National Academy of Sciences. 2, 139

[5] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, December 1943, doi:10.1007/BF02478259. 2, 139

[6] Frank Rosenblatt. Perceptron Simulation Experiments. *Proceedings of the IRE*, 48(3):301–309, March 1960, doi:10.1109/JRPROC.1960.287598. 2, 139

[7] Marvin Minsky and Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, September 2017. 2, 140

[8] *Rectified Linear Units Improve Restricted Boltzmann Machines*, 2010. 2

[9] Y. LeCun, B. Boser, J. S. Denker, et al. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1989, doi:10.1162/neco.1989.1.4.541. 3, 4, 45, 86, 93

[10] Yann Lecun, L.D. Jackel, Leon Bottou, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. In J.H. Oh, C. Kwon, and S. Cho, editors, *Neural networks*, pages 261–276. World Scientific, 1995. 3, 32, 46

[11] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, January 1962, doi:10.1113/jphysiol.1962.sp006837. 3

[12] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, March 1968, doi:10.1113/jphysiol.1968.sp008455. 3

[13] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybernetics*, 36:193–202, 1980. 3

[14] Yann LeCun. Deep learning and convolutional networks. In *2015 IEEE Hot Chips 27 Symposium (HCS)*, pages 1–95, Cupertino, CA, USA, August 2015. IEEE. 3, 57

[15] Jia Deng, Wei Dong, Richard Socher, et al. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 3, 45

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, Red Hook, NY, USA, December 2012. Curran Associates Inc. 3, 4

[17] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. 3

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385. 3

[19] Robin M. Schmidt. Recurrent Neural Networks (RNNs): A gentle Introduction and Overview, November 2019. 3

[20] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982, doi:10.1073/pnas.79.8.2554. 3, 8, 9, 17, 142

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks, February 2013. 3

[22] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, et al. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531, May 2011. ISSN: 2379-190X. 3

[23] Mohammad K. Nammous and Khalid Saeed. Natural Language Processing: Speaker, Language, and Gender Identification with LSTM. In Rituparna Chaki, Agostino Cortesi, Khalid Saeed, and Nabendu Chaki, editors, *Advanced Computing and Systems for Security: Volume Eight*, Advances in Intelligent Systems and Computing, pages 143–156. Springer, Singapore, 2019. 3

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, et al., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. 3

[25] Mark Chen, Alec Radford, Rewon Child, et al. Generative Pretraining From Pixels. In *Proceedings of the 37th International Conference on Machine Learning*, pages 1691–1703. PMLR, November 2020. ISSN: 2640-3498. 3

[26] OpenAI. GPT-4 Technical Report, March 2023. arXiv:2303.08774 [cs]. 3

[27] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986, doi:10.1038/323533a0. 4

[28] R.G.M Morris. D.O. Hebb: The Organization of Behavior, Wiley: New York; 1949. *Brain Research Bulletin*, 50(5-6):437, November 1999, doi:10.1016/S0361-9230(99)00182-3. 4, 7, 18, 33, 34, 63, 68, 79

[29] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised Visual Representation Learning by Context Prediction. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1422–1430, December 2015. ISSN: 2380-7504. 4

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. arXiv:1810.04805 [cs]. 4

[31] Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46, July 1995, doi:10.1016/0921-8890(95)00004-Y. 4, 31

[32] Mark B. Ring. CHILD: A First Step Towards Continual Learning. *Machine Learning*, 28(1):77–104, July 1997, doi:10.1023/A:1007331723572. 4, 31

[33] Vithursan Thangarasa, Thomas Miconi, and Graham W. Taylor. Enabling Continual Learning with Differentiable Hebbian Plasticity, June 2020. arXiv:2006.16558 [cs, stat]. 4, 31

[34] Michael McCloskey and Neal J. Cohen. Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem. In Gordon H. Bower, editor, *Psychology of Learning and Motivation*, volume 24, pages 109–165. Academic Press, January 1989. 4, 31

[35] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4), 1999. 4, 31

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, June 2016. ISSN: 1063-6919. 4

[37] Malay Shah and Rupal Kapdi. Object detection using deep neural networks. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 787–790, June 2017. 4

[38] Tiejun Yang and Jikun Song. An Automatic Brain Tumor Image Segmentation Method Based on the U-net. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pages 1600–1604, December 2018. 4

[39] Licheng Jiao, Fan Zhang, Fang Liu, et al. A Survey of Deep Learning-Based Object Detection. *IEEE Access*, 7:128837–128868, 2019, doi:10.1109/ACCESS.2019.2939201. Conference Name: IEEE Access. 4

[40] W. Xiong, L. Wu, F. Alleva, et al. The Microsoft 2017 Conversational Speech Recognition System. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5934–5938, April 2018. ISSN: 2379-190X. 4

[41] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, et al. Speech Recognition Using Deep Neural Networks: A Systematic Review. *IEEE Access*, 7:19143–19165, 2019, doi:10.1109/ACCESS.2019.2896880. Conference Name: IEEE Access. 4

[42] Sylvain Saïghi. Green AI : La nécessité de la soutenabilité énergétique dans l'IA. In *14ème édition de FETCH*, Montréal, France, 2020. 4

[43] Jesse Dodge, Taylor Prewitt, Remi Tachet des Combes, et al. Measuring the Carbon Intensity of AI in Cloud Instances. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, pages 1877–1894, New York, NY, USA, June 2022. Association for Computing Machinery. 4

[44] Beyond von Neumann. *Nature Nanotechnology*, 15(7):507–507, July 2020, doi:10.1038/s41565-020-0738-x. 5

[45] Tien-Ju Yang, Yu-Hsin Chen, Joel Emer, and Vivienne Sze. A method to estimate the energy consumption of deep neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 1916–1920, Pacific Grove, CA, USA, October 2017. IEEE. 5

[46] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and Scalability of GPU-Based Convolutional Neural Networks. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pages 317–324, February 2010. ISSN: 2377-5750. 5

[47] Norman P. Jouppi, Cliff Young, Nishant Patil, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12, June 2017. 5

[48] Qianru Zhang, Meng Zhang, Tinghuan Chen, et al. Recent advances in convolutional neural network acceleration. *Neurocomputing*, 323:37–51, January 2019, doi:10.1016/j.neucom.2018.09.038. 5

[49] Yiran Chen, Yuan Xie, Linghao Song, et al. A Survey of Accelerator Architectures for Deep Neural Networks. *Engineering*, 6(3):264–274, March 2020, doi:10.1016/j.eng.2020.01.007. 5

[50] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, et al. A Survey of FPGA-Based Neural Network Accelerator, December 2018. arXiv:1712.08934 [cs]. 5

[51] Yunji Chen, Tianshi Chen, Zhiwei Xu, et al. DianNao family: energy-efficient hardware accelerators for machine learning. *Communications of the ACM*, 59(11):105–112, October 2016, doi:10.1145/2996864. 5, 6

[52] Vincent Gaudet. A survey and tutorial on contemporary aspects of multiple-valued logic and its application to microelectronic circuits. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 6(1):5–12, 2016, doi:10.1109/JETCAS.2016.2528041. 5

[53] Jack D. Kendall and Suhas Kumar. The building blocks of a brain-inspired computer. *Applied Physics Reviews*, 7(1):011305, March 2020, doi:10.1063/1.5129306. 5, 7

[54] John Paul Strachan, Can Li, and Catherine Graves. Analog error detection and correction in analog in-memory crossbars, August 2023. 6

[55] Bing Chen, Fuxi Cai, Jiantao Zhou, et al. Efficient in-memory computing architecture based on crossbar arrays. In *2015 IEEE International Electron Devices Meeting (IEDM)*, pages 17.5.1–17.5.4, 2015. 6

[56] George Plastiras, Maria Terzi, Christos Kyrkou, and Theocharis Theocharidcs. Edge Intelligence: Challenges and Opportunities of Near-Sensor Machine Learning Applications. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–7, July 2018. 6

[57] Lanfang Sun, Xin Jiang, Huixia Ren, and Yi Guo. Edge-Cloud Computing and Artificial Intelligence in Internet of Medical Things: Architecture, Technology and Application. *IEEE Access*, 8:101079–101092, 2020, doi:10.1109/ACCESS.2020.2997831. Conference Name: IEEE Access. 6

[58] Benjamin Scellier and Yoshua Bengio. Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Frontiers in Computational Neuroscience*, 11:24, 2017, doi:10.3389/fncom.2017.00024. 6, 32, 45, 49, 50, 54, 85, 111, 112

[59] Enrique Miranda and Jordi Suñé. Memristors for Neuromorphic Circuits and Artificial Intelligence Applications. *Materials*, 13(4):938, February 2020, doi:10.3390/ma13040938. 6

[60] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 KB RAM for the internet of things. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1935–1944. PMLR, 06–11 Aug 2017. 6

[61] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, et al. Benchmarking TinyML Systems: Challenges and Direction, January 2021. arXiv:2003.04821 [cs]. 6

[62] Binary neural networks: A survey | Elsevier Enhanced Reader. 6

[63] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, et al. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv:1602.02830 [cs]*, March 2016. 6

[64] Koen Helwegen, James Widdicombe, Lukas Geiger, et al. Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization, November 2019. 6

[65] Jérémie Laydevant, Maxence Ernoult, Damien Querlioz, and Julie Grollier. Training Dynamical Binary Neural Networks with Equilibrium Propagation. *arXiv:2103.08953 [cs]*, April 2021. 6, 54

[66] Kamal Danouchi, Guillaume Prenat, and Lorena Anghel. Spin orbit torque-based crossbar array for error resilient binary convolutional neural network. In *2022 IEEE 23rd Latin American Test Symposium (LATS)*, pages 1–6, 2022. 6

[67] Ghislain Takam Tchendjou, Kamal Danouchi, Guillaume Prenat, and Lorena Anghel. Spintronic memristor-based binarized ensemble convolutional neural network architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(6):1885–1897, 2023, doi:10.1109/TCAD.2022.3213612. 6

[68] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, 1989. 6

[69] Dennis Valbjørn Christensen, Regina Dittmann, Bernabe Linares-Barranco, et al. 2022 roadmap on neuromorphic computing and engineering. *Neuromorphic Computing and Engineering*, 2022, doi:10.1088/2634-4386/ac4a83. 6, 7, 8

[70] Jason Yik, Soikat Hasan Ahmed, Zergham Ahmed, et al. Neurobench: Advancing neuromorphic computing through collaborative, fair and representative benchmarking, 2023. 6

[71] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, et al. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *arXiv:1705.06963 [cs]*, May 2017. arXiv: 1705.06963. 7

[72] J. Grollier, D. Querlioz, K. Y. Camsari, et al. Neuromorphic Spintronics. *Nature electronics*, 3(7):10.1038/s41928–019–0360–9, 2020, doi:10.1038/s41928-019-0360-9. 7, 10, 11, 142

