UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

Wendell Fioravante da Silva Diniz

Conception and realization of an FPGA-based framework for embedded systems applied to Optimum-path Forest classifier

Concepção e realização de um framework *para sistemas embarcados baseados em FPGA aplicado a um classificador Floresta de Caminhos Ótimos*

CAMPINAS
2017

Wendell Fioravante da Silva Diniz

Conception and realization of an FPGA-based framework for embedded systems applied to Optimum-path Forest classifier

Concepção e realização de um framework *para sistemas embarcados baseados em FPGA aplicado a um classificador Floresta de Caminhos Ótimos*

Thesis presented to the School of Mechanical Engineering of University of Campinas in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the area of Solid Mechanics and Mechanical Design and for the degree of *Docteur de l'UTC* in the area of Computer Engineering, in the context of a Co-supervising Agreement signed between Unicamp and *Sorbonne Universités - Université de Technologie de Compiégne.*

Tese apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas, como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Mecânica na área de Mecânica dos Sólidos e Projeto Mecânico e para o título de *Docteur de l'UTC* na área de Engenharia de Computação, no âmbito do Acordo de Cotutela firmado entre a Unicamp e a *Sorbonne Universités - Université de Technologie de Compiègne.*

ESTE ARQUIVO DIGITAL CORRESPONDE À VERSÃO FINAL DA TESE DEFENDIDA PELO ALUNO WENDELL FIORAVANTE DA SILVA DINIZ E ORIENTADA POR: PROF. DR. EURÍPEDES GUILHERME DE OLIVEIRA NÓBREGA, PROF. DR. ISABELLE FANTONI-COICHOT E PROF. DR. VINCENT FRÉMONT.

Supervisor: Eurípedes Guilherme de Oliveira Nóbrega, Ph.D.
Supervisor: Isabelle Fantoni-Coichot, Ph.D.
Co-supervisor: Vincent Frémont, Ph.D.

Campinas
2017

Agência(s) de fomento e nº(s) de processo(s): CAPES, 13.077/2013-09
ORCID: <http://orcid.org/0000-0002-8398-6631>

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

	Diniz, Wendell Fioravante da Silva, 1982-
D615c	Conception and realization of an FPGA-based framework for embedded systems applied to Optimum-path Forest classifier / Wendell Fioravante da Silva Diniz – Campinas, SP: [s.n.], 2017.
	Orientadores: Eurípedes Guilherme de Oliveira Nóbrega e Isabelle Fantoni-Coichot. Coorientador: Vincent Frémont. Tese (Doutorado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica. Em cotutela com: Sorbonne Universités - Université de Technologie de Compiègne.
	1. FPGA (Field Programmable Gateway Array). 2. Sistemas embarcados (Computadores). 3. Aprendizado de máquina. 4. Reconhecimento de padrões. 5. Floresta de caminhos ótimos. I. Nóbrega, Eurípedes Guilherme de Oliveira, 1950-. II. Fantoni-Coichot, Isabelle. III. Frémont, Vincent. IV. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. VI. Título.

Informações para Biblioteca Digital

Título em outro idioma: Concepção e realização de um *framework* para sistemas embarcados baseados em FPGA aplicado a um classificador Floresta de Caminhos Ótimos
Palavras-chave em inglês:
FPGA (Field Programmable Gateway Array)
Embedded computer systems
Machine learning
Pattern Recognition
Optimum-path forest
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Doutor em Engenharia Mecânica
Banca examinadora:
Eurípedes Guilherme de Oliveira Nóbrega[Orientador]
Vincent Frémont
Luiz Carlos Sandoval Góes
Osamu Saotome
Denis Silva Loubach
Alain Mérigot
Philippe Xu
Data da defesa: 23-03-2017
Programa de Pós Graduação: Engenharia Mecânica

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA
COMISSÃO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
DEPARTAMENTO DE MECÂNICA COMPUTACIONAL

TESE DE DOUTORADO ACADÊMICO

Conception and realization of an FPGA-based framework for embedded systems applied to Optimum-path Forest classifier

Concepção e realização de um framework *para sistemas embarcados baseados em FPGA aplicado a um classificador Floresta de Caminhos Ótimos*

Autor: Wendell Fioravante da Silva Diniz
Orientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega
Orientador: Prof. Dr. Isabelle Fantoni-Coichot
Co-orientador: Prof. Dr. Vincent Frémont

A Banca Examinadora composta pelos membros abaixo aprovou esta tese:

Prof. Dr. Luiz Carlos Sandoval Góes
Instituto Tecnológico de Aeronáutica - ITA

Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega
Univ. Estadual de Campinas

Prof. Dr. Vincent Frémont
Université de Technologie de Compiègne

Prof. Dr. Osamu Saotome
Instituto Tecnológico de Aeronáutica - ITA

Prof. Dr. Denis Silva Loubach
Univ. Estadual de Campinas

Prof. Dr. Alain Mérigot
Université Paris-Sud

Prof. Dr. Philippe Xu
Université de Technologie de Compiègne

A Ata da defesa com as respectivas assinaturas dos membros encontra-se no processo de vida acadêmica do aluno.

Campinas, 23 de março de 2017

Abstract

Many modern applications rely on Artificial Intelligence methods such as automatic classification. However, the computational cost associated with these techniques limit their use in resource constrained embedded platforms. A high amount of data may overcome the computational power available in such embedded environments while turning the process of designing them a challenging task. Common processing pipelines use many high computational cost functions, which brings the necessity of combining high computational capacity with energy efficiency.

One of the strategies to overcome this limitation and provide sufficient computational power allied with low energy consumption is the use of specialized hardware such as Field Programmable Gateway Arrays (FPGAs). This class of devices is widely known for their performance to consumption ratio, being an interesting alternative to building capable embedded systems.

This thesis proposes an FPGA-based framework for performance acceleration of a classification algorithm to be implemented in an embedded system. Acceleration is achieved using Single Instructions, Multiple Data (SIMD)-based parallelization scheme, taking advantage of FPGA characteristics of fine-grain parallelism. The proposed system is implemented and tested in actual FPGA hardware. For the architecture validation, a graph-based classifier, the Optimum-path Forest (OPF), is evaluated in an application proposition and afterward applied to the proposed architecture. The OPF study led to a proposition of a new learning algorithm using evolutionary computation concepts, aiming at classification processing time reduction, which combined to the hardware implementation offers sufficient performance acceleration to be applied in a variety of embedded systems.

Keywords: FPGA (Field Programmable Gateway Array); Embedded computer systems; Machine learning; Pattern Recognition; Optimum-path forest.

Resumo

Muitas aplicações modernas dependem de métodos de Inteligência Artificial, tais como classificação automática. Entretanto, o alto custo computacional associado a essas técnicas limita seu uso em plataformas embarcadas com recursos restritos. Grandes quantidades de dados podem superar o poder computacional disponível em tais ambientes, o que torna o processo de projetá-los uma tarefa desafiadora. As condutas de processamento mais comuns usam muitas funções de custo computacional elevadas, o que traz a necessidade de combinar alta capacidade computacional com eficiência energética.

Uma possível estratégia para superar essas limitações e prover poder computacional suficiente aliado ao baixo consumo de energia é o uso de hardware especializado como, por exemplo, Field Programmable Gateway Arrays (FPGAs). Esta classe de dispositivos é amplamente conhecida por sua boa relação desempenho/consumo, sendo uma alternativa interessante para a construção de sistemas embarcados eficazes e eficientes.

Esta tese propõe um *framework* baseado em FPGA para a aceleração de desempenho de um algoritmo de classificação a ser implementado em um sistema embarcado. A aceleração do desempenho foi atingida usando o esquema de paralelização Single Instructions, Multiple Data (SIMD), aproveitando as características de paralelismo de grão fino dos FPGAs. O sistema proposto foi implementado e testado em hardware FPGA real. Para a validação da arquitetura, um classificador baseado em Teoria dos Grafos, o Optimum-path Forest (OPF), foi avaliado em uma proposta de aplicação e posteriormente implementado na arquitetura proposta. O estudo do OPF levou à proposição de um novo algoritmo de aprendizagem para o mesmo, usando conceitos de Computação Evolutiva, visando a redução do tempo de processamento de classificação, que, combinada à implementação em hardware, oferece uma aceleração de desempenho suficiente para ser aplicada em uma variedade de sistemas embarcados.

Palavras-chave: FPGA (Field Programmable Gateway Array); Sistemas embarcados (Computadores); Aprendizado de máquina; Reconhecimento de padrões; Floresta de caminhos ótimos.

Résumé

De nombreuses applications modernes s'appuient sur des méthodes d'Intelligence Artificielle telles que la classification automatique. Cependant, le coût de calcul associé à ces techniques limite leur utilisation dans les plates-formes embarquées contraintes par les ressources. Une grande quantité de données peut surmonter la puissance de calcul disponible dans de tels environnements embarqués, transformant le processus de concevoir une tâche difficile. Les pipelines de traitement courants utilisent de nombreuses fonctions de coût de calcul élevé, ce qui amène la nécessité de combiner une capacité de calcul élevée avec une efficacité énergétique.

Une des stratégies pour surmonter cette limitation et fournir une puissance de calcul suffisante alliée à la faible consommation d'énergie est l'utilisation de matériel spécialisé tel que Field Programmable Gateway Arrays (FPGAs). Cette classe de dispositifs est largement connue pour leur rapport performance/consommation, étant une alternative intéressante à la construction de systèmes embarqués capables.

Cette thèse propose un *framework* basé sur FPGA pour l'accélération de la performance d'un algorithme de classification à implémenter dans un système embarqué. L'accélération est réalisée en utilisant le système de parallélisation basé sur Single Instructions, Multiple Data (SIMD), en tirant parti des caractéristiques de parallélisme à grain fin presentées pour les FPGA. Le système proposé est implémenté et testé dans un plate-forme actuel de dévelloppement FPGA. Pour la validation de l'architecture, un classificateur basé sur la théorie des graphes, l'Optimum-path Forest (OPF), est évalué dans une proposition d'application ensuite realisé dans l'architecture proposée. L'étude de l'OPF a conduit à la proposition d'un nouvel algorithme d'apprentissage pour l'OPF, en utilisant des concepts de calcul évolutifs, visant à réduire le temps de traitement de la classification, combiné à la mise en œuvre matérielle offrant une accélération de performance suffisante pour être appliquée dans une variété de systèmes embarqués.

Mots-clés: FPGA (Field Programmable Gateway Array); Systèmes embarqués; Apprentissage mécanique; Reconaissance des formes; Optimum-path Forest .

List of Figures

1.1 The Bebop drone is an modern advanced embedded system 21
1.2 The NVIDIA® Drive™ PX 2 board 23
1.3 Audi AG Central Driver Assistance Controller (zFAS) system 24
1.4 Overview of the main proposed architecture 25

2.1 An FPGA internal architecture 33
2.2 Basic logic cell architecture 34
2.3 OpenCL Platform Model hierarchy 37
2.4 OpenCL architecture Memory Model hierarchy 39
2.5 Comparison between Open Computing Language (OpenCL) and Register Transfer Level (RTL) workflows 42
2.6 OPF Training visualisation 47
2.7 OPF classification visualisation 48

3.1 Road traffic mortality chart 52
3.2 A pedestrian detection system 55
3.3 The HOG descriptor 57
3.4 Samples of pedestrian detection dataset 60
3.5 Metrics for classifying HOG descriptors 61
3.6 Metrics for classifying HOG+PCA descriptors. 62
3.7 Receiver Operating Characteristic space 64
3.8 Accuracy histogram showing classification stability 65

4.1 Self-organizing map training procedure 71
4.2 Growing Self-organizing Map learning procedure 72
4.3 Overview of the SOEL algorithm structural hierarchy 73
4.4 Node adjustment process 76
4.5 Node spawning process 77
4.6 Neigborhood radius function 79
4.7 Node adjustment factor ψ 80
4.8 Learning rate function 81

5.1 Parallel processing architecture overview of the proposed framework 87
5.2 Elementary Processor Array 89
5.3 Elementary Processor architecture organization 90
5.4 Host code task organization 91
5.5 Kernel data distribution 94
5.6 The Arrow SoCKit development board 95

List of Tables

2.1 Graph matching based applications published up to 2002, according to Conte et al. (2004). 43

3.1 Pedestrian mortality rates in USA, from 2004 to 20013 52
3.2 Training and testing stages processing times using only HOG descriptors. 63
3.3 Training and testing stages processing times for PCA+HOG descriptors. 63

4.1 Dataset descriptions 82
4.2 OPF learning algorithms comparison 83

5.1 Dataset descriptions 97
5.2 Accuracy and classification times for software (S) and hardware (H) versions of the OPF classifier 98
5.3 Processing time reduction with combined SOEL+hardware acceleration 98
5.4 Final peak power consumption for the implemented architecture 98

List of Algorithms

2.1 OPF training algorithm 46
2.2 Enhanced OPF classification algorithm 49
2.3 OPF learning procedure 50
4.1 Self-organizing Evolutionary Learning for OPF classifier training 75

List of Abbreviations and Acronyms

ADAS Advanced Driver Assistance Systems
AI Artificial Intelligence
ANN Artificial Neural Network
API Application Programming Interface
ARM Advanced RISC Machine
ASIC Application Specific Integrated Circuit

BMU Best Matching Unit

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CLM Control Logic Module
CNN Convolutional Neural Network
CNRS Centre National de Recherche Scientifique
CPU Central Processing Unit
CU Compute Unit
CV Computer Vision

DDR2 Double Data Rate, level 2
DDR3 Double Data Rate, level 3
DMA Direct Memory Access
DMC Departamento de Mecânica Computacional
DRAM Dynamic RAM
DSP Digital Signal Processor

ECG Electrocardiogram
EP Elementary Processor
EPA Elementary Processors Array
ESA European Space Agency

FEM Faculdade de Engenharia Mecânica

FPGA Field Programmable Gateway Array
FPS Frames per Second

GB Gigabyte
GE Global Expert
GHz Gigahertz
GM Global Memory
GPU Graphics Processing Unit
G-SOM Growing Self-organizing Map

HDL Hardware Description Language
Heudiasyc Heuristique et Diagnose de Systèmes Complexes
HOG Histogram of Oriented Gradients
HP Host Processor
HPS Hard Processor System

IDE Integrated Development Environment
IFT Image Foresting Transform
IO Input/Output
IP Intellectual Property

kb Kilobits

LE Local Expert
LPM Local Private Memory
LSM Local Shared Memory
LUT Look-up Table

MB Megabyte
MER Mars Exploration Rover
MHz Megahertz
MIMD Multiple Instructions, Multiple Data
MISD Multiple Instructions, Single Data

ML	Machine Learning
MLP	Multi-layer Perceptron
MM	Memory Management
MOE	Mixture-Of-Experts
MP	Mega Pixels
MST	Minimum Spanning Tree
MWR	Microwave Radiometer
NASA	National Aeronautic and Space Administration
OpenCL	Open Computing Language
OpenCV	Open Computer Vision
OpenGL	Open Graphics Language
OPF	Optimum-path Forest
OTP	One Time Programmable
PCA	Principal Component Analysis
PE	Processing Element
PLD	Programmable Logic Device
PP	Parallel Processor
PS	Prototypes as Seeds
PSP	Prototypes as Seeds Policy
RAM	Random Access Memory
ROI	Region of Interest
ROM	Read-only Memory
RPROP	Resilient Propagation
RS	Random Seeds
RSP	Random Seeds Policy
RTL	Register Transfer Level
SBC	Single Board Computer
SDK	Software Development Kit

SDRAM Synchronous Dynamic RAM

SEU Single Event Upset

SIMD Single Instructions, Multiple Data

SISD Single Instructions, Single Data

SoC System on a Chip

SOEL Self-Organizing Evolutionary Learning

SOFM Self-organizing Feature Map

SOM Self-organizing Map

SPS Soft Processor System

SPSD Single Program Multiple Data

SRAM Static RAM

SVM Support Vector Machines

TDP Thermal Design Power

TFLOPS Tera Floating-point Operations per Second

UAV Unmanned Aerial Vehicle

UNICAMP Universidade Estadual de Campinas

USA United States of America

USB Universal Serial Bus

UTC Université de Technologie de Compiègne

VHDL VHSIC Hardware Description Language

VHSIC Very High Speed Integrated Circuit

zFAS Central Driver Assistance Controller

Summary

1 Introduction . . . **18**
1.1 Context and motivations . . . 20
1.2 Data to information general process . . . 22
1.3 Machine Learning applications for embedded systems . . . 22
1.4 Proposed approach overview . . . 25
1.5 Summary of contributions . . . 28
1.6 Document outline . . . 29

2 Fundamental concepts . . . **31**
2.1 Embedded design with FPGAs . . . 31
2.2 OpenCL as an option for FPGA development . . . 35
2.3 The Optimum-path Forest classifier . . . 41
2.3.1 Supervised learning variation . . . 45

3 Implementation of an OPF-based pedestrian detection system . . . **51**
3.1 Introduction . . . 51
3.2 Current approaches on Pedestrian Detection . . . 53
3.3 System overview . . . 54
3.4 Feature extraction . . . 55
3.4.1 Histogram of Oriented Gradients . . . 55
3.4.2 Principal Component Analysis . . . 57
3.5 Experimental results . . . 58
3.5.1 Methods used for comparison . . . 58
3.5.2 Data set description . . . 59
3.5.3 Results . . . 60
3.6 Conclusion . . . 64

4 Self-organizing Evolutionary learning for supervised OPF training **67**
4.1 Introduction . . . 67
4.2 Self-organizing maps . . . 70
4.3 Method description . . . 72
4.3.1 Node adjustment determination . . . 78

4.4 Experimental results . . . 81
4.4.1 Metrics and performance indicators . . . 81
4.4.2 Datasets description . . . 82
4.4.3 Results . . . 82
4.5 Conclusions . . . 83

5 FPGA based framework for classification with OPF . . . 85
5.1 Introduction . . . 85
5.2 High level system design . . . 86
5.2.1 Host Processor . . . 87
5.2.2 Parallel Processor . . . 88
5.3 System realization . . . 91
5.3.1 Host Processor code organization . . . 91
5.3.2 Parallel Processor code organization . . . 93
5.3.3 Hardware platform specifications . . . 94
5.4 Experimental results . . . 95
5.4.1 Hardware and software specifications . . . 95
5.4.2 Metrics and performance indicators . . . 96
5.4.3 Dataset descriptions . . . 96
5.4.4 Performance analysis . . . 97
5.5 Conclusions . . . 99

6 General conclusion . . . 100
6.1 Key contributions . . . 100
6.2 Future perspectives . . . 101

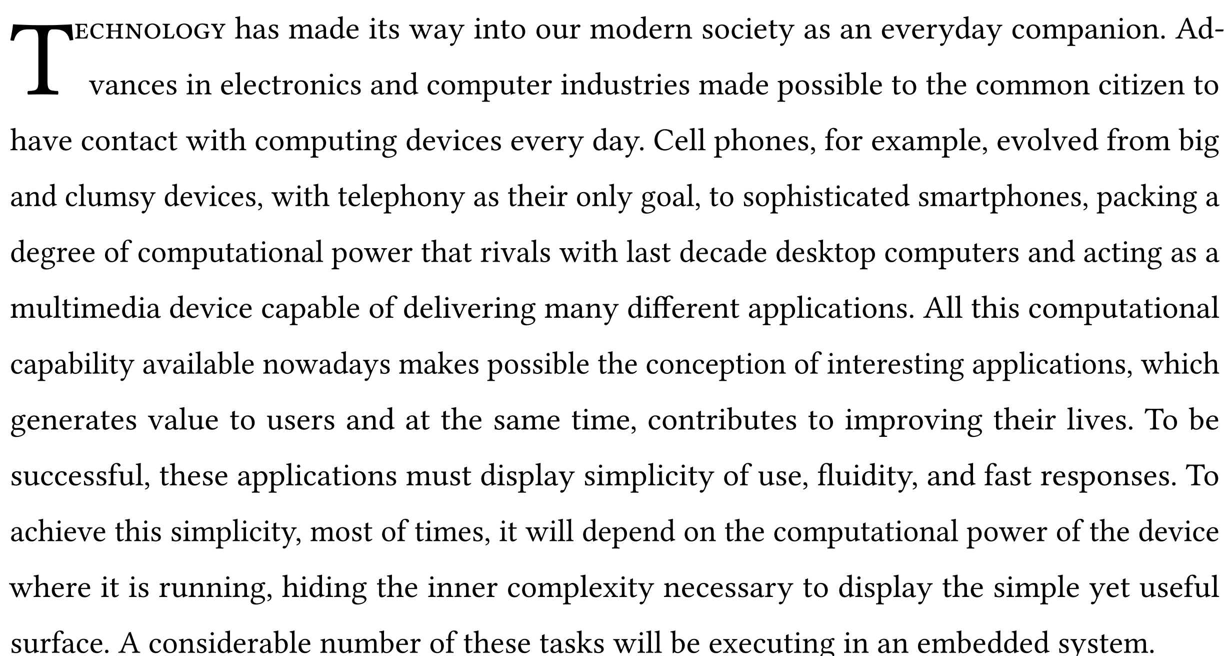
Bibliography . . . 104

Chapter 1

Introduction

"Start by doing what is necessary, then do what is possible, and suddenly you are doing the impossible."

— St. Francis of Assisi



Technology has made its way into our modern society as an everyday companion. Advances in electronics and computer industries made possible to the common citizen to have contact with computing devices every day. Cell phones, for example, evolved from big and clumsy devices, with telephony as their only goal, to sophisticated smartphones, packing a degree of computational power that rivals with last decade desktop computers and acting as a multimedia device capable of delivering many different applications. All this computational capability available nowadays makes possible the conception of interesting applications, which generates value to users and at the same time, contributes to improving their lives. To be successful, these applications must display simplicity of use, fluidity, and fast responses. To achieve this simplicity, most of times, it will depend on the computational power of the device where it is running, hiding the inner complexity necessary to display the simple yet useful surface. A considerable number of these tasks will be executing in an embedded system.

A general definition of embedded systems could be: "embedded systems are computing systems with tightly coupled hardware and software integration, that are designed to perform a dedicated function." (LI; YAO, 2003). However, considering the computational power available for embedded systems nowadays, this definition can be extended in the sense that the dedicated embedded function may encompass several complex subsystems and runtime routines. Therefore, embedded systems take a variety of forms, ranging from industrial controllers, medical devices, and personal devices to space exploration and field research. Comparing with general purpose microprocessors, like the ones found in desktop computers, the devices that are used in general for embedded systems lack raw computational power. Nonetheless, embedded microprocessors must generally meet size and power consumption requirements, which is the factor that classifies them as low power devices. And yet, many modern applications still

demand embedded systems capable of solving complex tasks without breaking the foretold requirements. Industry has always searched for developing new devices capable to completely fulfill these specifications.

With the emerging of these new technologies, embedded systems also evolved and started to cover new innovative applications. We are now familiar with mobile robots, as small drones are seen everywhere. Large ones have been used by many institutions for vigilance, reporting or research. Autonomous vehicles are close to being released to the public. In space, our robotic explorers unravel new discoveries, increasing our understanding of the universe and our role in it. All these innovative applications share one thing in common: They depend on efficient embedded systems. Therefore, researching suitable techniques to explore these systems capabilities to the maximum are of prime interest.

Nowadays, the daily produced quantity of information around the world is huge. According to IBM, in 2014, Humanity was able to produce every day, 2.5 quintillion bytes of data – so much that 90% of the data in the world today has been created in the last two years alone (IBM, 2014). All these data need to be processed to produce meaningful information. Logically, it is impossible to trust this data to human processing, thus, we must get computers to be able to do so. Artificial Intelligence (AI) is one field that makes this possible. Among the many techniques and methods under AI hood, Machine Learning (ML) is one that has received crescent interest lately. Defined as the set of methods that gives the computer capacity to learn from data, without being explicitly programmed to do so. These techniques aid computers to reach the ability to find the relevant information from all that amount of data.

Perhaps the most frequent task in ML is data classification. It allows computers to identify classes of objects from data acquired by a range of sensors. For example, an autonomous vehicle needs to identify the road it is running on, other vehicles in transit, people, and transit signs, all in order to build and follow its navigation plan. The data can be obtained by a camera and after some processing, ML can be applied to identify each class of the objects showing in the image. This task is amazingly well done by the human brain, but equally difficult to implement on computers. Nonetheless, modern approaches are able to get close to brain performance, but as one can imagine, they require a considerable amount of computational power, which makes them hard to apply to embedded systems. However, some implementations depend on embedded systems , requiring development effort to design solutions capable of realizing those

difficult tasks.

This chapter introduces the problem of ML applied to embedded systems. An efficient implementation can have many applications, like mobile robotics, human-machine interaction, automatic data analysis for personal or commercial applications, and intelligent autonomous vehicles. It starts with the context and motivations that lead to this work development in Section 1.1 followed by an overview of the general process to produce information from gathered data used as the guideline in this work in Section 1.2. An introduction on ML applications for embedded systems is given in Section 1.3. Section 1.4 presents an overview of the proposed approach for this works implementations and Section 1.5 closes the chapter with a summary of this work contributions and published works.

1.1 Context and motivations

This thesis is part of a doctorate program with a joint supervision agreement between the Department of Computational Mechanics of the Faculty of Mechanical Engineering from University of Campinas (DMC/FEM/UNICAMP) in São Paulo, Brazil, and the Heudiasyc Laboratory, UMR 7253 CNRS/UTC at the Sorbonne Universités, Université de Technologie de Compiègne (UTC) (Heudiasyc/CNRS/UTC) in Compiègne, France.

Universidade Estadual de Campinas (UNICAMP) and UTC have been working jointly in the context of the *Brafitec* project (GELAS et al., 2007), an exchange program for French and Brazilian Engineering undergraduate students. During a visit of the program coordinator at UTC, prof. Alessandro CORREA VICTORINO, a proposition for scientific collaboration for graduate students, relating to Research and Technological Development was made. Following the proposition, UTC received at Heuristique et Diagnose de Systèmes Complexes (Heudiasyc) the first exchange Ph.D. candidate in 2008. Since then, three more joint supervision thesis projects were accepted, with the current thesis being the third to be in conclusion. The know-how and excellence of Heudiasyc laboratory, currently labeled as part of Labex MS2T and ROBOTEX projects was one of the determining factors to proceed with the partnership, increasing the level of France-Brazil scientific relationship.

This research received a scholarship from Coordination for Improvement of Higher

Education Personnel (CAPES) under the process nº 13.077/2013-9 from Brazilian Government. Most of the experimental part of this work was carried out and funded in the framework of the Labex MS2T (Reference ANR-11-IDEX-0004-02), from French Government, through the program "Investments for the future" managed by the National Agency for Research in France.