[73] Catherine D. Schuman, Shruti R. Kulkarni, Maryam Parsa, et al. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2(1):10–19, January 2022, doi:10.1038/s43588-021-00184-y. 7

[74] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659–1671, December 1997, doi:10.1016/S0893-6080(97)00011-7. 7, 141

[75] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, August 1952. 7

[76] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: single neurons, populations, plasticity*. Cambridge University Press, Cambridge, U.K.; New York, 2002. 7

[77] G. Indiveri, E. Chicca, and R. Douglas. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks*, 17(1):211–221, January 2006, doi:10.1109/TNN.2005.860850. Conference Name: IEEE Transactions on Neural Networks. 7

[78] E.M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, November 2003, doi:10.1109/TNN.2003.820440. 7

[79] Sangya Dutta, Vinay Kumar, Aditya Shukla, et al. Leaky Integrate and Fire Neuron by Charge-Discharge Dynamics in Floating-Body MOSFET. *Scientific Reports*, 7(1):8257, August 2017, doi:10.1038/s41598-017-07418-y. Number: 1 Publisher: Nature Publishing Group. 7

[80] Clémence Gillet, Adrien F. Vincent, Bertrand Le Gal, and Sylvain Saïghi. Flexible design methodology for spike encoding implementation on fpga. In *2022 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 379–383, 2022. 7

[81] Shirin Dora and Nikola Kasabov. Spiking Neural Networks for Computational Intelligence: An Overview. *Big Data and Cognitive Computing*, 5(4):67, December 2021, doi:10.3390/bdcc5040067. 7

[82] Kashu Yamazaki, Viet-Khoa Vo-Ho, Darshan Bulsara, and Ngan Le. Spiking Neural Networks and Their Applications: A Review. *Brain Sciences*, 12(7):863, June 2022, doi:10.3390/brainsci12070863. 7

[83] Gourav Datta, Zeyu Liu, Md Abdullah-Al Kaiser, et al. In-sensor and neuromorphic computing are all you need for energy efficient computer vision. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023. 7

[84] Manon Dampfhoffer, Thomas Mesquida, Alexandre Valentian, and Lorena Anghel. Backpropagation-based learning techniques for deep spiking neural networks: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–16, 2023, doi:10.1109/TNNLS.2023.3263008. 7

[85] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training Deep Spiking Neural Networks Using Backpropagation. *Frontiers in Neuroscience*, 10, 2016. 7

[86] João D. Nunes, Marcelo Carvalho, Diogo Carneiro, and Jaime S. Cardoso. Spiking Neural Networks: A Survey. *IEEE Access*, 10:60738–60764, 2022, doi:10.1109/ACCESS.2022.3179968. Conference Name: IEEE Access. 7

[87] Sander M Bohte and Joost N Kok. SpikeProp: Backpropagation for Networks of Spiking Neurons. *ESANN*, 2000. 7

[88] Friedemann Zenke and Surya Ganguli. SuperSpike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, 30(6):1514–1541, June 2018, doi:10.1162/neco.a.01086. 7

[89] Lukas Paulun, Anne Wendt, and Nikola Kasabov. A Retinotopic Spiking Neural Network System for Accurate Recognition of Moving Objects Using NeuCube and Dynamic Vision Sensors. *Frontiers in Computational Neuroscience*, 12, 2018. 7

[90] Jesus L. Lobo, Javier Del Ser, Albert Bifet, and Nikola Kasabov. Spiking Neural Networks and Online Learning: An Overview and Perspectives, July 2019. 7

[91] Peter O'Connor, Efstratios Gavves, and Max Welling. Training a Spiking Neural Network with Equilibrium Propagation. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, pages 1516–1523. PMLR, April 2019. ISSN: 2640-3498. 7, 54

[92] Upasana Sahu, Aadit Pandey, Kushaagra Goyal, and Debanjan Bhowmik. Spike time dependent plasticity (STDP) enabled learning in spiking neural networks using domain wall based synapses and neurons. *AIP Advances*, 9(12):125339, December 2019, doi:10.1063/1.5129729. 7

[93] Shelah Ameli, Adam Foshie, Drew Friend, et al. Algorithm and application impacts of programmable plasticity in spiking neuromorphic hardware. In *Proceedings of the 2023 International Conference on Neuromorphic Systems*, number 39 in ICONS '23. Association for Computing Machinery, 2023. 7

[94] Amar Shrestha, Khadeer Ahmed, Yanzhi Wang, and Qinru Qiu. Stable spike-timing dependent plasticity rule for multilayer unsupervised and supervised learning. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1999–2006, 2017. 7

[95] Milad Mozafari, Saeed Reza Kheradpisheh, Timothée Masquelier, et al. First-spike-based visual categorization using reward-modulated stdp. *IEEE Transactions on Neural Networks and Learning Systems*, 29(12):6178–6190, 2018, doi:10.1109/TNNLS.2018.2826721. 7

[96] James B. Aimone, Yang Ho, Ojas Parekh, et al. Provable Neuromorphic Advantages for Computing Shortest Paths. In *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '20, pages 497–499, New York, NY, USA, July 2020. Association for Computing Machinery. 7

[97] J. Darby Smith, Aaron J. Hill, Leah E. Reeder, et al. Neuromorphic scaling advantages for energy-efficient random walk computations. *Nature Electronics*, pages 1–11, February 2022, doi:10.1038/s41928-021-00705-7. Publisher: Nature Publishing Group. 7

[98] Haralampos-G. Stratigopoulos, Theofilos Spyrou, and Spyridon Raptis. Testing and Reliability of Spiking Neural Networks: A Review of the State-of-the-Art. In *36th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT 2023)*, Juan-Les-Pins, France, October 2023. 8

[99] Theofilos Spyrou and Haralampos-G. Stratigopoulos. On-Line Testing of Neuromorphic Hardware. In *28th IEEE European Test Symposium (ETS23)*, Venise, Italy, May 2023. 8

[100] Giacomo Indiveri, Bernabe Linares-Barranco, Tara Hamilton, et al. Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, 5, 2011, doi:10.3389/fnins.2011.00073. 8

[101] Eustace Painkras, Luis A. Plana, Jim Garside, et al. SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation. *IEEE Journal of Solid-State Circuits*, 48(8):1943–1953, August 2013, doi:10.1109/JSSC.2013.2259038. Conference Name: IEEE Journal of Solid-State Circuits. 8

[102] Filipp Akopyan, Jun Sawada, Andrew Cassidy, et al. TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(10):1537–1557, October 2015, doi:10.1109/TCAD.2015.2474396. Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 8

[103] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, et al. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro*, 38(1):82–99, January 2018, doi:10.1109/MM.2018.112130359. Conference Name: IEEE Micro. 8

[104] Yannik Stradmann, Sebastian Billaudelle, Oliver Breitwieser, et al. Demonstrating Analog Inference on the BrainScaleS-2 Mobile System. *IEEE Open Journal of Circuits and Systems*, 3:252–262, 2022, doi:10.1109/OJCAS.2022.3208413. Conference Name: IEEE Open Journal of Circuits and Systems. 8

[105] Daniele Ielmini and Stefano Ambrogio. Emerging neuromorphic devices. *Nanotechnology*, 31(9):092001, December 2019, doi:10.1088/1361-6528/ab554b. Publisher: IOP Publishing. 8

[106] Pentti Kanerva. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation*, 1(2):139–159, June 2009, doi:10.1007/s12559-009-9009-8. 8

[107] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, et al. In-memory hyperdimensional computing. *Nature Electronics*, 3(6):327–337, June 2020, doi:10.1038/s41928-020-0410-3. 8

[108] Tetsuro Endo and Kazuhiro Takeyama. Neural network using oscillators. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 75(5):51–59, 1992, doi:10.1002/ecjc.4430750505. 8, 9

[109] Emmanuelle Tognoli and J. A. Scott Kelso. Brain coordination dynamics: True and false faces of phase synchrony and metastability. *Progress in Neurobiology*, 87(1):31–40, January 2009. 8

[110] Arthur T. Winfree. Biological rhythms and the behavior of populations of coupled oscillators. *Journal of Theoretical Biology*, 16(1):15–42, July 1967, doi:10.1016/0022-5193(67)90051-3. 8

[111] Florian Dörfler and Francesco Bullo. Synchronization in complex networks of phase oscillators: A survey. *Automatica*, 50(6):1539–1564, June 2014, doi:10.1016/j.automatica.2014.04.012. 8

[112] Gyorgy Csaba and Wolfgang Porod. Coupled oscillators for computing: A review and perspective. *Applied Physics Reviews*, 7(1):011302, March 2020, doi:10.1063/1.5120412. 8

[113] E. Paxon Frady and Friedrich T. Sommer. Robust computation with rhythmic spike patterns. *Proceedings of the National Academy of Sciences*, 116(36):18050–18059, September 2019, doi:10.1073/pnas.1902653116. 8

[114] Christiaan Huygens and Richard J. Blackwell. *Christiaan Huygens' the pendulum clock, or, Geometrical demonstrations concerning the motion of pendula as applied to clocks*. Iowa State University Press series in the history of technology and science. Iowa State University Press, Ames, Iowa, 1st ed edition, 1986. 8

[115] John Von Neumann. Non-linear capacitance or inductance switching, amplifying, and memory organs, December 1957. 8

[116] Eiichi Goto. The Parametron, a Digital Computing Element Which Utilizes Parametric Oscillation. *Proceedings of the IRE*, 47(8):1304–1316, August 1959, doi:10.1109/JRPROC.1959.287195. Conference Name: Proceedings of the IRE. 8, 81

[117] J. J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81(10):3088–3092, May 1984, doi:10.1073/pnas.81.10.3088. 8, 9

[118] Ernst Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik*, 31(1):253–258, February 1925, doi:10.1007/BF02980577. 8, 16

[119] F.C. Hoppensteadt and E.M. Izhikevich. Pattern recognition via synchronization in phase-locked loop neural networks. *IEEE Transactions on Neural Networks*, 11(3):734–738, May 2000, doi:10.1109/72.846744. 8, 9, 15, 16, 141

[120] Toshio Aoyagi. Network of Neural Oscillators for Retrieving Phase Information. *Physical Review Letters*, 74(20):4075–4078, May 1995, doi:10.1103/PhysRevLett.74.4075. Publisher: American Physical Society. 8

[121] Yoshiki Kuramoto. *Chemical Oscillations, Waves, and Turbulence*, volume 19 of *Springer Series in Synergetics*. Springer, Berlin, Heidelberg, 1984. 8, 9

[122] Edmund T. Rolls. Attractor networks. *WIREs Cognitive Science*, 1(1):119–134, January 2010, doi:10.1002/wcs.1. 9, 16