The motivations for this research are born from the growing necessity of advanced embedded systems in many fields. However, the increasing performance of such systems is often accompanied by an increase in power consumption, leading to the use of heavier power sources. Depending on the type of application, this can be a severe drawback. Let us take an example: a small quadcopter drone equipped with a camera, like the one shown in Figure 1.1.

Figure 1.1 – The Bebop drone made by Parrot Inc. is an example of an advanced embedded system. The drone is equipped with a high definition camera that can transmit video over a wireless connection. Its control board packs a significant amount of intelligence, making it easy to operate by minimally skilled people. The control and video algorithms are quite demanding, a fact that is perceived from its relatively short autonomy, 25 minutes, according to the manufacturer.
Source: <www.parrot.com>

This setup can be used in many applications, like surveillance of a restricted area. It could search for people inside the perimeter and fly to a location where it could take a good shot of the trespasser, and then send it to be identified and then answer for his acts later. Such an application would demand capabilities to identify a person, predict a suitable flight route and send the retrieved data over a wireless connection. An ML algorithm could be used to identify the detected object as a person or not, and based on this recognition, proceed with the corresponding action.

This is one among many examples of how significant applications can be accomplished by using ML in embedded systems. Next Section will present a survey of such applications.

1.2 Data to information general process

Considering the applications that were previously mentioned in this chapter, it is clear that the main difficulty for implementing ML into embedded systems is achieving the computational power to meet hard real-time requirements in resource-constrained systems. This limitation greatly restrains the applications that can be built in such systems. An ML system usually is structured around three basic tasks:

Data preparation: Data acquired by the sensors rarely comes in a directly usable format. They require treatment before they can be effectively used by the ML algorithm. This process adds a layer of processing that must be incorporated in the platform design.

Data processing: After being prepared, the data can now be properly used by the ML algorithm. This is usually the most costly process, in terms of computational power, then it can be identified as the critical task. This stage performance can severely affect the other tasks sharing the resources.

Data presentation: After data processing, the derived information can be used to actually complete the main task.

Meeting hard real-time requirements with just these tasks alone is already a challenge itself. Furthermore, there are other tasks that will sum up and compete for processing time.

1.3 Machine Learning applications for embedded systems

ML has been used in a vast range of applications that can be implemented in embedded systems. For example, recently, NVIDIA™ has launched the Drive™ PX 2, shown in Figure 1.2, which is a Single Board Computer (SBC) aimed for autonomous vehicles development. Besides interfaces for several types of sensors, it includes a pair of Tegra® X1 System on a Chip (SoC),

alongside a middleware offering ML solutions. The SoCs can be programmed with CUDA™ , the proprietary language for general purpose programming of the Graphics Processing Units (GPUs) brand. The language offers the GPUs parallel processing characteristics to be explored for all kinds of computations, aside from graphics processing, and have been used worldwide since its debut in 2008.

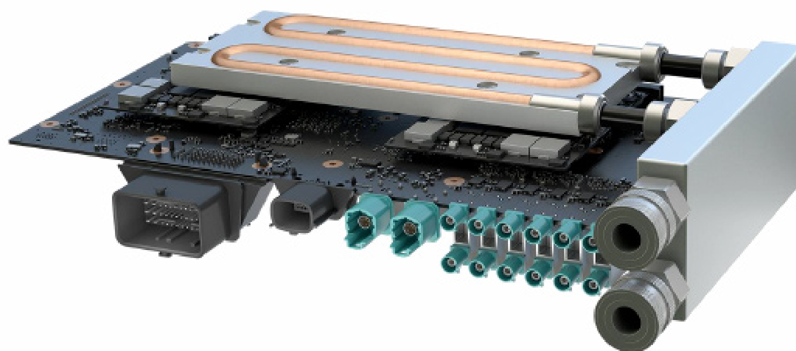


Figure 1.2 – The Drive™ PX 2 board is NVIDIA™ 's response to an emerging marketing for ML capable embedded systems. Equipped with a pair of Tegra® X1 SoCs and a middleware with integrated solutions it offers up to 2.3 TFLOPS of processing power. Notice the liquid cooling solution necessary due to the 250 W TDP.
Source: <http://www.clubic.com/>

Clearly visible in Figure1.2, the pipes of the liquid cooling system required to dissipate the heat generated by the SoC. Although GPUs have been widely used as ML computational platform, they are still power hungry devices. Even the embedded version used in the PX 2 most capable configuration has a Thermal Design Power (TDP) of 250 W, requiring that liquid cooling solution (ANANDTECH, 2016). These requirements severely restrain the range of devices that can benefit from the board.

Still, in autonomous and semi-autonomous vehicles field, some players have chosen a different platform. In order to integrate the many different tasks needed to bring together a functional system, Audi AG contracted Austrian group *TTTech Computertechnik AG* to design and manufacture their Central Driver Assistance Controller (zFAS) system, presented in Figure 1.3 (Safe Car News, 2015).

The board makes use of an Altera® Cyclone V to perform the heavy data fusion from the many sensors installed around the vehicle. The data is then integrated to provide a range

Figure 1.3 – Audi AG has chosen Austrian TTTech group to supply their zFAS system. The board uses a Cyclone™ V FPGA alongside a Tegra™ processor to capture and fuse data from various sensors. This data is used to provide ADAS services to the driver. The technology is present in the newer Audi A8 car.
Source: Wikimedia Commons.

of Advanced Driver Assistance Systems (ADAS) services to help the driver to get a better situational awareness of the environment, improving safety for him and the other characters the vehicle interacts while in transit.

Space exploration is another area that has been using Field Programmable Gateway Arrays (FPGAs) to improve their computational power while keeping low power consumption. In fact, satellites and space probes have probably the harshest design requirements, as they operate in extreme environments, exposed to a varied range of harms. Low temperatures, radiation, power surges, and impacts are common. Therefore, increasing their computational power without exceeding power, thermal dissipation and weight restrictions is an imperative design choice.

The European Space Agency (ESA) made extensive studies about the feasibility of FPGA for space applications as early as 2002 (HABINC, 2002). The study concentrated on the resistance and mitigation strategies to Single Event Upsets (SEUs) that Static RAM (SRAM)-based reprogrammable FPGAs would be exposed due to space radiation. One proven FPGA user in space application is the USA agency National Aeronautic and Space Administration (NASA). One very well known mission to use FPGAs as processing unit were the Mars Exploration Rovers (MERs) Spirit and Opportunity. They used four Xilinx XQR4062XL FPGAs to control the lander pyrotechnics that was crucial to the Entry, Descent and Landing phases of the mission (PINGREE, 2010). The more recent Curiosity rover pushed the limits even far, being equipped with 31 FPGAs. Also, the Juno mission, that reached Jupiter recently this year, was equipped with an One Time Programmable (OTP) FPGA in one key instrument, the Microwave

Radiometer (MWR). Although OTP FPGAs offers the advantages of hardware acceleration, they cannot be updated later.

With their robustness and capabilities proven in a varied range of applications with different requirements, but having in common a demand for computational power alongside low power consumption, FPGAs have shown to be an interesting choice for accelerating embedded ML applications.

1.4 Proposed approach overview

The proposed approach for embedded ML is to use an FPGA-based architecture to provide hardware acceleration by specialized process encoding and parallelism. The specialized hardware built into FPGA fabric encompasses the parallelization of the chosen algorithm and will be used as an auxiliary processor. A host processor will carry on general tasks including data preparation and program flow control and also control the auxiliary processor. They will communicate with each other through a channel. Figure 1.4 shows this organization.

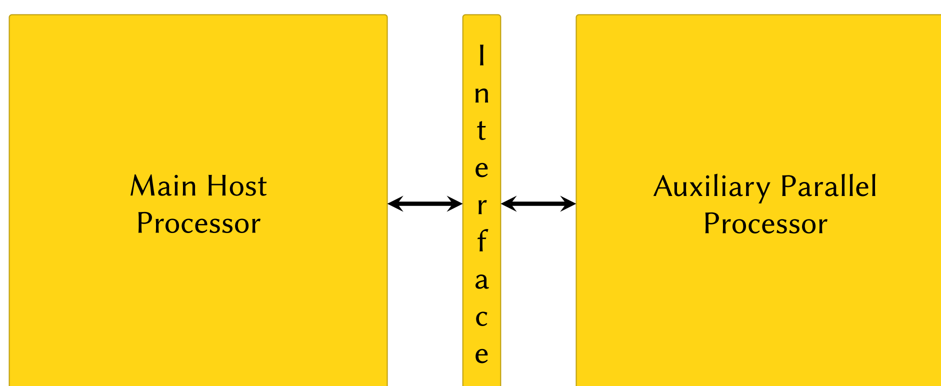


Figure 1.4 – The proposed architecture uses a host processor to perform data preparation and an auxiliary parallel processor to do the heavy processing. A communication interface between them manage the exchange of data and commands.

In parallel computing, granularity refers to the amount of work executed in a computational task over a parallel processing system. It can be also perceived as the ratio between the computation time of a task and the communication time required to exchange data among the parallel elements (HWANG; KAI, 1992; KWIATKOWSKI, 2002). Parallel algorithms can be classified according to their granularity as coarse, medium or fine-grained. Coarse-grained

algorithms are those in which the program is divided into large tasks. Usually, this kind is performed in distributed systems using message-passing architectures to communicate data. Medium-grained algorithms are those in which there is a compromise in the computation to communication ratio, and it encompasses the majority of general-purpose parallel computers (MILLER; STOUT, 1997). Finally, fine-grained algorithms are those where the task is evenly divided among a large number of processing elements, hence, load-balancing plays an important role. This kind of parallelism is found in general-purpose GPU programming and vectorial processors.

Parallel computing systems can also be classified according to their parallel characteristics. Flynn's Taxonomy is the generally adopted classification, which states four categories based on the kind of parallelism they present (FLYNN, 1966 apud HENNESSY; PATTERSON, 2006):

SISD: Single Instructions, Single Data, this is the uniprocessor, a single processing element executes instructions that operate in a single data, one at the time.

SIMD: Single Instructions, Multiple Data, this category of processors exploit data-level parallelism by applying the same operations to multiple items of data in parallel, but using a unique instruction memory in a control processors, that dispatches the instructions to the processing elements. Each processing element has its own data stream. GPUs and vectorial processors are included in this category. This category is suited to tasks that are linked to fine-grain parallelism.

MISD: Multiple Instructions, Single Data, here, different processing elements with different instructions operate in a unique piece of data. This category is regarded as hypotetical, as no practical design of it exists yet. Some authors consider the systolic array proposed by Kung & Leiserson (1978) and pipelined processors to belong here, but there is no consensus. Pipelined processors expore instruction-level parallelism, taking advantage of overlapping instructions to evaluate them in parallel. Others include in this category fault-tolerant computers, which have redundant processing elements operating in parallel, applying the same processing leading to equal results, to detect and correct errors.

MIMD: Multiple Instructions, Multiple Data, the most general class, where a series of processing elements, each one possessing its own flow of instructions and data, operate in parallel in possibly uncorrelated processes. Multi-core processors present in modern computer systems belong to this category, as they explore thread-level

parallelism that is optimal for general purpose computing devices.

FPGA possess a large number of logical cells, therefore, their parallelization capabilities are, by its nature, more suitable to encoding algorithms that display fine-grain parallelism affinity. Although it is possible to implement coarse-grained and task parallelism into FPGA devices, fine-grain makes the most profit from the combinational nature of FPGA circuits. Therefore, choosing to encode an ML algorithm which displays fine-grain parallelism capabilities indicated the path followed in the work presented in this thesis.

Among the many ML techniques, automatic classification plays a prominent role, serving as a starting point for developing more complex solutions. A recently developed graph-based classification method called Optimum-path Forest (OPF) was chosen to be adapted as a hardware function, in order to demonstrate the architecture operation. The method has been tested through diverse applications and presented attractive characteristics, confirming the expected fine-grain parallelism potential (CHIACHIA et al., 2009; PISANI et al., 2011; SPADOTTO et al., 2008; GUIDO et al., 2010). The OPF is presented in details in Section 2.3. A proposed new application of this method is also proposed and discussed in Chapter 3.

Additionally, a new supervised learning algorithm for an OPF classifier was proposed and implemented. It was able to accelerate the classification with negligible performance penalty. Chapter 4 presents and discusses this proposition.

The final goal is to achieve processing acceleration in comparison with the software-only version of the method executed by an embedded processor. The resulting time reduction can make possible the future development of new applications focusing embedded systems, as well as contribute to improving current implementations. Although the architecture is conceived around a specific algorithm, it was designed to be flexible enough to accommodate different ones that are best suited for a given application. Nonetheless, the acceleration and parallelization capacities of FPGA-based architectures can drive innovative embedded solutions.

1.5 Summary of contributions

To achieve its main goal, this thesis faces the challenges of designing, implementing, and testing an FPGA-based embedded architecture for classification, validating the proposed architecture with an established classification algorithm.

There are three major contributions that derived from this goal pursuit:

- The first one is the implementation evaluation of a new application for the OPF, a Pedestrian Detection system. The system performance is evaluated and compared with known methods used in current implementations, which provides insights to contribute to a better understanding of the algorithm itself, as well as presenting a new option for building object detection systems.
- The second contribution is the development of a new supervised learning algorithm for training an OPF classifier. The new algorithm changes the way used to build the solution, aiming to reduce the amount of data necessary to reach a feasible classifier. It uses concepts of Self-organizing Map (SOM) algorithm to iteratively build the classifier from fewer seed nodes, differently from the classical OPF, that uses all nodes in a previously defined training set. The new proposition is able to accelerate the classification time , having in average a drop of less than 3% in accuracy yet in some cases, presenting a better one. This is a significant characteristic considering that online training is necessary for real classification applications, as discussed later in this work.
- Finally, the main contribution is the development of an FPGA-based architecture for classification, with hardware acceleration and low power consumption, enabling the use of ML techniques by resource-constrained embedded systems. The proposed architecture makes use of intrinsic algorithmic parallelism translation to a parallel hardware using FPGAs potential for this kind of implementations. Although validated with the OPF algorithm, the system is flexible enough to allow acceleration of different algorithms that can benefit from the same kind of fine-grain parallelism, contributing to performance enhancement of many applications.