[123] DeLiang Wang and D. Terman. Locally excitatory globally inhibitory oscillator networks: theory and application to pattern segmentation. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 136–145, Ermioni, Greece, 1994. IEEE. 9, 10

[124] F.C. Hoppensteadt and E.M. Izhikevich. Associative memory of weakly connected oscillators. In *Proceedings of International Conference on Neural Networks (ICNN'97)*, volume 2, pages 1135–1138, Houston, TX, USA, 1997. IEEE. 9, 17, 77

[125] David Terman and DeLiang Wang. Global competition and local cooperation in a network of neural oscillators. *Physica D: Nonlinear Phenomena*, 81(1-2):148–176, February 1995, doi:10.1016/0167-2789(94)00205-5. 9, 10

[126] S. Campbell and De Liang Wang. Synchrony and desynchrony in integrate-and-fire oscillators. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, volume 2, pages 1498–1503, Anchorage, AK, USA, 1998. IEEE. 9

[127] Ke Chen and DeL.L. Wang. Image segmentation based on a dynamically coupled neural oscillator network. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 4, pages 2653–2658, Washington, DC, USA, 1999. IEEE. 9

[128] J. Cosp, J. Madrenas, and J. Cabestany. A VLSI implementation of a neuromorphic network for scene segmentation. In *Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, pages 403–408, April 1999. 9

[129] Jordi Cosp, Jordi Madrenas, and Daniel Fernández. Design and basic blocks of a neuromorphic VLSI analogue vision system. *Neurocomputing*, 69(16):1962–1970, October 2006, doi:10.1016/j.neucom.2005.09.019. 9

[130] D.N. Fernandes, J.P. Stedile, and P.O.A. Navaux. Architecture of oscillatory neural network for image segmentation. In *14th Symposium on Computer Architecture and High Performance Computing, 2002. Proceedings.*, pages 29–36, Vitoria, Brazil, 2002. IEEE Comput. Soc. 9

[131] D. Fernandes, P. Olivier, and A. Navaux. A low complexity digital oscillatory neural network for image segmentation. In *Proceedings of the Fourth IEEE International Symposium on Signal Processing and Information Technology, 2004.*, pages 365–368, Rome, Italy, 2004. IEEE. 9, 77

[132] Bernard Girau and Cesar Torres-Huitzil. Massively distributed digital implementation of an integrate-and-fire LEGION network for visual scene segmentation. *Neurocomputing*, 70(7-9):1186–1197, March 2007, doi:10.1016/j.neucom.2006.11.009. 9

[133] Guoshen Yu and Jean-Jacques Slotine. Visual Grouping by Neural Oscillator Networks. *IEEE Transactions on Neural Networks*, 20(12):1871–1884, December 2009, doi:10.1109/TNN.2009.2031678. Conference Name: IEEE Transactions on Neural Networks. 9

[134] Xiaodong Tan, Huajun Dong, Xiaoyi Yang, and Xiaojun Tan. A hierarchical image segmentation oscillator network based on shared contextual synchronization. In *2012 International Conference on Computer Science and Information Processing (CSIP)*, pages 113–116, August 2012. 9

[135] Matthew Cotter, Yan Fang, S.P. Levitan, et al. Computational Architectures Based on Coupled Oscillators. In *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, July 2014. 9

[136] A. A. Sharma, Y. Kesim, M. Shulaker, et al. Low-power, high-performance S-NDR oscillators for stereo (3D) vision using directly-coupled oscillator networks. In *2016 IEEE Symposium on VLSI Technology*, pages 1–2, June 2016. ISSN: 2158-9682. 9, 16, 141

[137] André Noest. Phasor neural networks. In D. Anderson, editor, *Neural Information Processing Systems*, volume 0. American Institute of Physics, 1987. 9

[138] R W Hölzel and K Krischer. Pattern recognition with simple oscillating circuits. *New Journal of Physics*, 13(7):073031, July 2011, doi:10.1088/1367-2630/13/7/073031. 9, 10, 16, 29, 30, 141

[139] Gyorgy Csaba and Wolfgang Porod. Computational Study of Spin-Torque Oscillator Interactions for Non-Boolean Computing Applications. *IEEE Transactions on Magnetics*, 49(7):4447–4451, July 2013, doi:10.1109/TMAG.2013.2244202. 9

[140] Thomas C. Jackson, Abhishek A. Sharma, James A. Bain, et al. Oscillatory Neural Networks Based on TMO Nano-Oscillators and Multi-Level RRAM Cells. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(2):230–241, June 2015, doi:10.1109/JETCAS.2015.2433551. Conference Name: IEEE Journal on Emerging and Selected Topics in Circuits and Systems. 9, 86, 111

[141] Thomas Jackson, Samuel Pagliarini, and Lawrence Pileggi. An Oscillatory Neural Network with Programmable Resistive Synapses in 28 Nm CMOS. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–7, McLean, VA, USA, November 2018. IEEE. 9, 10, 11, 15, 19, 29, 30, 43, 44, 77, 142

[142] Rongye Shi, Thomas C. Jackson, Brian Swenson, et al. On the design of phase locked loop oscillatory neural networks: Mitigation of transmission delay effects. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2039–2046, Vancouver, BC, Canada, July 2016. IEEE. 9

[143] Elisabetta Corti, Joaquin Antonio Cornejo Jimenez, Kham M. Niang, et al. Coupled VO2 oscillators circuit as analog first layer filter in convolutional neural networks. *arXiv:2011.09167 [cs]*, November 2020. arXiv: 2011.09167. 9, 10, 79, 81, 112

[144] Aida Todri-Sanial, Stefania Carapezzi, Corentin Delacour, et al. How Frequency Injection Locking Can Train Oscillatory Neural Networks to Compute in Phase. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2021, doi:10.1109/TNNLS.2021.3107771. Conference Name: IEEE Transactions on Neural Networks and Learning Systems. 9

[145] Juan Núñez, María J. Avedillo, Manuel Jiménez, et al. Oscillatory Neural Networks Using VO2 Based Phase Encoded Logic. *Frontiers in Neuroscience*, 15, 2021. 9

[146] Makoto Itoh and Leon O. Chua. Star cellular neural networks for associative and dynamic memories. *International Journal of Bifurcation and Chaos*, 14(05):1725–1772, May 2004, doi:10.1142/S0218127404010308. Publisher: World Scientific Publishing Co. 10

[147] Fernando Corinto, Michele Bonnin, and Marco Gilli. Weakly connected oscillatory network models for associative and dynamic memories. *International Journal of Bifurcation and Chaos*, 17(12):4365–4379, December 2007, doi:10.1142/S0218127407020014. Publisher: World Scientific Publishing Co. 10

[148] Dmitri E. Nikonov, Gyorgy Csaba, Wolfgang Porod, et al. Coupled-Oscillator Associative Memory Array Operation for Pattern Recognition. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 1:85–93, December 2015, doi:10.1109/JXCDC.2015.2504049. 10, 16, 141

[149] N. Shukla, W.-Y Tsai, M. Jerry, et al. Ultra low power coupled oscillator arrays for computer vision applications. In *2016 IEEE Symposium on VLSI Technology*, pages 1–2, June 2016. ISSN: 2158-9682. 10, 16, 141

[150] Dmitri E. Nikonov, Peter Kurahashi, James S. Ayers, et al. Convolution Inference via Synchronization of a Coupled CMOS Oscillator Array. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(2):170–176, December 2020, doi:10.1109/JXCDC.2020.3046143. Conference Name: IEEE Journal on Exploratory Solid-State Computational Devices and Circuits. 10, 16, 112, 141

[151] Andrei Velichko, Maksim Belyaev, and Petr Boriskov. A Model of an Oscillatory Neural Network with Multilevel Neurons for Pattern Recognition and Computing. *Electronics*, 8(1):75, January 2019, doi:10.3390/electronics8010075. 10, 85, 86

[152] Damir Vodenicarevic, Nicolas Locatelli, Flavio Abreu Araujo, et al. A Nanotechnology-Ready Computing Scheme based on a Weakly Coupled Oscillator Network. *Scientific Reports*, 7(1):44772, March 2017, doi:10.1038/srep44772. Number: 1 Publisher: Nature Publishing Group. 10, 84

[153] Miguel Romera, Philippe Talatchian, Sumito Tsunegi, et al. Vowel recognition with four coupled spin-torque nano-oscillators. *Nature*, 563(7730):230–234, November 2018, doi:10.1038/s41586-018-0632-y. 10, 16, 84, 141

[154] A. Ravishankar Rao. An oscillatory neural network model that demonstrates the benefits of multisensory learning. *Cognitive Neurodynamics*, 12(5):481–499, October 2018, doi:10.1007/s11571-018-9489-x. 10

[155] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, et al. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, July 2019, doi:10.1016/j.neunet.2019.03.005. 10

[156] Danijela Marković, Nathan Leroux, Mathieu Riou, et al. Reservoir computing with the frequency, phase and amplitude of spin-torque nano-oscillators. *Applied Physics Letters*, 114(1):012409, January 2019, doi:10.1063/1.5079305. 10

[157] A. A. Velichko, D. V. Ryabokon, S. D. Khanin, et al. Reservoir computing using high order synchronization of coupled oscillators. *IOP Conference Series: Materials Science and Engineering*, 862:052062, May 2020, doi:10.1088/1757-899X/862/5/052062. Publisher: IOP Publishing. 10

[158] Chai Wah Wu. Graph coloring via synchronization of coupled oscillators. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 45(9):974–978, September 1998, doi:10.1109/81.721263. Conference Name: IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications. 10

[159] S. Dutta, A. Khanna, A. S. Assoa, et al. An Ising Hamiltonian solver based on coupled stochastic phase-transition nano-oscillators. *Nature Electronics*, 4(7):502–512, July 2021, doi:10.1038/s41928-021-00616-7. Number: 7 Publisher: Nature Publishing Group. 10, 11, 142

[160] Jaykumar Vaidya, R. S. Surya Kanthi, and Nikhil Shukla. Creating electronic oscillator-based Ising machines without external injection locking. *Scientific Reports*, 12(1):981, January 2022, doi:10.1038/s41598-021-04057-2. 10

[161] Mohammad Khairul Bashar, Antik Mallick, Daniel S. Truesdell, et al. Experimental Demonstration of a Reconfigurable Coupled Oscillator Platform to Solve the Max-Cut Problem. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 6(2):116–121, December 2020, doi:10.1109/JXCDC.2020.3025994. Conference Name: IEEE Journal on Exploratory Solid-State Computational Devices and Circuits. 11, 15, 29, 30, 43, 44, 142

[162] William Moy, Ibrahim Ahmed, Po-wei Chiu, et al. A 1,968-node coupled ring oscillator circuit for combinatorial optimization problem solving. *Nature Electronics*, 5(5):310–317, May 2022, doi:10.1038/s41928-022-00749-3. 11, 142

[163] Paolo Maffezzoni, Bichoy Bahr, Zheng Zhang, and Luca Daniel. Analysis and Design of Boolean Associative Memories Made of Resonant Oscillator Arrays. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(11):1964–1973, November 2016, doi:10.1109/TCSI.2016.2596300. 11, 142

[164] Elisabetta Corti, Bernd Gotsmann, Kirsten Moselund, et al. Resistive Coupled VO2 Oscillators for Image Recognition. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–7, McLean, VA, USA, November 2018. IEEE. 11, 142

[165] Jiaao Yu, Paul-Philipp Manea, Sara Ameli, et al. Analog Feedback-Controlled Memristor programming Circuit for analog Content Addressable Memory, April 2023. arXiv:2304.11030 [cs]. 11, 142

[166] Gyorgy Csaba, Arijit Raychowdhury, Suman Datta, and Wolfgang Porod. Computing with Coupled Oscillators: Theory, Devices, and Applications. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, Florence, 2018. IEEE. 15

[167] Arijit Raychowdhury, Abhinav Parihar, Gus Henry Smith, et al. Computing With Networks of Oscillatory Dynamical Systems. *Proceedings of the IEEE*, 107(1):73–89, January 2019, doi:10.1109/JPROC.2018.2878854. 15

[168] Henrique M. Oliveira and Luís V. Melo. Huygens synchronization of two clocks. *Scientific Reports*, 5(1):11548, July 2015, doi:10.1038/srep11548. Number: 1 Publisher: Nature Publishing Group. 15

[169] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, et al. Hopfield Networks is All You Need, April 2021. arXiv:2008.02217 [cs, stat]. 16, 17

[170] Madeleine Abernot, Thierry Gil, Manuel Jiménez, et al. Digital Implementation of Oscillatory Neural Network for Image Recognition Applications. *Frontiers in Neuroscience*, 15, 2021. 16, 18, 29, 30, 37, 38, 43, 44, 45, 57, 64

[171] Yonggang Wu, Zhigang Zheng, Longkun Tang, and Can Xu. Synchronization dynamics of phase oscillator populations with generalized heterogeneous coupling. *Chaos, Solitons & Fractals*, 164:112680, November 2022, doi:10.1016/j.chaos.2022.112680. 16, 141

[172] Frank C. Hoppensteadt and Eugene M. Izhikevich. Oscillatory Neurocomputers with Dynamic Connectivity. *Physical Review Letters*, 82(14):2983–2986, April 1999, doi:10.1103/PhysRevLett.82.2983. Publisher: American Physical Society. 16, 141

[173] Dagur I. Albertsson and Ana Rusu. Highly reconfigurable oscillator-based Ising Machine through quasiperiodic modulation of coupling strength. *Scientific Reports*, 13(1):4005, March 2023, doi:10.1038/s41598-023-31155-0. Number: 1 Publisher: Nature Publishing Group. 16, 141

[174] Sourav Dutta, Abhinav Parihar, Abhishek Khanna, et al. Programmable coupled oscillators for synchronized locomotion. *Nature Communications*, 10(1):3299, July 2019, doi:10.1038/s41467-019-11198-6. Number: 1 Publisher: Nature Publishing Group. 16, 141

[175] M.K. Muezzinoglu, C. Guzelis, and J.M. Zurada. A new design method for the complex-valued multistate Hopfield associative memory. *IEEE Transactions on Neural Networks*, 14(4):891–899, July 2003, doi:10.1109/TNN.2003.813844. Conference Name: IEEE Transactions on Neural Networks. 17

[176] Gouhei Tanaka and Kazuyuki Aihara. Complex-Valued Multistate Associative Memory With Nonlinear Multilevel Functions for Gray-Level Image Reconstruction. *IEEE Transactions on Neural Networks*, 20(9):1463–1473, September 2009, doi:10.1109/TNN.2009.2025500. Conference Name: IEEE Transactions on Neural Networks. 17

[177] Dmitry Krotov and John Hopfield. Large Associative Memory Problem in Neurobiology and Machine Learning. *arXiv:2008.06996 [cond-mat, q-bio, stat]*, August 2020. arXiv: 2008.06996. 17

[178] Pavel Tolmachev and Jonathan H. Manton. New Insights on Learning Rules for Hopfield Networks: Memory and Objective Function Minimisation. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, Glasgow, United Kingdom, July 2020. IEEE. 18, 34, 37, 47, 54

[179] Digilent. *Zybo Z7 reference guide*, 2018. 23, 37

[180] Digilent. *Pcam 5C reference manual*, 2017. 28

[181] Digilent. *Zybo Z7 -20 Pcam 5C Demo*, 2020. 28

[182] Thomas C. Jackson, Rongye Shi, Abhishek A. Sharma, et al. Implementing delay insensitive oscillatory neural networks using CMOS and emerging technology. *Analog Integrated Circuits and Signal Processing*, 89(3):619–629, December 2016, doi:10.1007/s10470-016-0803-4. 29, 30

[183] Corentin Delacour, Stefania Carapezzi, Gabriele Boschetto, et al. A mixed-signal oscillatory neural network for scalable analog computations in phase domain. *Neuromorphic Computing and Engineering*, 3(3):034004, aug 2023, doi:10.1088/2634-4386/ace9f5. 29, 30, 35, 43, 44, 86, 111

[184] Hao Lo, William Moy, Hanzhao Yu, et al. A 48-node All-to-all Connected Coupled Ring Oscillator Ising Solver Chip. preprint, In Review, January 2023. 29, 30

[185] Zheda Mai, Ruiwen Li, Jihwan Jeong, et al. Online continual learning in image classification: An empirical survey. *Neurocomputing*, 469:28–51, January 2022, doi:10.1016/j.neucom.2021.10.021. 31

[186] Matthias De Lange, Rahaf Aljundi, Marc Masana, et al. A Continual Learning Survey: Defying Forgetting in Classification Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, July 2022, doi:10.1109/TPAMI.2021.3057446. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. 31

[187] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, et al. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, June 2020, doi:10.1016/j.inffus.2019.12.004. 31

[188] Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards Continual Reinforcement Learning: A Review and Perspectives. *Journal of Artificial Intelligence Research*, 75:1401–1476, December 2022, doi:10.1613/jair.1.13673. 31

[189] Javier R. Movellan. Contrastive Hebbian Learning in the Continuous Hopfield Model. In *Connectionist Models*, pages 10–17. Elsevier, 1991. 32, 49

[190] L. Personnaz, I. Guyon, and G. Dreyfus. Collective computational properties of neural networks: New learning mechanisms. *Physical Review A*, 34(5):4217–4228, November 1986, doi:10.1103/PhysRevA.34.4217. Publisher: American Physical Society. 33, 34

[191] Sigurd Diederich and Manfred Opper. Learning of correlated patterns in spin-glass networks by local learning rules. *Physical Review Letters*, 58(9):949–952, March 1987, doi:10.1103/PhysRevLett.58.949. 33, 34

[192] E. Gardner. The space of interactions in neural network models. *Journal of Physics A: Mathematical and General*, 21(1):257–270, January 1988, doi:10.1088/0305-4470/21/1/030. Publisher: IOP Publishing. 34

[193] W. Krauth and M. Mezard. Learning algorithms with optimal stability in neural networks. *Journal of Physics A: Mathematical and General*, 20(11):L745–L752, August 1987, doi:10.1088/0305-4470/20/11/013. Publisher: IOP Publishing. 34

[194] Amos Storkey. Increasing the capacity of a hopfield network without sacrificing functionality. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, et al., editors, *Artificial Neural Networks — ICANN'97*, volume 1327, pages 451–456. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997. Series Title: Lecture Notes in Computer Science. 34, 41, 47

[195] Corentin Delacour and Aida Todri-Sanial. Mapping Hebbian Learning Rules to Coupling Resistances for Oscillatory Neural Networks. *Frontiers in Neuroscience*, 15, 2021. 34, 77

[196] Madeleine Abernot, Thierry Gil, and Aida Todri-Sanial. On-Chip Learning with a 15-neuron Digital Oscillatory Neural Network Implemented on ZYNQ Processor. In *Proceedings of the International Conference on Neuromorphic Systems 2022*, ICONS '22, pages 1–4, New York, NY, USA, September 2022. Association for Computing Machinery. 38, 43, 55, 57

[197] Madeleine Abernot, Nadine Azemard, and Aida Todri-Sanial. Oscillatory neural network learning for pattern recognition: an on-chip learning perspective and implementation. *Frontiers in Neuroscience*, 17, 2023. 38, 39, 55, 57, 76

[198] Madeleine Abernot, Thierry Gil, Evgenii Kurylin, et al. Oscillatory Neural Networks for Obstacle Avoidance on Mobile Surveillance Robot E4. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2022. ISSN: 2161-4407. 44, 45, 60, 82

[199] Madeleine Abernot, Thierry Gil, and Aida Todri-Sanial. ONN-Based On-chip Learning for Obstacle Avoidance on Mobile Robot, March 2023. 45, 82

[200] M A Belyaev and A A Velichko. Classification of handwritten digits using the Hopfield network. *IOP Conference Series: Materials Science and Engineering*, 862:052048, May 2020, doi:10.1088/1757-899X/862/5/052048. 45, 46, 53

[201] P.J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990, doi:10.1109/5.58337. 49

[202] Tamas Rudner, Wolfgang Porod, and Gyorgy Csaba. Design of Oscillatory Neural Networks by Machine Learning, September 2023. 49, 54, 55, 85

[203] Benjamin Scellier and Yoshua Bengio. Equivalence of Equilibrium Propagation and Recurrent Backpropagation. *arXiv:1711.08416 [cs]*, May 2018. arXiv: 1711.08416. 49

[204] Gianluca Zoppo, Francesco Marrone, Michele Bonnin, and Fernando Corinto. Equilibrium Propagation and (Memristor-based) Oscillatory Neural Networks. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 639–643, May 2022. ISSN: 2158-1525. 50

[205] Jia Liu, Maoguo Gong, and Haibo He. Deep associative neural network for associative memory based on unsupervised representation learning. *Neural Networks*, 113:41–53, May 2019, doi:10.1016/j.neunet.2019.01.004. 54

[206] Madeleine Abernot and Aida Todri-Sanial. Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures. *Neuromorphic Computing and Engineering*, 2023, doi:10.1088/2634-4386/acb2ef. 54, 112

[207] Teuvo Kohonen. Correlation Matrix Memories. *IEEE Transactions on Computers*, C-21(4):353–359, April 1972, doi:10.1109/TC.1972.5008975. Conference Name: IEEE Transactions on Computers. 54, 85