Alongside the main contributions, a new workflow for FPGA programming was used

in this work, which made possible to evaluate how much it contributes to diminishing the technical challenges that are often associated with FPGA development.

The propositions and results detailed in this thesis were also published in the following papers:

DINIZ, W. F. S. et al. Evaluation of optimum path forest classifier for pedestrian detection. In: **2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)**. Zhuhai: IEEE, 2015. p. 899–904. ISBN 978-1-4673-9675-2. Disponível em: <doi://10.1109/ROBIO.2015.7418885>.

_____. FPGA accelerated Optimum-Path Forest classifier framework for embedded systems. **Microprocessors and Microsystems**, 2017. Under review.

1.6 Document outline

This chapter presented a general introduction to the problem of implementing ML methods on embedded systems, described some possible applications that could benefit from such a system, as well provided some background on current developments and perspectives. Then finished with an overview of the proposed architecture and a summary of the key contributions produced during this Ph.D. development. The following chapters proceed as detailed now:

Chapter 2 presents the fundamental concepts necessary for fully understanding the architecture that this work proposes. It presents the basics of embedded systems design using FPGAs.

Chapter 3 discusses the implementation of a Pedestrian Detection system using the classical form of a supervised learning OPF algorithm. It also presents the motivations behind such implementation and the methodology to analyze the results.

Chapter 4 presents a new supervised learning algorithm for training an OPF classifier, aiming to reduce the classification processing time, with a complete description of the method and an analysis of its performance.

Chapter 5 shows the main contribution of this work, the design, implementation and testing

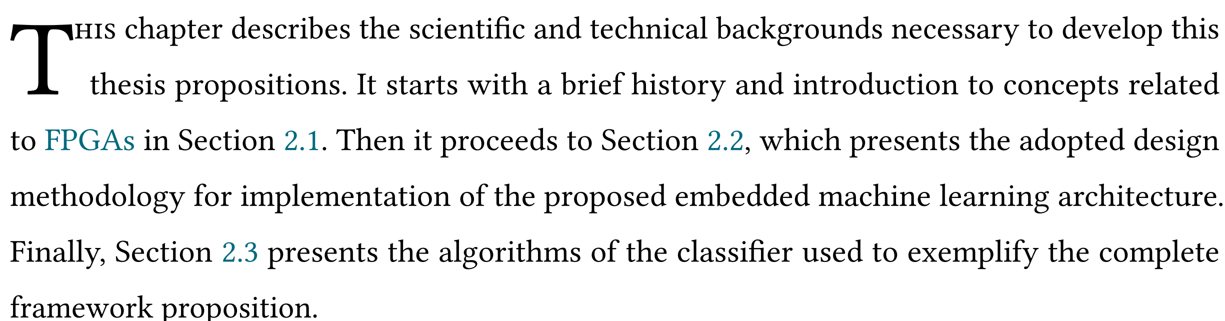
of an FPGA-based architecture for classification, using OPF as an example of application. The acceleration provided by the new architecture is evaluated by comparison with software-only OPF classical form.

Chapter 6 finally concludes this work with a subsume of the key contributions and prospects for future development.

Chapter 2

Fundamental concepts

> "Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning."
>
> — Albert Einstein



This chapter describes the scientific and technical backgrounds necessary to develop this thesis propositions. It starts with a brief history and introduction to concepts related to FPGAs in Section 2.1. Then it proceeds to Section 2.2, which presents the adopted design methodology for implementation of the proposed embedded machine learning architecture. Finally, Section 2.3 presents the algorithms of the classifier used to exemplify the complete framework proposition.

2.1 Embedded design with FPGAs

Modern integrated circuits include many interesting devices; among them, FPGAs are a class of Programmable Logic Devices (PLDs), meaning that they can be configured to assume any task that can be accomplished by a digital circuit. They were created to supply the necessity of highly-efficient digital circuits design while cutting off development and production costs.

The origins of PLDs can be traced back to the early 1960s, tightly tied to the development of the integrated circuit itself. First attempts on creating a configurable integrated circuit took the form of Cellular Arrays, which contained a series of simple logic cells with fixed, point-to-point communication, arranged as a two-dimensional array. These devices could be programmed by a metalization process at manufacturing and offered two-inputs logic functions. They could be used to develop custom combinational circuits, but, due to the lack of flip-flops, they could not be used to develop sequential circuits (KUON et al., 2007).

By the mid 60s, including programmable fuses in the cellular arrays led to the ability of programming the devices after its manufacturing. That ability was named field-programmability. The connections between cells were fixed, but their functionality could be programmed by applying specific currents or photo-conductive exposure (KUON et al., 2007 apud MINNICK, 1997). In the following decade, Read-only Memory (ROM)-based programmable devices were introduced, but the crescent costs due to increasing area exigencies for large logic severely restrained their growth.

The modern FPGA saw its first light when Xilinx® introduced in 1984 its first commercially available FPGA device, although the name was not used until 1988. That first FPGA introduced the now classic Configurable Logic Array, with 64 logic blocks and 58 inputs/outputs (KUON et al., 2007 apud CARTER et al., 1986). Today they evolved to include millions of logic blocks and with the advent of specialized blocks, like phase-locked loops, floating-point multipliers, and Digital Signal Processor (DSP) blocks, they offer higher levels of flexibility and applicability. Among the suppliers available today, the market is dominated by Xilinx® and Altera® (the latter recently acquired by Intel®), which hold together nearly 80% of the market-share. Other FPGA manufacturers have recently emerged since the 1990s, like former Actel® , now absorbed into Microsemi® , Lattice Semiconductors® , and QuickLogic® Corporation.

The modern FPGA internal architecture uses a bidimensional arrange of logic blocks with several interconnect bus lines linking them, as exemplified in Figure 2.1. Configurable routers switch the data paths to form the requested logic function. The most used solution for holding the chip configuration is a Look-up Table (LUT) composed by SRAM cells. This is a volatile configuration, meaning that once the device power is shut down, the configuration is lost. An external flash memory must be used to hold the configuration and a programming circuit added to the system for configuring back the device when it is turned on again. Another family of devices, called Flash-based FPGAs, have the ability to hold the configuration when powered down. Finally, there exists OTP FPGA, that uses configurable fuses and only admit being programmed once. They lose their flexibility, as they can not be reprogrammed, but offer smaller form factor and other characteristics that can be desirable for specific applications.

The logic block architecture is a trade secret for each individual supplier, however, they follow a basic scheme like the one shown in Figure 2.2. It consists of a 4-input LUT, a register, and multiplexer. The combination of the LUT inputs results in the desired logic function, while

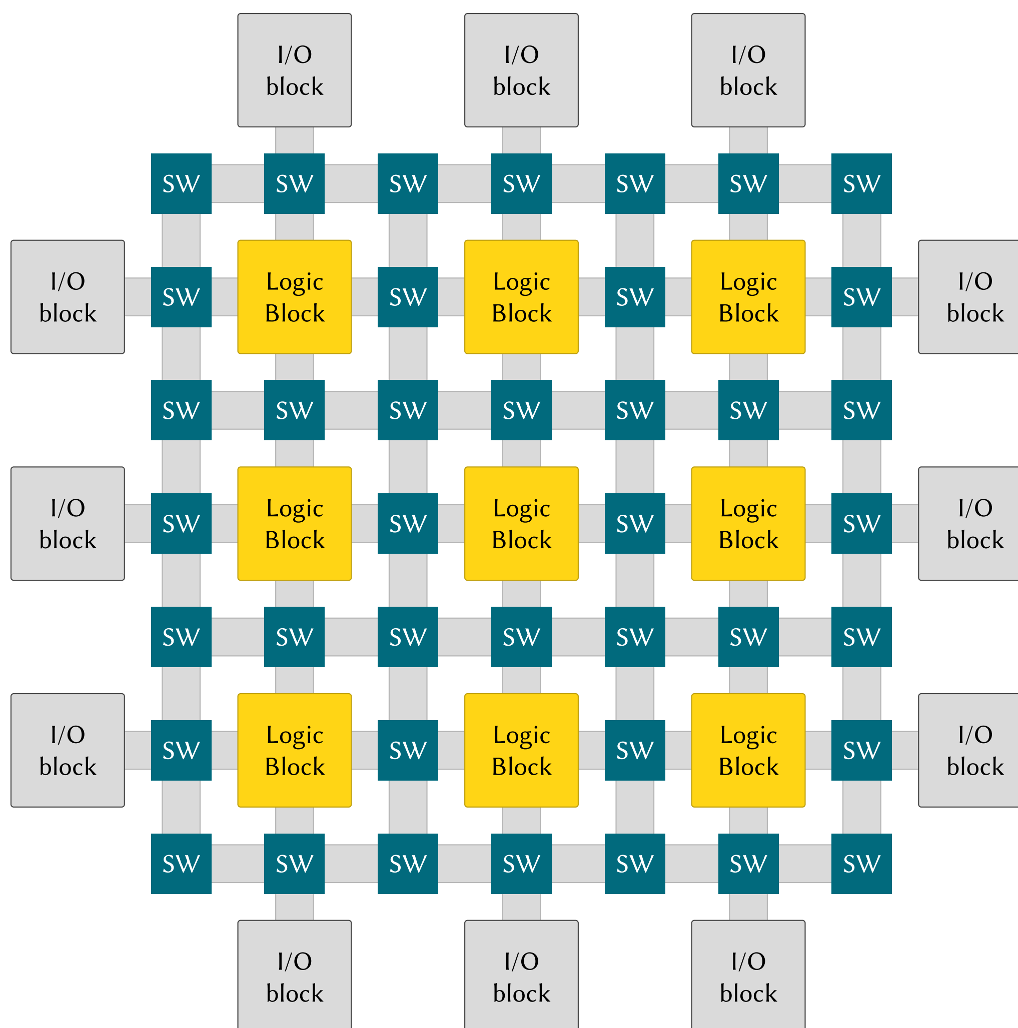


Figure 2.1 – The internal architecture of a modern SRAM-based FPGA is a bidimensional arrange of logic blocs interconnected by datapaths. Switches configured by a bit in the configuration memory connect I/O blocks and logic blocks, giving form to the desired circuit.

the multiplexer determines if the output will be synchronous or asynchronous. Clock signals have their own interconnection, independent of the data ones.

One interesting property of FPGAs is that they can be used to solve any computable problem. This can be easily demonstrated by the possibility of using them to implement Soft Processor Systems (SPSs). An SPS is a microprocessor architecture fully implemented using FPGA chip. The major suppliers offer their own SPS implementations in form of reusable Intellectual Properties (IPs), like Altera® NIOS® II and Xilinx® MicroBlaze™ . SPSs introduction greatly improved the variety of applications that can benefit from FPGA technology.

The latest introduction in the FPGA family are the so-called SoC/FPGAs. These devices are constituted by a Hard Processor System (HPS) and FPGA fabric in a single packaging, with

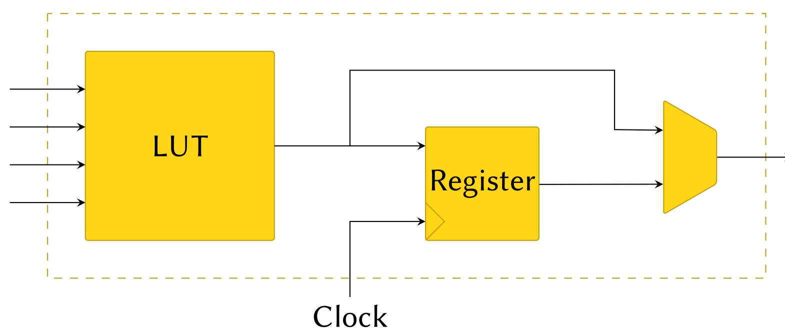


Figure 2.2 – A basic logic cell is formed by a multiple input LUT and a register. A multiplexer selects if the output will be synchronous or asynchronous. A logic block may contain several logic cells grouped together. Modern FPGAs incorporate specialized elements to some of these blocks, like adders and multipliers, enhancing the device's capabilities.

a communication interface between them. An HPS, differently from an SPS, is an actually separated hard microprocessor. It can also possess a variety of peripherals and, as it is separated from the FPGA fabric, it can run independently. However, their full capacity is brought when the FPGA specialization characteristics are used to build custom solutions for complex computational problems. The communication interface between the devices grants them the ability to exchange data, allowing shared processing. This kind of systems opened a new perspective to design and implement embedded systems.

FPGA development has traditionally used the same Hardware Description Languages (HDLs) used for developing other integrated circuits designs, like Application Specific Integrated Circuits (ASICs). VHSIC Hardware Description Language (VHDL) and Verilog are the most common languages, with all suppliers providing their own Integrated Development Environments (IDEs) specific tools for all development phases. Using those languages, the developer has to describe the behavior of the system, with the asynchronous parts as combinations of logic functions and synchronous parts as register-based memory operations. Because of these characteristics, it is called Register Transfer Level (RTL) design. It leads to a specific workflow that must be followed by the designer, who must possess an also specific set of skills to fully dominate all the development steps. Although development and integration times are quite smaller than the necessary for fully custom non-programmable devices, they are still a lot larger than software driven projects. This fact has always been a wall that drove away non-electronic/computer developers from using them, which otherwise could benefit from FPGA technology.

To deal with this issue, a range of high-level design tools were introduced to offer more

abstraction to FPGA development. The latest trend in this direction was the launching of Open Computing Language (OpenCL) compatible tools for FPGA development.

2.2 OpenCL as an option for FPGA development

With the barriers imposed by crescent power consumption as the clock speed increased, Central Processing Units (CPUs) manufacturers started to adopt parallelism as a way to increase computational power without exceeding a limited power envelope. Almost at the same time, some people started to use GPUs to perform general computing besides their natural graphics processing, mapping general computing like matrix multiplication into graphics Application Programming Interfaces (APIs) like Open Graphics Language (OpenGL) and DirectX® . A growing, but highly fragmented, heterogeneous ecosystem for parallel programming started to emerge, as each supplier was managing their own technology and releasing their own proprietary Software Development Kits (SDKs). That diversity made integrating different technologies rather complicated, so it was necessary to develop a common standard capable of unifying all those technologies under the same guidelines. The response for this request is OpenCL. It was developed in its first stages by Apple™ Computer Inc. By 2008, its first draft was ready. On August 28, 2009, several hardware providers like Intel® , AMD® , NVIDIA® and others, unified under the Khronos™ Group, released the first revision as an open standard.

OpenCL™ is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs, and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms (Khronos Group, 2009). It provides a framework for parallel computing, including a C/C++ language extension for writing programs for parallel devices, an API for controlling the program execution in those devices, libraries and runtime components. Its goal is to define a common point for parallel programming regardless of the employed hardware. To accomplish this, it defines standards and protocols that hardware providers must adopt to build a compliant device. Following those standards enables different types of hardware to execute the same code. The specific optimizations or specialized instructions for complying the code to each platform are hidden from the developer, as they are, in this point, obliged to comply with the computation model proposed by OpenCL. The developper then can concentrate only on its algorithm design, with the task of generating

the specific machine code to the platform he is developing for, taken care by the compiler.

Presenting highly parallel characteristics, FPGAs inclusion to OpenCL ecosystem seems natural. The historically demanding skills for FPGA development always encouraged endeavors in creating high-level tools; C-like languages as descriptors for automatic code generation tools were not uncommon. The emerging of OpenCL represents a unifying point for these initiatives, it provides as well a large community of users ready to explore the benefits of FPGA technology. Nonetheless, the all-new way of thinking FPGA development deserves a study to understand its characteristics and to make better use of them.

OpenCL Concepts

The next sections introduce some key concepts for understanding how to develop FPGA applications using the OpenCL framework. It is largely based on OpenCL Reference Guide, Revision 1 (Khronos Group, 2009).

OpenCL introduces its own architecture for organizing parallel programming, based on four models:

1. Platform model;
2. Execution model;
3. Memory model;
4. Programming model.

Each one of these models governs an aspect of the development philosophy resulting in the unification of parallel programming methodology and an abstraction of hardware capabilities.

The *Platform Model* defines the entities that take part in parallel code execution. The main entity is the *host*, that will be connected to one or more OpenCL *devices*. Parallel code is executed in devices, while hosts take care of non-parallel tasks, data distribution among entities, and controlling the devices execution. The computing runs on the host according to its own model. When necessary, it dispatches commands to the devices enabling parallel computation to occur. OpenCL devices are divided in *Compute Units (CUs)* that are further

divided into *Processing Elements (PEs)*. Figure 2.3 shows the platform model hierarchy.

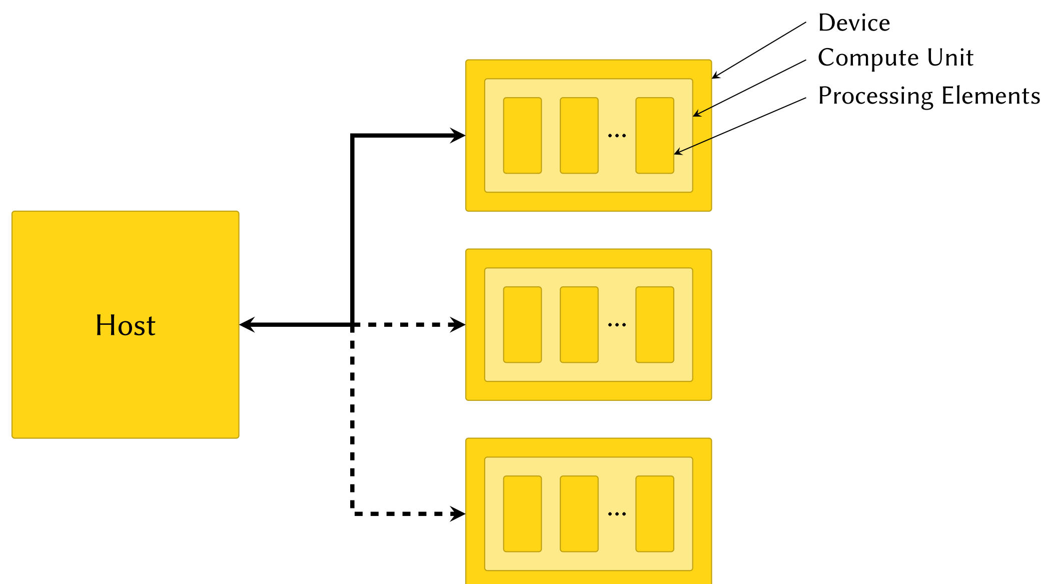


Figure 2.3 – The OpenCL architecture defines a Platform Model consisting of a host and one or several devices. The hosts manage non-parallel tasks and dispatches commands to control the parallel execution on devices. Devices themselves are divided in Compute Units, and these are divided in Processing Elements.

The *Execution Model* defines the directives to organize how the code runs. The *host code* is the part of an OpenCL program that runs on the non-parallel host processor. The parallel code is executed on the devices under the form of *kernel* functions. The OpenCL compiler converts the kernels into the platform specific machine code according to the programmable device.

The core of OpenCL execution model is the organization under which kernels are executed. When a kernel is enlisted to be executed, the API creates an index space to organize the data access. The kernel instructions are then executed in each point of this space. An individual instance of a kernel is called a *work-item*. They are uniquely identified by a global identifier associated with a point in the index space. A coarser division of the index space is provided by *work-groups*. They also receive a unique identifier with the same dimensionality of the work-items index space. Each work-item also receives a unique work-group level identifier, meaning that an individual work-item can be referenced by its global identifier or a combination of the identifier of the work-group it belongs to with its local identifier within the work-group. The set of work-items in the same work-group executes in parallel, on the processing elements of a computing unit, thus making the bridge between platform and execution models.

A collection of API commands prepares the data indexing to execution. OpenCL supports

two models of parallel execution: data parallelism and task parallelism. The set-up of the context and the data indexing is the developer responsibility and must be done in the host code using the appropriate API functions.

The next model is the *Memory Model*. It determines how hosts and devices must manage their access to the data. There are four defined memory spaces that have their own specific characteristics and are defined during compiling time using the respective qualifiers in the source code. They are:

Global Memory: this memory space can be accessed by both host and devices. All work-items and all work-groups have read and write access to any memory object allocated in this space. Depending on the hardware capabilities, the access can be cached to improve timing.

Constant Memory: the items allocated here remains constant during a kernel execution, hence the name. The host allocates and initializes the memory objects before calling the kernel, that will have read-only access. The memory content can be changed again by the host before executing another kernel call.

Local Memory: a memory that has local access to a work-group. It is used to store data that is shared among the work-items belonging to the same work-group. Depending on hardware design it can be either a specific memory bank on the OpenCL device or a mapped region of the global memory.

Private Memory: the deepest memory level that is individual for each work-item. The data stored here can not be seen by any other work-item than its owner.

The implementation of these memory spaces to the corresponding hardware specification is platform-dependent, meaning that the hardware manufacturer itself decides which type and amount of memory circuits, as long as the implementation respects the model. Logically, this decision impacts either on the device performance and on development, production, and final cost. Faster memory is more expensive, thus it is usually provided in smaller amounts and used for Local and Private memories, the ones closer to the work-items, thus providing better performance. Nevertheless, the framework provides functions to query the device capabilities so the developer has the information to adapt the program configuration in the most suited way to the platform of choice. Figure 2.4 shows a diagram of the memory model.

Finally, the *Programming Model* defines the guidelines for writing the programs. Although OpenCL focuses in data parallelism, it supports both data and task parallel programming models.

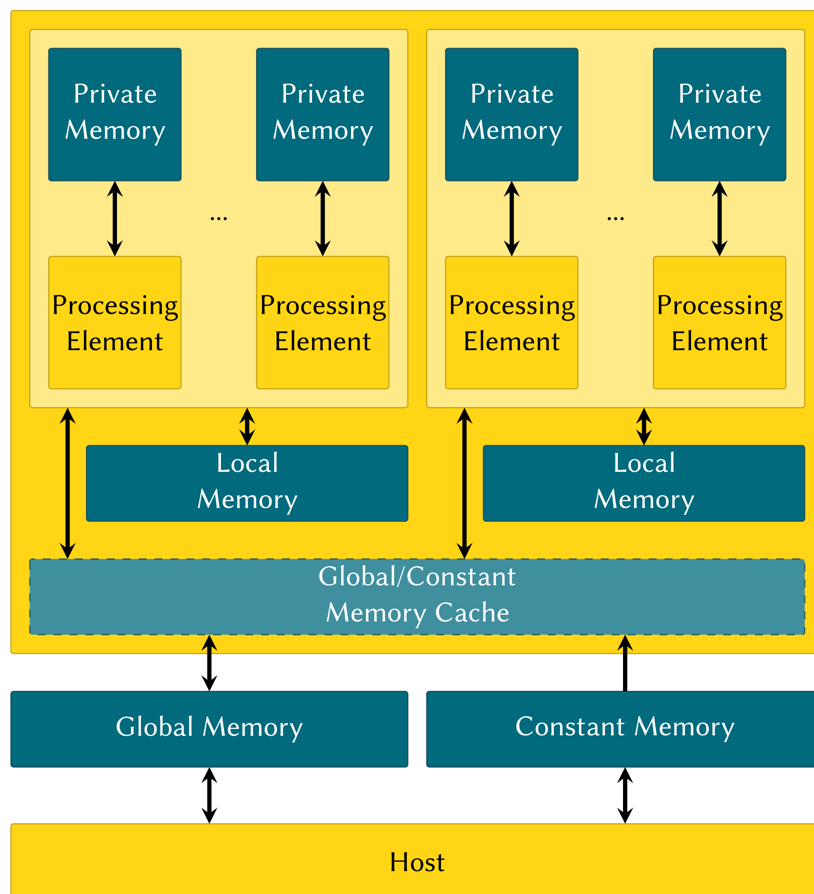


Figure 2.4 – OpenCL defines a Memory Model based on four memory spaces. The Global Memory can be accessed by both Host and Devices, with read and write capabilities. It is generally the largest available memory, but with the slowest access timing. It also may be cached, to improve timing. Each device has a Local Memory that can be accessed by all CUs and PEs, also for read and write. Each PE has also its own Private Memory, which only the owner can access. The last space is the Constant Memory, which the Host accesses for read/write and CUs and PEs as read-only. The implementation of the model to actual hardware is entirely up to manufacturer's decision.

Data parallelism is done under Single Instructions, Multiple Data (SIMD) paradigm, that is, a same instruction or sequence of instructions is executed in parallel over a collection of similar data, following the execution and memory models. The index space determines how the elements of the memory objects will be allocated to the processing elements. A strict data parallel model ensures a one-to-one mapping between memory elements and processing elements. OpenCL implements a relaxed model for data parallelism, so this one-to-one map is not required. It allows for the flexibility of using heterogeneous hardware platforms.

Task parallelism is achieved by executing a single kernel independently of any indexing. It is logically equivalent to executing a kernel on a computing unit with a work-group containing a single work-item. Parallelism is achieved by dispatching several tasks or using device specific

data vectors.

OpenCL workflow for FPGA programming

OpenCL is available as an FPGA design option since 2013, when Altera® released an SDK for its products, what was followed by Xilinx® . The language adoption represents a considerable advance to FPGA development (HILL et al., 2015). In reality, it introduces an entirely new workflow with a higher level of abstraction for FPGA-based systems development.

Traditionally, the RTL workflow is used for FPGA development when an HDL is the programmer option. Although these languages can always be used to design parallel systems, this process requires a deep knowledge of digital circuit singular design and involves time-consuming simulation and verification stages. However, the OpenCL workflow, with its higher abstraction level, facilitates the complete development in general, and it has to be forcefully adopted when OpenCL language is used. The main advantage of the OpenCL workflow is to incorporate the RTL procedures in a transparent way, which brings efficiency to the development. Figure 2.5 shows the relation between both workflows.

It can be seen in Figure 2.5 that RTL workflow comprises some fundamental steps, taken after the code development, which are: functional simulation of the compiled code, synthesis of the approved simulated code, routing of the viable synthesis result considering all power and timing constraints, to finally upload to the target board the bitstream hardware configuration file. All these steps are manually controlled by the developer, and the process has to return to the initial step if any intermediary check fails, requiring a number of iterative loops to ensure correctness at each stage. It is common to reach a high number of iterations until a design gets approved.

The OpenCL workflow, also shown in Figure 2.5, rests over the RTL one. It means that the RTL steps are still necessary, but now they proceed in an automatic and transparent way. This happens because the OpenCL compiler uses verified IPs to handle the data interfacing. These IPs have their constraints already mapped, then there is no need to execute a full analysis one more time.