[208] B. Kosko. Bidirectional associative memories. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):49–60, February 1988, doi:10.1109/21.87054. 54, 85

[209] Zijia Yang and Xiaoping Wang. Memristor-based BAM circuit implementation for image associative memory and filling-in. *Neural Computing and Applications*, 33(13):7929–7942, July 2021, doi:10.1007/s00521-020-05538-7. 54

[210] Shanshan Qin, Nayantara Mudur, and Cengiz Pehlevan. Contrastive Similarity Matching for Supervised Learning. *Neural computation*, 33(5):1300–1328, April 2021, doi:10.1162/neco.a.01374. 54

[211] Jack Kendall, Ross Pantone, Kalpana Manickavasagam, et al. Training End-to-End Analog Neural Networks with Equilibrium Propagation, June 2020. arXiv:2006.01981 [cs]. 54

[212] Madeleine Abernot and Aida Todri-Sanial. Training energy-based single-layer Hopfield and oscillatory networks with unsupervised and supervised algorithms for image classification. *Neural Computing and Applications*, June 2023, doi:10.1007/s00521-023-08672-0. 55, 57

[213] K. Yojitha, B. Bindu Priya, N. Hari Krishna, et al. A Survey on Obstacle Avoidance and Traffic Light Detection Approaches for Autonomous Vehicle. In *2021 International Conference on Intelligent Technologies (CONIT)*, pages 1–4, June 2021. 59

[214] Valentino Braitenberg. Vehicles: Experiments in Synthetic Psychology, 1986. 62

[215] Giorgio Gosti, Viola Folli, Marco Leonetti, and Giancarlo Ruocco. Beyond the Maximum Storage Capacity Limit in Hopfield Recurrent Neural Networks. *Entropy*, 21(8):726, August 2019, doi:10.3390/e21080726. 64

[216] Madeleine Abernot, Hamza Amara, Thierry Gil, and Aida Todri-Sanial. Oscillatory neural network for edge computing: A mobile robot obstacle avoidance application. In *2022 IEEE International Conference on Metrology for Extended Reality, Artificial Intelligence and Neural Engineering (MetroXRAINE)*, pages 181–186, 2022. 71, 72, 73, 74, 82

[217] Elisabetta Corti, Bernd Gotsmann, Kirsten Moselund, et al. Scaled resistively-coupled VO2 oscillators for neuromorphic computing. *Solid-State Electronics*, 168:107729, June 2020, doi:10.1016/j.sse.2019.107729. 77

[218] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004, doi:10.1023/B:VISI.0000029664.99615.94. 78, 100, 101

[219] Ailing De and Chengan Guo. An image segmentation method based on the fusion of vector quantization and edge detection with applications to medical image processing. *International Journal of Machine Learning and Cybernetics*, 5(4):543–551, August 2014, doi:10.1007/s13042-013-0205-1. 78, 100

[220] Juan Núñez, José M. Quintana, María J. Avedillo, et al. Insights Into the Dynamics of Coupled VO2 Oscillators for ONNs. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(10):3356–3360, October 2021, doi:10.1109/TCSII.2021.3085133. 79

[221] Corentin Delacour, Stefania Carapezzi, Madeleine Abernot, and Aida Todri-Sanial. Energy-performance assessment of oscillatory neural networks based on vo $_2$ devices for future edge ai computing. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2023, doi:10.1109/TNNLS.2023.3238473. 79, 107

[222] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, November 1986, doi:10.1109/TPAMI.1986.4767851. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence. 80, 88, 107, 112

[223] Irwin Sobel and Gary Feldman. A 3×3 isotropic gradient operator for image processing. *Pattern Classification and Scene Analysis*, pages 271–272, January 1973. 80, 88, 107, 112

[224] S. Amarel, G. Cooke, and R. O. Winder. Majority Gate Networks. *IEEE Transactions on Electronic Computers*, EC-13(1):4–13, February 1964, doi:10.1109/PGEC.1964.263829. 81

[225] Madeleine Abernot, Corentin Delacour, Ahmet Suna, et al. Two-Layered Oscillatory Neural Networks with Analog Feedforward Majority Gate for Image Edge Detection Application. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2023. 82, 86

[226] Young, Scott, and Nasrabadi. Object recognition using multi-layer Hopfield neural network. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 417–422, June 1994. ISSN: 1063-6919. 84

[227] Damir Vodenicarevic, Nicolas Locatelli, and Damien Querlioz. A Neural Network Based on Synchronized Pairs of Nano-Oscillators. In *2017 IEEE 17th International Conference on Nanotechnology (IEEE-NANO)*, pages 512–514, July 2017. arXiv:1709.02274 [cs]. 84

[228] María J. Avedillo, José María Quintana, and Juan Núñez. Phase Transition Device for Phase Storing. *IEEE Transactions on Nanotechnology*, 19:107–112, 2020, doi:10.1109/TNANO.2020.2965243. Conference Name: IEEE Transactions on Nanotechnology. 85, 86

[229] Jafar Shamsi, María José Avedillo, Bernabe Linares-Barranco, and Teresa Serrano-Gotarredona. Oscillatory Hebbian Rule (OHR): An adaption of the Hebbian rule to Oscillatory Neural Networks. In *2020 XXXV Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 1–6, November 2020. ISSN: 2640-5563. 85, 86, 111, 112

[230] Bart Kosko. Bidirectional Associative Memories: Unsupervised Hebbian Learning to Bidirectional Backpropagation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):103–115, January 2021, doi:10.1109/TSMC.2020.3043249. 85

[231] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015, doi:10.1038/nature14539. 86

[232] Markus Graber and Klaus Hofmann. A Coupled Oscillator Network to Solve Combinatorial Optimization Problems with Over 95% Accuracy. In *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, May 2023. ISSN: 2158-1525. 86, 111

[233] M.A. Ramalho and K.M. Curtis. Neural network arbitration for edge detection. In *Proceedings of Third International Conference on Electronics, Circuits, and Systems*, volume 2, pages 1112–1115 vol.2, 1996. 89

[234] Neo Christopher Chung, BłaŻej Miasojedow, Michał Startek, and Anna Gambin. Jaccard/Tanimoto similarity test and estimation methods for biological presence-absence data. *BMC Bioinformatics*, 20(15):644, December 2019, doi:10.1186/s12859-019-3118-5. 89, 105

[235] Zhengyang Guo, Wenbo Xu, and Zhilei Chai. Image edge detection based on fpga. In *2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science*, pages 169–171, 2010. 98

[236] Santanu Halder, Debotosh Bhattacharjee, Mita Nasipuri, and Dipak Kumar Basu. A Fast FPGA Based Architecture for Sobel Edge Detection. In Hafizur Rahaman, Sanatan Chattopadhyay, and Santanu Chattopadhyay, editors, *Progress in VLSI Design and Test*, Lecture Notes in Computer Science, pages 300–306, Berlin, Heidelberg, 2012. Springer. 98

[237] Nazma Nausheen, Ayan Seal, Pritee Khanna, and Santanu Halder. A FPGA based implementation of Sobel edge detection. *Microprocessors and Microsystems*, 56:84–91, February 2018, doi:10.1016/j.micpro.2017.10.011. 98

[238] Qian Xu, Srenivas Varadarajan, Chaitali Chakrabarti, and Lina J. Karam. A distributed canny edge detector: Algorithm and fpga implementation. *IEEE Transactions on Image Processing*, 23(7):2944–2960, 2014, doi:10.1109/TIP.2014.2311656. 98

[239] G.N. Desouza and A.C. Kak. Vision for mobile robot navigation: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):237–267, 2002, doi:10.1109/34.982903. 100

[240] P Anish, P Shalini, and DR Parhi. Mobile robot navigation and obstacle avoidance techniques: A review. *International Robotics & Automation Journal*, Volume 2(Issue 3), May 2017, doi:10.15406/iratj.2017.02.00023. 100

[241] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006, doi:10.1109/MRA.2006.1678144. 100

[242] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006, doi:10.1109/MRA.2006.1638022. 100

[243] Abbas M. Ali and Md Jan Nordin. Sift based monocular slam with multi-clouds features for indoor navigation. In *TENCON 2010 - 2010 IEEE Region 10 Conference*, pages 2326–2331, 2010. 100

[244] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics*, 31(5):1147–1163, October 2015, doi:10.1109/TRO.2015.2463671. 100

[245] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, et al. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics*, 37(6):1874–1890, December 2021, doi:10.1109/TRO.2021.3075644. 100

[246] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011. 100, 101, 112

[247] Cedric Cocaud and Takashi Kubota. Surf-based slam scheme using octree occupancy grid for autonomous landing on asteroids, 2010. 100

[248] Huiyu Zhou, Yuan Yuan, and Chunmei Shi. Object tracking using sift features and mean shift. *Computer Vision and Image Understanding*, 113(3):345–352, 2009, doi:https://doi.org/10.1016/j.cviu.2008.08.006. 100

[249] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J. Davison. Kaze features. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, et al., editors, *Computer Vision – ECCV 2012*, pages 214–227. Springer Berlin Heidelberg, 2012. 101

[250] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*, pages 2548–2555, 2011. 101

[251] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, volume 3951, pages 404–417. Springer Berlin Heidelberg, 2006. 101

[252] A clean and concise python implementation of sift (scale-invariant feature transform). 104

[253] The usc-sipi image database. 104

[254] Vanderlei Bonato, Eduardo Marques, and George A. Constantinides. A parallel hardware architecture for scale and rotation invariant feature detection. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(12):1703–1712, 2008, doi:10.1109/TCSVT.2008.2004936. 108

[255] J.Q. Peng, Y.H. Liu, C.Y. Lyu, et al. Fpga-based parallel hardware architecture for sift algorithm. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 277–282, 2016. 108

[256] Laura Kriener, Julian Göltz, and Mihai A. Petrovici. The Yin-Yang dataset, January 2022. arXiv:2102.08211 [cs, q-bio]. 108, 110, 111

[257] Madeleine Abernot, Thierry Gil, and Aida Todri-Sanial. Oscillatory Neural Network as Hetero-Associative Memory for Image Edge Detection. In *Neuro-Inspired Computational Elements Conference*, NICE 2022, pages 13–21, New York, NY, USA, March 2022. Association for Computing Machinery. 112

[258] Madeleine Abernot, Sylvain Gauthier, Theophile Gonos, and Aida Todri-Sanial. SIFT-ONN: SIFT Feature Detection Algorithm Employing ONNs for Edge Detection. In *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference*, NICE '23, pages 100–107, New York, NY, USA, April 2023. Association for Computing Machinery. 112

Madeleine Abernot

Email: madeleine.abernot@lirmm.fr
Tel: +33 (0)6 32 09 48 38
Linkedin: linkedin.com/in/madeleine-abernot
ORCiD: 0000-0002-8267-2972

LANGUAGES

French: native language
English: spoken and written well

RESEARCH INTERESTS

Neuromorphic computing, FPGA implementation, Oscillatory Neural Networks.