Following the compilation of the OpenCL code, the functional verification is done through emulation of the target system in the host developing computer. After functionally approved, the system is ready to be optimized and checked against the resource constraints, according to the specific FPGA device in use. The optimizations in this point are done by the compiler, meant for giving the developer a view of possible bottlenecks in the translated architecture. By following specific guidelines, these bottlenecks can be mitigated. The procedure follows up with the automatic synthesis and routing steps. This avoids the always present feedback to the initial step imposed from using the manual procedure.

The compiler also takes care of handling the interfacing between the Host and the Device through predefined interconnect IPs that map the data exchange to the target hardware. This is traditionally the most time-consuming task in the system design using RTL workflow, as the developer must tune his solution to work in very specific protocols to be able to manage the data transfer between the system components and extract the best performance, linking the design with the actual hardware in use.

In the previous sections, the technology and development methodology for the FPGA architecture implementation of the chosen classification algorithm were already introduced. Next section follows with an introductory overview of the OPF characteristics and main algorithms.

2.3 The Optimum-path Forest classifier

Classification problems are defined as the task of finding a way to partition a space, formed by feature vectors describing a set of samples, and then attributing a class to unknown samples corresponding to each partition found. In simple terms, they map labels to samples according to some decision boundaries. Automatic Classification has been a recurrent challenge in many science fields, driving considerable research that gave light to several methods. Many modern AI applications, for example, rely on classification as the basis for decision making.

The initial step for all classification methods is to learn about the problem to be classified, a procedure usually called as training. According to how the methods gather the knowledge about the classes and their distributions in the feature space, the learning methods can be classified in

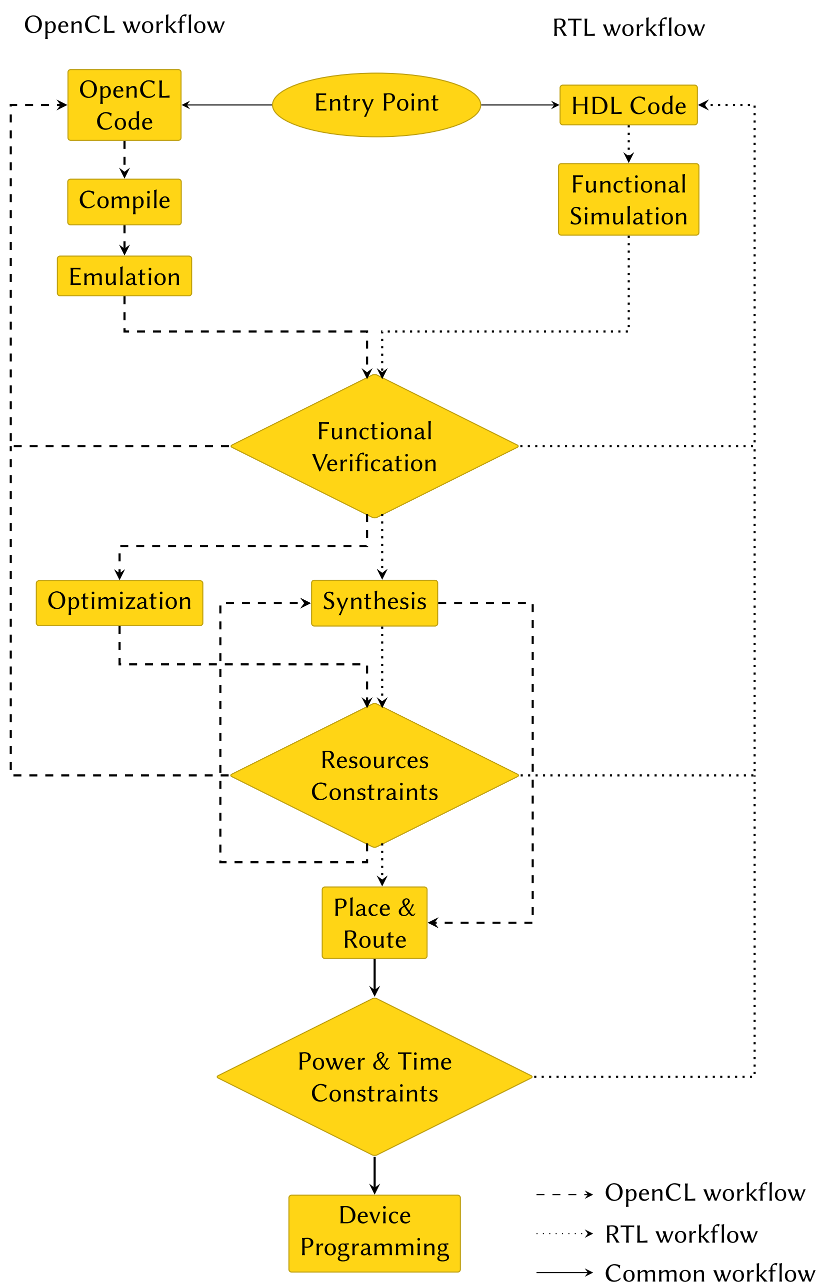


Figure 2.5 – The FPGA OpenCL programming workflow. It incorporates theRTLworkflow as an automatic process, based on verified components. This introduces a higher level of abstraction to the development, as the user will concentrate on the algorithm design.

three variations: *Supervised*, *Unsupervised* or *Semi-supervised* methods. In supervised methods, the learning is accomplished by analyzing a set of training samples whose classes are fully labeled. The unsupervised methods lack any information about the classes at training time. They are also known as clusterization methods, as they try to group samples by similarity and then infer classes from those groups. The semi-supervised methods use partially labeled data to classify the unknown samples. New classes can be created if the learning procedure includes that capacity.

There are different approaches to find the decision boundaries in the feature space that map the classification. Some classifiers use geometrical approaches, that is, they try to find descriptions of the decision boundary and classify the samples in relation to them. Support Vector Machiness (SVMs) apply this methodology. However, the feature space is not always linearly separable. For those cases, SVM solution is mapping the feature space to a higher dimension, making it linearly separable in a hyperspace. The goal is to find an optimal mapping to that high dimensional space. When a suitable mapping is found, it returns a description of the hyperplanes corresponding to the decision boundaries; this description is based on a collection of support vectors that corresponds to frontier elements in feature space. The hyperplane divides the feature space into two halves, each one belonging to a class. By verifying in which half a sample lies in feature space, the classification is achieved. However, it is not always possible to find a finite dimension that separates the classes or the computational cost grows fast with the increasing number of feature vectors (COLLOBERT; BENGIO, 2004; TANG; MAZZONI, 2006; PANDA et al., 2006). To solve this issue, SVM uses a kernel function to implicitly work in the higher dimension, without having to actually map the space, what is called the *kernel trick*. Other methods, like Bayesian Classification, use statistical analysis to find probability distributions to describe the decision boundaries.

Graph-based classification is a set of methods that represent feature spaces using Graph Theory paradigm. The field was introduced in the late 70s and represent an interesting and powerful framework for building classifiers. However, its use decreased as the computational complexity of graph algorithms increased. Starting in the late 90s, recent years have seen a renewed interest in the field, as the computational power of the current computer generation is more compatible with the cost of those algorithms (CONTE et al., 2004). A growing number of publications demonstrated new applications, especially on image and video processing. Table 2.1, reproduced from Conte et al. (2004), shows the research scenario so far for Graph Matching, one of the available graph-based methods.

Table 2.1 – Graph matching based applications published up to 2002, according to Conte et al. (2004).

Period	2D & 3D Image Analysis	Document Processing	Biometric Identification	Image Databases	Video Analysis	Biomedical/ Biological
up to 1990	3 (0)	1 (1)				
1991–1994	3 (0)	2 (2)	1 (1)			1 (1)
1995–1998	8 (3)	9 (5)	6 (6)	5 (5)		1 (0)
1999–2002	19 (6)	8 (4)	8 (8)	8 (7)	6 (6)	2 (2)

The last decade saw new developments to the area, with the adoption of new approaches and applications. An updated survey is presented by Foggia et al. (2014), surveying more than 180 papers in the topic. It shows recent advances and prospects to further development, including a comprehensive introduction to the most used techniques.

In the late 2000s, a new graph-based classifier, called OPF, was proposed by Papa et al. (2009). The method represents each class by one or more optimum-path trees rooted at key samples called prototypes. A heuristic for defining prototypes directs the graph partitioning whose goal is to expose a relationship between the nodes, that will form homogeneous groups. The classification is done by a competition where each tree will try to conquer the presented sample offering path-cost to connect the sample with a prototype, considering all possible paths to each prototype originating on every node. As the costs of each node are a measure of similarity, it can be concluded that the classification process is connectivity by minimization of dissimilarity.

The actual form of OPF as a general classifier derived from an application for image processing known as Image Foresting Transform (IFT) (TORRES et al., 2002). The IFT central concept is to consider a digital image as a graph, whose nodes are pixels and the edges are determined by a previously defined adjacency relation. The most common adjacency relation are 4-connectivity, where each pixel is linked to four neighbors, two vertical and two horizontal. The algorithm extracts a forest of minimum cost paths rooted on a previously defined set of seed pixels, obeying an application-specific path-cost function. In its essence, IFT consists of a generalization of Dijkstra's shortest path algorithm, notably, with the ability to use multiple starting nodes and general cost functions. Specifying different cost functions produces different applications (FALCÃO et al., 2004).

The concepts of IFT can be extended from images to general graphs, giving birth to a generalized framework from which the OPF classifier was developed (PAPA et al., 2009). What were pixels and intensity/color values in the image processing domain are converted to graph nodes and their respective feature vectors. Features can be extracted in many different application-specific ways.

OPF classifiers can be constructed using either supervised or unsupervised learning (PAPA, 2008). This thesis focuses on the supervised variation as it is the most common variation with the majority of applications. Next Section describes in details the training and classification

stages of a supervised learning based OPF classifier.

2.3.1 Supervised learning variation

Training algorithm

The basic training routine presented in Papa et al. (2009) proceeds in two distinct stages called fitting and learning. The fitting stage starts by selecting a subset of samples from the problem universe, referred from now as the training set. As this is a supervised learning procedure, all samples are already labeled with their correct classes. One requirement of the training set is that it must contain at least a sample for every class in the problem. From the training set, a complete graph using the samples as nodes is constructed with the edges weights determined by a previously defined dissimilarity function. The Euclidean Distance (L^2 norm) is a common dissimilarity function, used in many methods, however, any function that suits a particular requirement of an application can be used instead. From the complete graph, the algorithm proceeds with the extraction of a Minimum Spanning Tree (MST). The MST exposes a similarity relation between the nodes. Decision boundaries can be found in this structure in the points where two nodes that belong to different classes are connected. This is the heuristic for finding the prototypes. The edges between the prototypes are removed and they receive an associated path-cost of 0, representing that they are roots of homogeneous trees, that is, all nodes in the same tree belong to the same class represented by its corresponding prototype. Each non-prototype node will receive an associated cost that is equal to the path-cost to reach their respective prototype. The path-cost is given by the OPF connectivity function, defined as:

$$f_{max}(\pi_{n,p}) = \max\{w_n, \ldots, w_p\}, \tag{2.1}$$

where $\pi_{n,p}$ is a path from node n to prototype p, represented by a sequence of all edges weights in the respective path. So, each node associated cost will be in the end, the value of the largest edge weight in the path from itself to the respective prototype.

As mentioned before, the weights are calculated by the dissimilarity function $d(s,t)$.

Algorithm 2.1 is called OPF algorithm, and calculates the minimization of the connectivity function defined in Equation 2.1 on the domain of graph T, formed with the training set nodes.

Algorithm 2.1 OPF classifier training algorithm. The priority queue is a particular data structure that has an ordered storage policy; it ensures that the element in the head always has the minimum cost in the queue and also permits arbitrary removal. The λ function returns a label that identifies the class that the sample belongs to and function $d(x,y)$ is the dissimilarity function, that returns a metric of how separated the samples are in the feature space.

Require: Training set graph T, prototypes set $S \subset T$
Auxiliary: priority queue Q, real variable cst, function $d(x,y)$
Output: classifier P, cost map C, label map L

```
 1: function OPF_Training(T)
 2:     for all t ∈ T \ S do
 3:         C(t) ← +∞
 4:     for all s ∈ S do
 5:         C(s) ← 0, L(s) ← λ(s), P(s) ← nil, insert s in Q
 6:     while Q ≠ ∅ do
 7:         Remove s from Q such that C(s) is minimum
 8:         for all t ∈ T | t ≠ s and C(t) > C(s) do
 9:             cst ← max{C(s), d(s,t)}
10:             if cst < C(t) then
11:                 if C(t) ≠ +∞ then
12:                     Remove t from Q
13:                 P(t) ← s, L(t) ← L(s), C(t) ← cst
14:                 Insert t in Q
        return P
```

Figure 2.6 shows a graphical representation of the algorithm.

There is a relation between MST and the minimization of f_{max}, in the sense that all possible paths in a MST are minimal. The implication is that each edge will have a weight that is corresponding to the distance between adjacent nodes.