WORK EXPERIENCE

Nov. 2020 -Now **LIRMM, University of Montpellier – CNRS** (Montpellier, France)
- Ph.D. candidate in the H2020 EU NeurONN project (neuronn.eu/).
- Digital oscillatory neural network (ONN) implementation on FPGA for edge applications and learning.
  - Creating ONNs on FPGA with different network architectures,
  - Developing ONN-based demonstrators for edge applications,
  - Publishing journal and proceeding papers,
  - Presenting achievements in international conferences and events.

Feb.-Oct. 2020 **LIRMM, University of Montpellier – CNRS** (Montpellier, France)
- Research engineer in the H2020 EU NeurONN project (neuronn.eu/).
- Developing ONN-based applications and demonstrators.

Jun.-Aug. 2018 **Foreign intern at Develogic Gmbh** (Hamburg, Germany)
- Foreign intern for designing schematics and layout, and testing PCBs.

2016-2018 **Serma Safety and Security** (Cesson-Sévigné, France)
- Apprentice in embedded system development
  - Developing communication modules in VHDL language
  - Training people about IoT security (English/French)
  - Designing schematics and PCB, and writing C code for microcontroller

Apr.-Jul. 2016 **Opale Security** (Cesson-Sévigné, France)
- Electronic technician intern in embedded system development
- Designing electronic boards (analog circuits, C language)
- Training people about IoT security

EDUCATION

Nov. 2020 -Now **Ph.D. candidate** at the I2S Graduate School (Information, Structures and Systems Sciences) – Montpellier, France

2016-2018 **Apprentice** in Bordeaux Graduate School of Engineering, engineering degree, Master's level equivalence – Bordeaux, France

2014-2016 **Student** at the University Institute of Technology of Rennes in electrical engineering and industrial computing – Rennes, France

PUBLICATIONS

Journal articles

- M. Abernot, T. Gil, M. Jiménez, J. Núñez, et al., Digital Implementation of Oscillatory Neural Network for Image Recognition Application. Frontiers in Neuroscience, 2021, doi:10.3389/fnins.2021.713054
- M. Abernot, A. Todri-Sanial. Simulation and implementation of two-layer oscillatory neural networks for image edge detection: bidirectional and feedforward architectures. NICE, 2022, doi:10.1145/3517343.3517348
- M. Abernot, G. Boschetto, S. Carapezzi, C. Delacour, et al., Réseaux de neurones oscillants pour des calculs économes en énergie. Techniques de l'ingénieur, 2022, doi:10.51257/a-v1-h5040
- M. Abernot, N. Azémard, A. Todri-Sanial, Oscillatory neural network learning for pattern recognition: an on-chip learning perspective and implementation. Frontiers in Neuroscience, 2023, doi:10.3389/fnins.2023.1196796
- M. Abernot, A. Todri-Sanial, Training energy-based single-layer Hopfield and oscillatory networks with unsupervised and supervised algorithms for image classification. Neural Computing and Applications, 2023, doi:10.1007/s00521-023-08672-0

Conference proceedings

- M. Abernot, H. Amara, T. Gil, et al., Oscillatory Neural Network for Edge Computing: A Mobile Robot Obstacle Avoidance Application. MetroXRAINE 2022, doi:10.1109/MetroXRAINE54828.2022.996758
- M. Abernot, T. Gil, A. Todri-Sanial, On-Chip Learning with a 15-neuron Digital Oscillatory Neural Network Implemented on ZYNQ Processor. ICONS, 2022, doi:10.1145/3546790.3546822
- M. Abernot, T. Gil, A. Todri-Sanial, Oscillatory Neural Network as Hetero-Associative Memory for Image Edge Detection. NICE, 2022, doi:10.1145/3517343.3517348
- M. Abernot, T. Gil, E. Kurylin, T. Hardelin, et al., Oscillatory Neural Networks for Obstacle Avoidance on Mobile Surveillance Robot E4. IJCNN, 2022, doi:10.1109/IJCNN55064.2022.9891923
- M. Abernot, C. Delacour, A. Suna, J. Marty Gregg, et al., Two-Layered Oscillatory Neural Networks with Analog Feedforward Majority Gate for Image Edge Detection Application. ISCAS, 2023
- M. Abernot, S. Gaultier, T. Gonos, A. Todri-Sanial, SIFT-ONN: SIFT Feature Detection Algorithm Employing ONNs for Edge Detection. NICE, 2023, doi:10.1145/3584954.3584999

Talks and posters

- M. Abernot, T. Gil, A. Todri-Sanial, Using Oscillatory Neural Network for Pattern Recognition and Mobile Robot Control. SOPHI.A SUMMIT, 2020
- M. Abernot, G. Boschetto, S. Carapezzi, C. Delacour, et al., FPGA Implementation of Oscillatory Neural Networks for Artificial Intelligence Edge Computing. ACM Summer school on HPC Computer Architectures for AI and Dedicated Applications, 2021
- M. Abernot, T. Gil, C. Delacour, G. Boschetto, et al., Mobile Robot Obstacle Avoidance with Oscillatory Neural Networks on FPGA. IBM-IEEE AI compute symposium, 2021
- M. Abernot, C. Delacour, G. Boschetto, S. Carapezzi, et al., Digital Oscillatory Neural Networks for AI Edge Applications. GDR SoC2, 2022
- M. Abernot, T. Gonos, A. Todri-Sanial, Oscillatory neural networks applications for edge computing. HiPEAC, 2023
- M. Abernot, A. Todri Sanial, Oscillatory neural networks implemented on FPGA for edge computing applications. DATE, 2023
- M. Abernot, N. Azémard, A. Todri-Sanial, Digital Oscillatory Neural Networks for AI Edge Applications. GDR SoC2, 2023
- M. Abernot, T. Gil, A. Todri-Sanial, ONN-Based On-chip Learning for Obstacle Avoidance on Mobile Robot. SSI, 2023

RÉSUMÉ DE LA THÈSE

Au cours des dernières décennies, l'augmentation du nombre de données a fait émerger des algorithmes d'intelligence artificielle (IA) de plus en plus puissants. Toutefois, les systèmes matériels basés sur l'architecture de von Neuman ne peuvent pas traiter efficacement cette grande quantité de données et de calculs. Ainsi, de nouveaux paradigmes neuromorphiques inspirés des réseaux de neurones biologiques sont apparus pour proposer des systèmes rapides et à faible consommation d'énergie pour l'IA embarquée. En particulier, les réseaux de neurones oscillatoires ou *oscillatory neural networks* (ONNs) s'inspirent des ondes cérébrales pour calculer en utilisant le phénomène naturel de synchronisation des oscillateurs couplés. Les ONN sont principalement utilisés pour effectuer des tâches de reconnaissance de formes, ce qui est limitant pour l'IA embarquée. Dans cette thèse, nous utilisons une preuve de concept de l'ONN avec une conception numérique mise en œuvre sur puce FPGA pour explorer des algorithmes d'apprentissage et des architectures compatibles avec l'ONN pour de nouvelles applications d'IA embarquées.

.1 Intelligence artificielle et réseaux de neurones artificiels

Le développement de processeurs microélectroniques à grande échelle suivi par la prédiction de la loi de Moore, a conduit au développement de nouveaux algorithmes pour apporter des capacités de calcul du cerveau humain aux machines. On définit l'IA comme l'ensemble des algorithmes et des modèles, résolus par des machines, qui tentent soit de dépasser les capacités du cerveau humain pour résoudre une tâche spécifique, soit de remplacer le cerveau humain pour des tâches plus générales. Avec la complexité croissante des tâches d'IA, les algorithmes déterministes ont été abandonnés au profit de modèles probabilistes d'apprentissage. Les modèles d'apprentissage sont un sous-ensemble des algorithmes d'IA capables d'apprendre des tâches à partir de données en utilisant une fonction de coût et un algorithme d'apprentissage capable d'optimiser la fonction de coût [2]. Les modèles d'apprentissage sont souvent associés au sous-ensemble des réseaux de neurones artificiels ou *artificial neural networks* (ANNs).

Les ANN sont des modèles informatiques qui s'inspirent de la structure des réseaux neuronaux biologiques pour apprendre et calculer [3]. Le cerveau humain est composé d'un réseau de neurone connu pour exécuter des tâches difficiles tout en consommant peu d'énergie [4]. Succinctement, un réseau neuronal biologique est constitué de neurones interconnectés par des synapses. L'efficacité du cerveau humain provient principalement de sa grande échelle et de sa plasticité, lui permettant de se reconfigurer pour adapter ses connaissances au fil du temps. Les ANN sont formés de deux éléments principaux, des neurones artificiels interconnectés par des synapses artificielles. Le premier modèle mathématique d'un neurone biologique a été proposé par McCulloch et Pitts en 1943 [5] avant d'être utilisé pour calculer, sous forme de perceptron, par Rosenblatt en 1960 [6]. Il est défini par deux fonctions:

1. une **fonction d'intégration** qui intègre les informations synaptiques. La principale fonction d'intégration calcule une somme pondérée des états des neurones pré-synaptiques et des poids synaptiques.

2. Et une **fonction d'activation** qui calcule l'information intégrée pour activer ou non la sortie du neurone. A l'origine, la première fonction d'activation était une fonction $sign$, qui limite la sortie des neurones à des signaux binaires. C'est pourquoi d'autres fonctions continues sont désormais plus utilisées, telles que les fonctions $sigmoide$ et $ReLU$ (voir Figure 1).