The classification algorithm

The OPF classification algorithm assigns to the unknown sample the class of the most connected prototype considering all possible paths originating from every classifier node. It can be viewed as a competitive process, where each tree will try to conquer the new node for itself offering a reward, that is the path-cost to its prototype. The one that offers the best reward, meaning the optimum path, will connect to the sample and propagate its labels. As

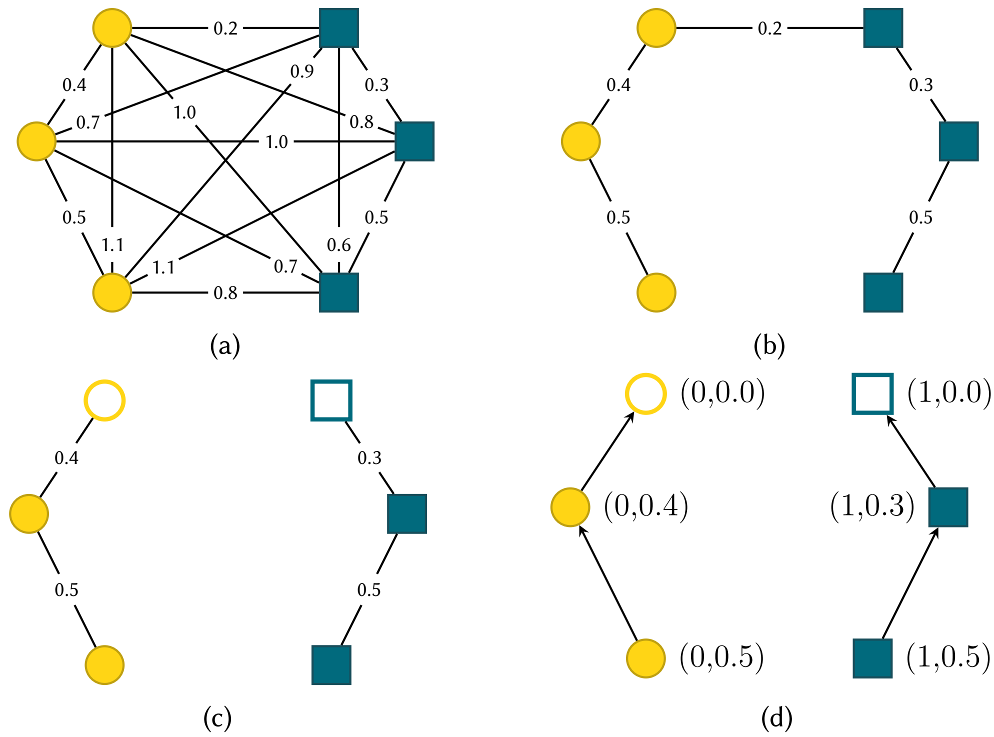


Figure 2.6 – Training sequence for the OPF classifier.
(a) The complete graph with edges weighted by dissimilarity.
(b) The Minimum Spanning Tree is found.
(c) The prototypes are marked and their connections undone.
(d) The last step is to assign the labels and the costs for each node. The prototypes are assigned a cost of 0 and propagate their labels to the nodes in their trees. The cost of the nodes are the maximum value of the edges in the path from them to their respective prototypes.

the path-costs are given by the dissimilarity function, the classification is, most essentially, a process of dissimilarity minimization.

Formally, for a given unknown sample s, its resulting classification label $\lambda(s)$ is given by the classification function defined as:

$$\lambda(s) = \lambda(t) \mid t = \min_{\forall t \in T}\{\max\{C(t), d(s,t)\}\}, \tag{2.2}$$

where T is the classifier set graph, $C(t)$ is the classifier cost map and $d(s,t)$ is the value returned by the dissimilarity function. The classification process is ilustrated in Figure 2.7.

The naive implementation of OPF algorithm is an extensive search where all the possible paths to all prototypes are searched. However, (PAPA et al., 2010) presented a variation of the algorithm that explores a theoretical property of the OPF that can make the classification

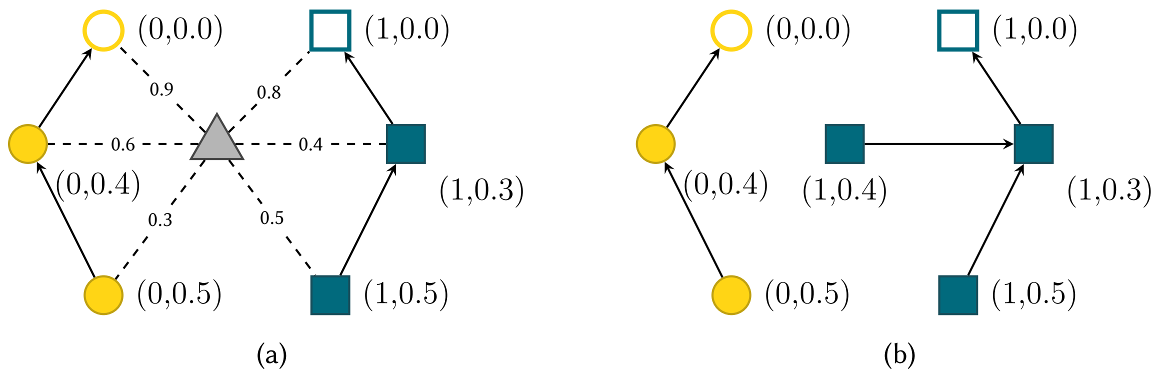


Figure 2.7 – Classification sequence for the Optimum-Path Forest classifier.
(a) The unknown sample is presented to the classifier nodes. The classifier nodes then try to connect to the unknown sample, offering it their costs. The resulting cost for the unknown sample is the greatest value between the cost offered by the classifier's node and the distance from the unknown to the sample in question.
(b) The node that offers the minimum cost path connects to the unknown sample and gives it its class label. Note that although the closest node to the unknown sample is of the class circle, the most connected prototype is one of the class squares. Thus, the sample is classified as square.

faster. As the classifier must find the node that offers the minimum cost to connect to the unknown sample, one can assume that the winning one will have a small cost (not necessarily the lowest). So, if the classifier presents its nodes in an ascending order of their associated costs, the probability of the winning one to be at the beginning of the list is high. Then, one must search through the list until it finishes (in the worst case) or until finding a node whose offered cost is greater than the assumed cost for the unknown sample to connect to the previous node. This previous node will then be the winner. Algorithm 2.2 shows the enhanced classification algorithm.

Learning from errors in the classification

The resulting forest from the fitting algorithm is already suitable to classify unknown samples. However, a learning method was proposed to increase the classifier accuracy (PAPA et al., 2009). The current version of the learning algorithm published in the library used in this work consists of using a third set of samples called evaluation set to assess the accuracy of classification. This set is kept apart during the fitting stage so that the classifier does not know any of its samples. This procedure keeps the training set from producing an over-fitted classifier. The classifier accuracy is evaluated by classifying the samples in the evaluation set.

Algorithm 2.2 Enhanced OPF training algorithm. A theoretical property of the OPF can make the classification faster. As the classification is a minimization problem, the search may be faster if the nodes are presented in order. The probability of the winner node to be at the beginning of the list high, thus, not necessarily all nodes need to be checked.

Require: Ordered classifier set T, label map $\lambda(T)$, connectivity cost $C(T)$ and test set S.
Output: Test set label map $\lambda(S)$.
Auxiliary: Variables tmp and $mincost$, counter i.

```
1:  function OPF_Classifying(T)
2:      for all s ∈ S do
3:          i ← 1
4:          mincost ← max{C(t_i),d(s,t_i)}
5:          λ(s) ← λ(t_i)
6:          while i < |T| and mincost > C(t_{i+1}) do
7:              tmp ← max{C(t_{i+1}),d(s,C(ti+i)}
8:              if tmp < mincost then
9:                  mincost ← tmp
10:                 λ(s) ← λ(t_{i+1})
11:             i ← i+1
        return λ(S)
```

Then, randomly-chosen misclassified nodes in evaluation set are exchanged with non-prototype samples of the training set and the fitting process is executed again in this new set, producing a new classifier instance. The process of fitting-evaluating-replacing nodes is repeated until the variation of the accuracy is lesser than a pre-specified value or a maximum number of iterations is reached. This procedure is meant to find the most informative samples in the universe set for being used in the final classifier. Algorithm 2.3 exposes the learning procedure.

The principle is that by including the misclassified nodes in the training, we can get the most informative nodes to form the classifier, thus increasing accuracy. It is also important to remark that the accuracy measurement must take into account the relative distribution of the samples in the feature space. Unbalanced distributions, in which a class has a much smaller number of samples, may increase the error rate, because the classifier may assign most frequent labels to such samples, possibly eliminating all its representatives. Then, the accuracy is calculated using a per class procedure and the selection of samples for swapping must take the classes distributions into account.

Let $NE(i)$, $i = 0,1,...,c$, with c being the number of classes, be the number of samples of each class in the evaluation set. We define two error metrics:

$$e_{i,1} = \frac{FP(i)}{|E| - NE(i)} \quad \text{and} \quad e_{i,2} = \frac{FN(i)}{NE(i)}, \tag{2.3}$$

Algorithm 2.3 Leaning procedure for the OPF classifier. $CalculateAccuracy$ is a function that returns the accuracy of an OPF classified set. $SwapErrors$ is a function that swaps misclassified nodes in the evaluation set for non-prototype nodes in the training set.

Require: test set T, evaluation set E.
Auxiliary: δ: variation from previous to current accuracy, Λ: accuracy variation limit, $iter$: current iteration, $maxIter$: maximum number of allowed iterations, if the procedure does not converge, $curAcc$: accuracy of the current iteration, $prevAcc$: accuracy of the previous iteration, function $CalculateAccuracy$, function $SwapErrors$.
Output: Classifier P.

```
1:  function OPF_Learning(T, E)
2:      iter ← 0
3:      prevAcc ← −∞
4:      while δ < Λ AND iter < maxIter do
5:          C ← OPF_Training(T)
6:          OPF_Classify(C, E)
7:          curAcc ← CalculateAccuracy(E)
8:          δ ← |(curAcc − prevAcc)|
9:          if curAcc > prevAcc then
10:             prevAcc ← curAcc
11:             P ← C
12:         SwapErrors(T, E)
13:         iter ← iter + 1
        return P
```

where $FP(i)$ and $FN(i)$ are, respectively, the number of false positives and false negatives of each individual class. False positives are samples of a different class that are classified as being of class i and false negatives are samples of class i that are classified as not. These two errors define the main error as:

$$\varepsilon(i) = e_{i,1} + e_{i,2} \tag{2.4}$$

The balanced accuracy A for an instance I of a candidate classifier is then given by:

$$A(I) = \frac{2c - \sum_{i=1}^{c} \varepsilon(i)}{2c} \tag{2.5}$$

With the conclusion of this chapter, all the algorithms needed to build a supervised OPF classifier are presented. Next chapter presents a practical application of such classifier.

Chapter 3

Implementation of an OPF-based pedestrian detection system

"Apply yourself. Get all the education you can, but then... do something. Don't just stand there, make it happen."

— Lee Iacocca

3.1 Introduction

Recently, a particular interest has grown in the industry for systems that help the task of driving a vehicle. The main objective is helping to mitigate the risk of accidents. Figure 3.1 shows the mortality rate of traffic accidents per 100,000 population in the world. Given this alarming number of accidents, many of them caused by driver's mistakes, a new class of systems appeared to help mitigate this issues. These systems are collectively called Advanced Driver Assistance Systems (ADAS).

ADAS can comprehend one or many sub-systems that may provide information to the driver, thus increasing his awareness of the situation around him or actuating in the vehicle sub-systems, like the brakes, to prevent a dangerous situation. One of such systems is the Pedestrian Detection systems. Pedestrians represent a significant amount of fatalities in traffic accidents. Table 3.1 shows the pedestrian mortality in USA in a 10 years timeframe. It can be noticed that although the total number of accidents have decreased in the considered period, the proportional number of pedestrian fatalities increased.

The pedestrian detection task is challenging, especially because of the large in-class variation and complexity of poses that a human form can assume. Despite the difficulties, the field has been attracting a considerable amount of research, due to the large benefits that can be derived from it.

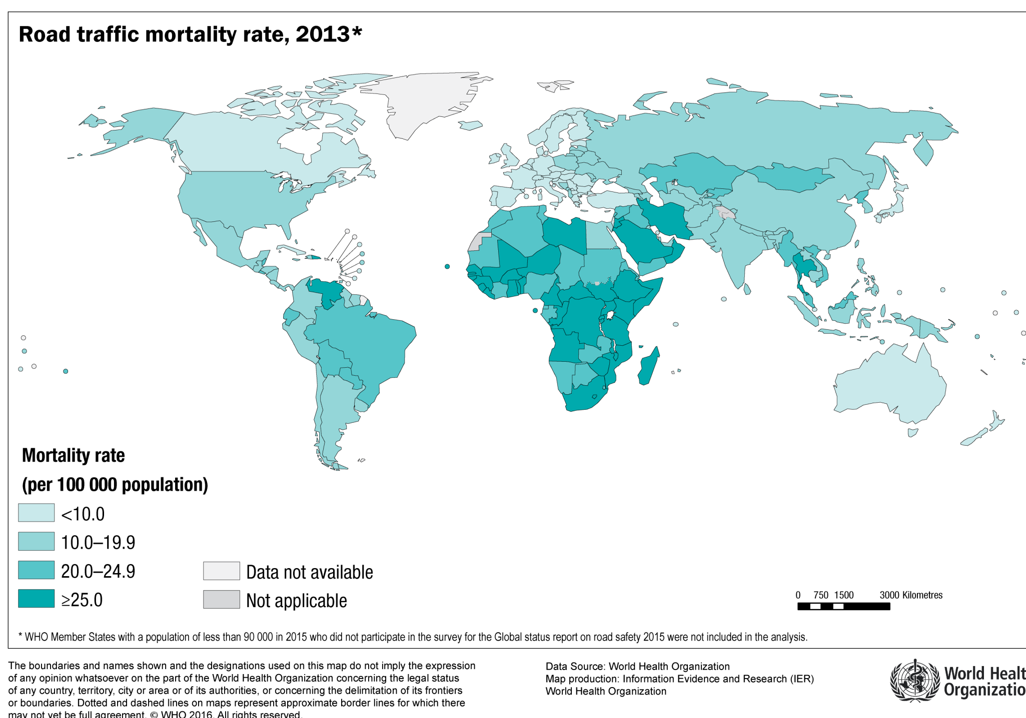


Figure 3.1 – Road traffic mortality rate by country. Source: World Health Organization, website. URL: <http://gamapserver.who.int/mapLibrary/Files/Maps/Global_RoadTraffic_Mortality_2013.png>, accessed 07/13/2016

Table 3.1 – Pedestrian mortality rates in USA, from 2004 to 20013.
Source: Fatality Analysis Reporting System (FARS) 2004-2012 Final File, 2013 Annual Report File (ARF).