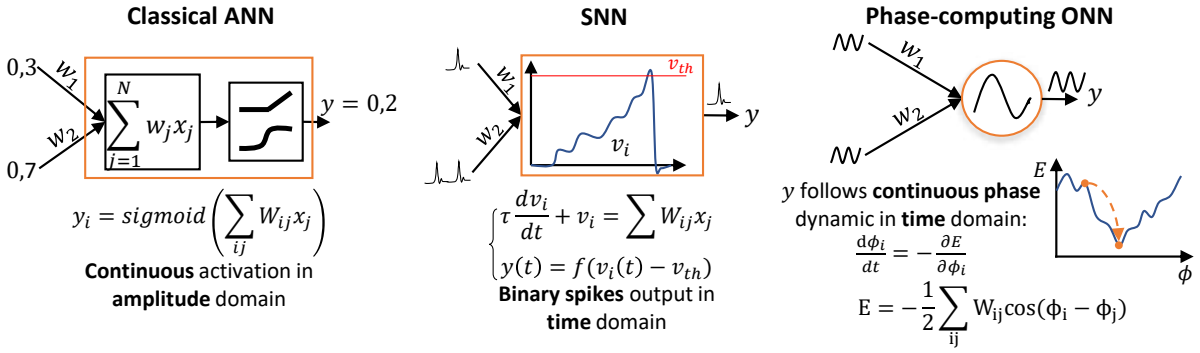


FIGURE 1 – *Différents types de réseaux de neurones, ANN, SNN, et ONN.*

Un réseau composé d'un seul neurone limite son application à des tâches linéaires entre entrées et sorties [7]. Ainsi, depuis l'introduction du perceptron, de nouvelles architectures d'ANN ont été proposées. Particulièrement, les architectures multi-couche ont été introduites, dans lesquelles il n'y a pas de connexion synaptique au sein des couches, mais il y a des connexions synaptiques entre les couches. Les synapses peuvent être unidirectionnelles, transmettant des informations des couches d'entrée aux couches de sortie. Par ailleurs, les synapses peuvent être multidirectionnelles et créer des redondances entre les couches ou entre les neurones d'une même couche, produisant ainsi des modèles récurrents. Si l'architecture d'un réseau est composée de plus d'une couche, on l'appelle un réseau de neurone profond ou *deep neural network* (DNN). Un réseau DNN de neurones perceptron produit une relation non linéaire entre l'entrée et la sortie, ce qui accroît la complexité des tâches que les ANNs peuvent résoudre. Les différentes architectures de réseaux permettent de traiter différents types de données. Par exemple, la redondance des réseaux récurrents permet de traiter des données temporelles, tandis que les réseaux unidirectionnels sont généralement utilisés pour traiter des données statiques telles que des images.

L'apprentissage des réseaux consiste à configurer les poids synaptiques pour que le réseau résolve correctement la tâche demandée. L'apprentissage dépend majoritairement de la donnée à traiter et se catégorise en deux types:

1. l'apprentissage **supervisé** considère des données contenant des échantillons d'entrées avec les sorties associées pour effectuer des tâches de classification ou de régression, en associant une prédiction aux données d'entrée.

2. Et l'apprentissage **non supervisé** considère des données contenant des échantillons d'entrée sans la prédiction associée. L'algorithme doit alors organiser les données suivant leur corrélation et leur structure. Il est souvent utilisé pour des applications de groupement de données. Les algorithmes supervisés obtiennent généralement de meilleures précisions mais demandent plus de resources de mémoire et de calcul.

La récente multiplication des objets embarqués dans de nombreux domaines a considérablement augmenté la quantité de données à traiter et la complexité des tâches à résoudre sur les systèmes embarqués. Cependant, les systèmes matériels pour le calcul embarqué basés sur l'architecture von Neuman ne sont pas efficace pour traiter cette quantité de données. C'est pourquoi des paradigmes neuromorphiques dotés d'une mémoire distribuée sont étudiés, s'inspirant de la structure et de la représentation de l'information des réseaux de neurones biologiques.

.2 Calcul neuromorphique et réseaux de neurones oscillatoires

Dernièrement, la plupart de la recherche autour des paradigmes neuromorphiques ont exploré les réseaux de neurones à impulsion ou *spiking neural networks* (SNNs) [74], qui s'inspirent des impulsions utilisées pour transmettre l'information dans les réseaux biologiques. Les SNNs encodent l'information temporellement à l'aide d'impulsions pour assurer un calcul de données continues naturel et à faible énergie (voir la Figure 1).

Récemment, les ONNs sont apparu comme un paradigme neuromorphique alternatif pour du calcul temporel, rapide et efficace, à basse consommation. Les ONNs sont des réseaux d'oscillateurs couplés qui émulent les propriétés de calcul collectif des zones du cerveau par le biais d'oscillations. Le paradigme de l'ONN utilise la synchronisation physique entre des oscillateurs couplés. La dynamique des oscillateurs couplés peut être représentée par le modèle de Kuramoto qui exprime la dérivée temporelle de la phase ou de la fréquence des oscillateurs :

$$\frac{d\phi_i}{dt} = \omega_i + \sum_j K_{ij} \sin\left(\phi_j - \phi_i\right) \tag{1}$$

où ω_i est la fréquence de fonctionnement de l'oscillateur. Les termes d'interaction sinusoïdale sont responsables de l'ajustement de la fréquence et modélisent l'adaptation du neurone oscillant i à d'autres neurones oscillants j. Malgré son expression simple, le modèle de Kuramoto produit une dynamique très complexe qui dépend de la connectivité (K_{ij}) et de la distribution des fréquences [171].

La dynamique non linéaire des oscillateurs couplés peut être exploitée de nombreuses manières pour effectuer des tâches intelligentes. Cependant, la plupart des développements de l'ONN se répartissent en deux catégories en fonction du type de codage entrée/sortie : 1) les **ONN basés sur la fréquence** et 2) les **ONNs basés sur la phase** (voir Figure 1).

Dans un **ONN basé sur la fréquence**, des signaux d'entrée dépendants de la fréquence sont injectés dans l'ONN qui réagit aux perturbations d'entrée. Pendant le calcul, les groupes d'oscillateurs se verrouillent en fréquence, ce qui représente une synchronisation. Le calcul dans le domaine fréquentiel peut supprimer certaines connexions physiques entre les oscillateurs [172], mais malgré la mise à l'échelle physique avantageuse, la génération du signal de modulation qui inclut toutes les interactions des oscillateurs par paire est complexe [138, 173]. L'ONN basé dur la fréquence a déjà été utilisé pour le traitement d'images [136, 149, 150], les tâches de mémoire associative [148], ou la classification de voyelles [153, 174].

Dans cette thèse, nous nous concentrons sur **l'ONN basé sur la phase**, dans lequel les oscillateurs ont des fréquences égales, les éléments de couplage sont symétriques ($K_{ij} = K_{ji}$), et les entrées et sorties sont codées dans la relation de phase entre les oscillateurs. Hoppensteadt et Izhikevich [119] ont montré que le calcul d'inférence de l'ONN minimise une fonction

d'énergie dans le temps, tel que :

$$E = -\frac{1}{2}\sum_{i}\sum_{j} K_{ij}\cos\left(\phi_i - \phi_j\right) \tag{2}$$

La dynamique de l'oscillateur dépend alors de l'initialisation de la phase et de l'énergie de l'ONN, ce qui permet une large gamme d'applications.

.3 Motivations de la thèse

La théorie des oscillateurs couplés encourage la recherche autour des ONNs pour des accélérateurs d'IA à faible consommation. En particulier, dans cette thèse nous nous concentrons sur les ONNs calculant en phase, motivés par :

1. la **dynamique physique** qui peut faciliter la conception d'ONNs avec une grande variété d'oscillateurs, analogiques ou numériques, compacts à faible consommation,

2. la **synchronisation parallèle** des oscillateurs couplés permettant une inférence rapide,

3. la **représentation des données** dans le domaine temporel, les phases des oscillateurs limitant l'amplitude de la tension et la consommation d'énergie.

Dans la littérature, l'état de l'art configure un ONN calculant en phase comme un réseau de Hopfield oscillatoire (OHN) avec une architecture d'oscillateurs entièrement couplés pour effectuer de la reconnaissance de formes, comme les réseaux de Hopfield ou *Hopfield neural networks* (HNNs) [20].

L'architecture entièrement couplée génère un grand nombre de synapse et limite l'implementation d'OHN à grande échelle. Par exemple, la plus grande implémentation d'OHN sur puce intègre 100 neurones et est conçu principalement avec des technologies numériques [141], tandis qu'un travail récent a construit un OHN analogique de 30 neurones [161]. En comparaison, [162] a proposé un ONN numérique intégrant 1968 oscillateurs interconnectés dans une topologie en grille pour résoudre des problèmes d'optimisation. Même si une grande partie de la communauté explore de nouveaux matériaux, dispositifs et circuits compacts à faible consommation pour les ONNs [163, 164, 72, 159], nous pensons qu'il est nécessaire d'étudier d'autres architectures pour l'ONN pour aller au-delà de l'OHN. De plus, l'OHN est généralement configuré avec l'algorithme d'apprentissage non supervisé Hebbian, obtenant des résultats de précision non compétitifs par rapport à d'autres modèles exécutant de la reconnaissance de formes [165]. Il est donc également nécessaire d'étudier des solutions d'apprentissage innovantes, d'abord pour essayer d'améliorer les performances de l'ONN sur des tâches de reconnaissance de formes, ensuite pour fournir une capacité d'apprentissage sur puce ou en ligne à l'ONN, et enfin pour élargir le champ des architectures et des applications de l'ONN.

Cette thèse se concentre pour étudier si et comment l'ONN peut résoudre des applications significatives d'IA embarquée à l'aide d'une preuve de concept de l'ONN implémenté en digital sur FPGA.

.4 Implémentation digitale d'un réseau de neurones oscillatoire

Pour étudier et démontrer différentes architectures de l'ONN, algorithmes d'apprentissage et applications, il est nécessaire d'utiliser une implémentation de l'ONN facilement reconfigurable. Généralement, avec les implémentations numériques ou analogiques actuelles d'ONN, les poids synaptiques peuvent être modifiés mais la structure du réseau ne peut pas être changée. Donc, nous utilisons un ONN entièrement numérique implémenté dans une puce FPGA pour pouvoir reconfigurer les poids synaptiques et la structure de l'ONN et pour permettre de déployer facilement des démonstrations sur des systèmes embarqués. L'ONN numérique intègre:

1. des oscillateurs à changement de phase composés d'un registre à état, d'un calculateur de phase et d'un oscillateur contrôlé phase,
2. et un bloc numérique contenant les poids synaptiques implémentés sur 5 bits signés et des circuits logiques arithmétiques pour calculer la somme pondérée des signaux sortants des oscillateurs et définir la nouvelle oscillation d'entrée de chaque oscillateur (voir Figure 2).

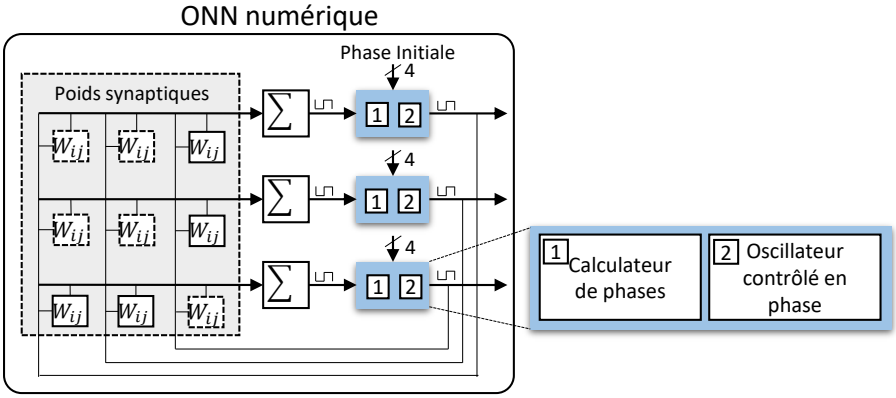


FIGURE 2 – *ONN numérique.*

Dans le Chapitre 2 de ce mémoire, l'ONN numérique est décrit, testé et validé sur des applications de reconnaissances de formes avec des OHNs de petite taille, contenant 5x3 et 10x6 neurones, entraînés avec l'algorithme de Hebbian. Les premiers tests confirment les limites de l'OHN en termes de d'échelle et de précision. En effet, l'architecture de neurones entièrement connectés limite l'implémentation à grande échelle de l'OHN numérique à 120 neurones. De plus, les performances de précisions obtenues pour la reconnaissance de formes avec l'algorithme d'apprentissage de Hebbian sont limitées. Ainsi, nous explorons ensuite de nouvelles architectures et de nouveaux algorithmes d'apprentissage pour aller au-delà de l'OHN en utilisant l'implémentation numérique sur FPGA et nous démontrons ces avancées sur des applications d'IA embarquée.

.5 Amélioration de l'apprentissage d'un OHN

Nous commençons par explorer de nouvelles solutions d'apprentissage pour améliorer les performances de l'OHN pour la reconnaissance de forme. Dans le chapitre 3 nous étudions différents algorithmes d'apprentissages. Dans un premier temps, nous testons des algorithmes

d'apprentissage non supervisés introduits pour la reconnaissance de formes avec les réseaux de Hopfield. Puis, nous proposons un algorithme d'apprentissage supervisé introduit pour des réseaux récurrents multi-couche (voir Figure 3). Nous fournissons également une première solution pour réaliser de l'apprentissage non supervisé d'OHN sur puce. Même si l'apprentissage supervisé augmente la précision de l'OHN, l'OHN configuré avec un apprentissage non supervisé ou supervisé, avec ou sans apprentissage sur puce, ne surpasse pas les autres modèles d'ANN. Nous pensons que les principales limitations en matière de précision sont 1) les sorties binaires de l'OHN, et 2) l'architecture simple-couche. C'est pourquoi nous explorons ensuite de nouvelles architectures.

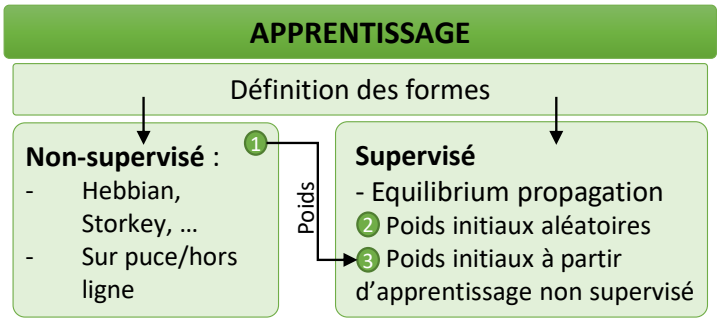


FIGURE 3 – *Différentes méthodes d'apprentissage de l'OHN.*

.6 Amélioration de l'architecture de réseaux de neurones oscillatoire

Pour éviter les limites de précision et d'implémentation à grande échelle de l'OHN, nous étudions de nouvelles architectures pour l'ONN (voir Figure 4). Tout d'abord, dans le Chapitre 4, nous présentons une architecture avec des couches de petits OHN connectés en cascade. Nous divisons une tâche de reconnaissance de formes en sous-tâches afin de réduire le nombre d'éléments synaptiques. Même si nous effectuons des applications d'IA embarquée, il n'est pas facile de les configurer avec les algorithmes d'apprentissages utilisés et cela limitait encore l'ONN à des tâches de reconnaissance de formes. Dans le chapitre 5, nous proposons donc des architectures ONN multi-couche avec des connections bidirectionnelles et unidirectionnelles pour ressembler aux ANN conventionnels et aller au-delà des tâches de reconnaissance des formes. Nous validons d'abord un ONN à 2 couches configuré avec l'algorithme d'apprentissage Hebbian pour faire de la reconnaissance de formes hétérogène, et nous mettons en évidence sa stabilisation uniquement sur des phases de sortie binaires, comme dans l'OHN. Cependant, nous démontrons ensuite qu'un ONN unidirectionnel à 3 couches configuré pour la classification avec un apprentissage supervisé peut se stabiliser sur des sorties multi-phases. Nous utilisons pour cela l'apprentissage supervisé de la rétropropagation de gradient sur un ANN conventionnel se rapprochant de l'ONN et transférons les poids obtenus à l'ONN numérique.Le transfert de poids de l'ANN conventionnel à l'ONN numérique réduit considérablement la précision de la classification, ne permettant pas à l'ONN multi-couche d'être compétitif par rapport à d'autres ANNs. Nous pensons que les principales limitations de la précision sont dues à :

1. l'ONN numérique, avec la précision de ses poids limitée à 5 bits, et ses périodes d'oscillation de 16 phases, qui réduisent les sorties possibles lors de l'utilisation de valeurs multi-phases pour la classification, et qui rendent les classes difficilement différentiables,
2. et l'algorithme d'apprentissage par rétropropagation de gradient appliqué à un ANN conventionnel qui peut ne pas correspondre parfaitement à l'ONN.

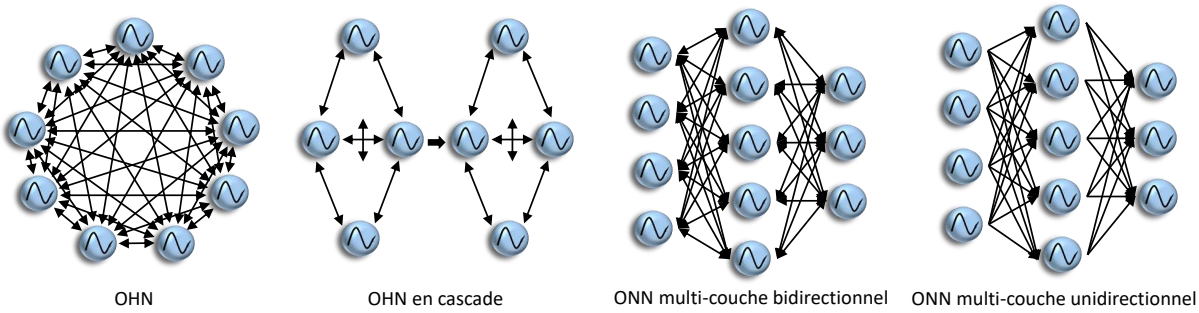


FIGURE 4 – *Différentes architectures de l'ONN étudiées.*

.7 Applications de l'ONN pour l'IA embarquée

Nous démontrons l'efficacité de l'ONN avec différentes architectures et différents algorithmes d'apprentissage, en réalisant diverses applications pour l'IA embarquée (voir Figure 5). Nous validons dans un premier temps la conception numérique de l'OHN dans le chapitre 2 en effectuant une reconnaissance de chiffres en temps réel à partir d'un flux de données provenant d'une caméra. Ensuite, dans le chapitre 4, nous présentons l'architecture de l'OHN en cascade pour une application d'évitement d'obstacles en temps réel en traitant des données de capteurs de proximité positionnés sur des robots mobiles, tels que le robot industriel E4 d'A.I.Mergence. Nous proposons également une conception analogique de l'architecture OHN en cascade et l'utilisons pour effectuer une détection de contour dans une image, en remplaçant les filtres de convolution habituels. Même si cette première solution de détection de contour d'image atteint une précision intéressante, le temps de calcul est trop long. C'est pourquoi, au chapitre 5, nous implémentons numériquement un ONN à deux couches configuré pour la détection de contours d'une image capable de traiter une image en temps réel grâce au parallélisme du FPGA. Nous utilisons cette implémentation pour accélérer l'algorithme d'extraction d'amers SIFT. Enfin, nous étudions comment résoudre des tâches de classification avec l'ONN. Tout d'abord, dans le chapitre 3, nous utilisons l'ONN pour résoudre la classification d'images simplifiées de MNIST, en transformant une application de classification d'images en une tâche de reconnaissance de formes. Ensuite, dans le chapitre 5, nous configurons un ONN à trois couches unidirectionnelles pour résoudre la classification des données de Yin-Yang. Actuellement, la précision de classification de l'ONN n'est pas supérieure à celle des ANNs conventionnels, certainement en raison de la précision limitée de la conception numérique et des solutions d'apprentissage non adaptées.

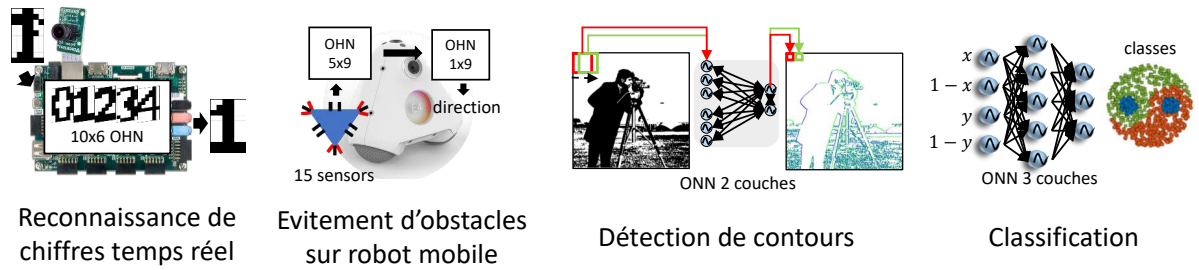


FIGURE 5 – *Différentes applications démontrées avec l'ONN.*

.8 Conclusion et perspectives

Avec les architectures multicouches proposées, formées à l'aide d'algorithmes d'apprentissage non supervisés et supervisés, et mises en oeuvre dans l'ONN numérique, le paradigme de l'ONN n'est pas encore compétitif avec les modèles ANN multicouches conventionnels en termes de précision. Ainsi, des recherches futures sont nécessaires pour améliorer la précision des ONN multicouches, mais aussi pour explorer d'autres architectures et applications pour l'ONN. Les pistes majeures d'améliorations sont

1. l'étude de solution pour permettre à l'ONN de se stabiliser sur plusieurs phases au lieu de phases binaires comme c'est le cas pour le moment,
2. l'exploration de nouveaux algorithmes d'apprentissages compatibles avec les ONNs multicouche et multi-phase,
3. le dévelopemment de nouveaux composants compacts, rapides et à faible consommation pour la conception d'ONNs analogiques afin de bénéficier des avantages du domaine analogique en terme d'efficacité energétique et temporel, et
4. la recherche d'architectures et d'applications alternatives, telles que l'architecture en réservoir pour le traitement de données temporelles, et les architectures en graphes pour la résolution de problèmes d'optimisation.