Year	Total Fatalities	Pedestrian Fatalities	Percentage
2004	42836	4675	11%
2005	43510	4892	11%
2006	42708	4795	11%
2007	41259	4699	11%
2008	37423	4414	12%
2009	33883	4109	12%
2010	32999	4302	13%
2011	32479	4457	14%
2012	33782	4818	14%
2013	32719	4735	14%

Pedestrian detection systems work processing the data captured by a sensor (usually an imaging sensor such a video camera or radar or yet a combination of both) to identify people crossing the vehicle path, especially the ones not noticed by the driver. Once detected, the information could be relayed to the driver by some kind of interface, or a higher level monitoring system can, for example, decide to apply the breaks and stop the vehicle, if it judges the situation harmful. The sensors may be night vision capable, increasing the safety even

more.

This chapter introduces a proof of concept for the pedestrian detection problem, with the objective of evaluating characteristics of the proposed classifier and also its suitability for implementation in the proposed framework for embedded systems.

3.2 Current approaches on Pedestrian Detection

Specifically, vision-based pedestrian detection is a variation of human detection that focuses on finding people in traffic environments. The people in this environment are usually visible and approximately in an upright position. A number of propositions have been done in the last decade.

Recently, Dollar et al. (2012) published a sensible survey of state-of-the-art methods for pedestrian detection, also publishing a unified methodology for their evaluation that is the current benchmark of the field. In their work, it is possible to perceive that the majority of the systems use a sliding windows approach for object detection followed by binary classification, with candidate regions on the image encoded by dense features set. The Histogram of Oriented Gradients (HOG) feature set introduced by Dalal & Triggs (2005) is the most used, alone or alongside with other techniques that can improve its information. As for the classifier, linear SVMs have been common, with some applications using also Artificial Neural Network (ANN) such as Multi-layer Perceptrons (MLPs), Random Forests and most recently Convolutional Neural Networks (CNNs).

Considering this, the pedestrian detection system to be described in the following of this thesis is based on sliding windows for coarse detection, HOG feature descriptors and new proposition for the binary classification part, using a supervised OPF classifier. The performance of OPF is compared in terms of classification accuracy and processing speed with the same datasets applied other classifiers currently used for pedestrian detection.

3.3 System overview

A vision-based pedestrian detection system executes the task of taking an image from a camera pointed to the vehicle forward motion and then marking the positions of the persons that appear in the camera field of view, especially those in the vehicle path. This spatial information is then translated to the vehicle reference frame to be at disposal of other systems. For example, an alert system can monitor the vehicle speed and attitude and then signal the driver of a potentially dangerous situation or even actuate on the vehicle control system to prevent it. Automatic braking and cruise control are some systems that can use this information.

Among the many propositions for pedestrian detection systems, the most common structure is the one briefly described bellow:

Image acquisition: The first task is to acquire the image that will be used to detect the pedestrians. This function can be based on different kinds of imaging devices, most commonly a video camera. The image may be taken in color or gray-scale. Infrared is also a common choice that enables the ability to track the pedestrians even in the lack of natural or artificial illumination.

Coarse detection: The next step is to proceed with the detection of the objects in the image frame that are potential pedestrians. The objective here is to provide an initial guess and also eliminate obvious errors. One common approach is to apply a sliding window to the frame and to extract features in these Region of Interests (ROIs), followed by non-maximal suppression for eliminating repeated detections.

Feature extraction: The candidates selected in the previous process will be transformed from image fragments to a numerical or categorical collection of values, organized sequentially to form a feature vector. The resulting feature vector is then used to classify the candidate object as a pedestrian or not.

Classification: Finally, the information given by the feature vector can be classified and made available to be used by other systems or exhibition to the driver.

The implementation of an OPF-based pedestrian detection associated with HOG feature descriptor is presented in the next sections, followed by an evaluation of its performance and a comparison with linear SVM, MLP and Random Forests classifiers, using the same descriptor.

The training was done using a pre-processed dataset, also described in next sections.

Figure 3.2 displays a schematic representation of a possible solution for a pedestrian detection system, showing each sub-task, its corresponding method and an example of generated output.

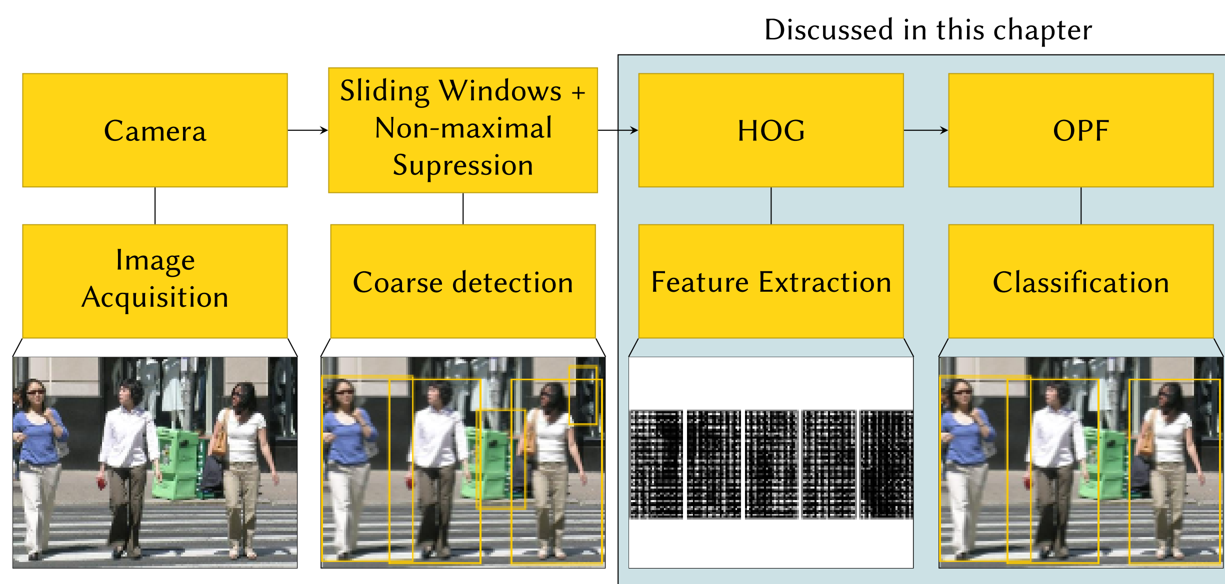


Figure 3.2 – A proposition of a pedestrian detection system using an OPF classifier. The highlited part is discussed in this chapter.

3.4 Feature extraction

This work evaluates two propositions of feature extraction methods for using with the OPF classifier. Both of them make use of the HOG as the primary descriptor; the first evaluates HOG alone and the second uses HOG in conjunction with a feature selection method, specifically, the Principal Component Analysis (PCA).

3.4.1 Histogram of Oriented Gradients

The HOG feature descriptor was introduced for pedestrian detection in 2005 in the landmark work of Dalal & Triggs (2005), as already mentioned. It became the dominant descriptor

in the field, being used in nearly all modern detectors (DOLLAR et al., 2012).

Essentially, HOG is based on evaluating well-normalized local histograms of image gradient orientations in a dense grid. The idea is that the distribution of local intensity gradient or edge orientations describes well the local object appearance and shape. In practice, it is done by dividing a search window into small regions called *cells*. For each cell, it is accumulated in a local one-dimensional histogram over several gradient directions or edge orientations for each pixel in the cell. The representation is formed by the combination of the histograms. Illumination and shadow variations can degrade the results, so a normalization of contrast is applied to increase the method robustness. This is accomplished by accumulating a measure of the histograms energy across a larger region comprised by a *block* of cells, and then using this value to normalize all the cells in the respective block. The final feature vector is given by distributing a dense, overlapping grid of blocks over the window and then concatenating the resulting histograms. Figure 3.3 shows an example of a HOG descriptor.

HOG descriptors have a number of advantages, most notably its ability to capture local shape from the gradient structure with a controllable degree of invariance to geometric or photometric transformations. If rotations or translations are much smaller than the orientation histogram bin size, they make little difference in the final representation. For pedestrian detection, it translates as a certain freedom of movement, for example, limbs can move laterally in big amounts without compromising the model, given that they maintain a more or less upright position.

This works uses a HOG implementation provided by the Open Computer Vision (OpenCV) library, using the following setup: The detection window is divided into a 8×16 grid, with each cell measuring 8×8 non-overlapping pixels. Four neighboring cells form a block in a 2×2 configuration. Blocks overlap by one cell in horizontal and vertical directions, resulting in $7 \times 15 = 105$ blocks per window. For each cell, the histogram is divided into 9 gradient orientation bins. Each bin cumulates the gradient magnitude for a given pixel. Each block results in 36 elements (4 cells $\times$ 9 bins) that are concatenated into a single vector. After concatenating all blocks, the resulting HOG feature vector will have $105 \times 36 = 3780$ dimensions.

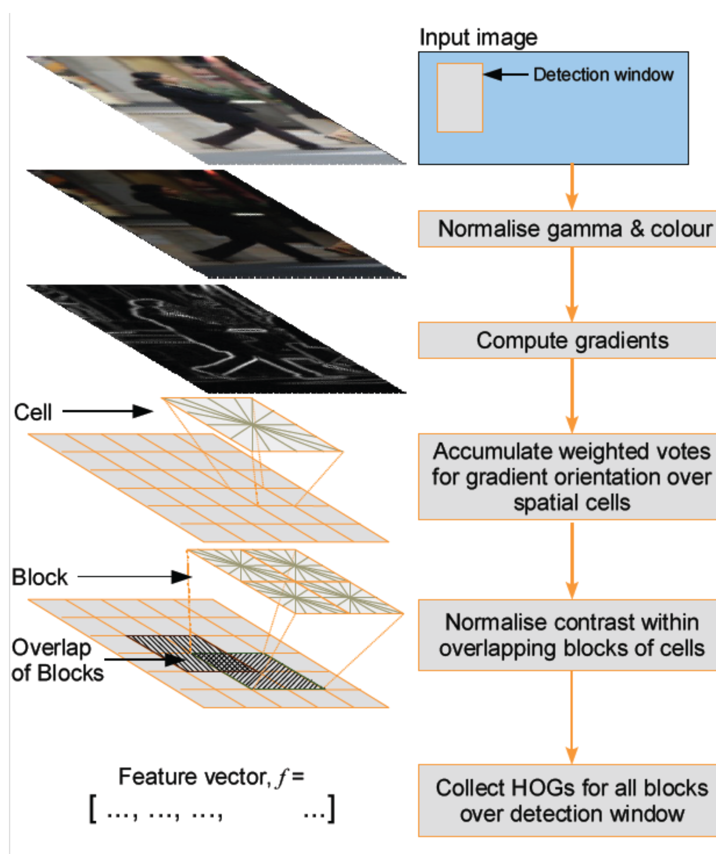


Figure 3.3 – The HOG descriptor is formed by an overlapping grid of blocks containing a number of cells, then calculating a cumulative histogram in different gradient direction for each pixel in the cell. The concatenation of the resulting histograms gives the feature vector. Local contrast normalization at block level helps to increase the invariance to changing illumination conditions.
Source: (DALAL; TRIGGS, 2005).

3.4.2 Principal Component Analysis

Although very precise, HOG descriptor feature vectors present high dimensionality (3780 dimensions in our configuration). Therefore, it can be useful to apply a dimension reduction method, like PCA, allowing faster classification times than when using full-sized vectors. This processing time reduction is a desirable effect for real-time applications.

PCA was created in 1901 by Karl Pearson (PEARSON, 1901) and developed later by Hotelling (HOTELLING, 1933). It is a statistical process that transforms a set of possibly correlated variables into a linearly uncorrelated set with equal or fewer dimensions, called *Principal Component*. The transformation is defined in a way that the first component has the largest possible variance and all the subsequent components have the next largest possible

variance if they obey the constraint of being orthogonal to the precedent components. In practical terms, PCA can be used for dimensionality reduction. Given a set with a given number of elements, PCA searches for an orthogonal basis that can describe the data in a more understandable way, mapping the data to a new coordnate system with a subset of components found by the analysis, achieving a reduction of the data processing through analysis in the new coordinate system.

PCA has many applications in fields as diverse as Signal Processing to Psychology, mostly applied as a tool for exploratory data analysis and predictive models. It is also used in ML for dimensionality reduction. The transformation performed by the PCA maps the data vectors from an original space with p variables to a new space with also p variables, now uncorrelated. However, a truncated transformation is also possible, by defining a space that has only a part of the vectors. The resulting space still maximizes the variance by minimizing the least squares error of the reconstruction.

The dimensionality reduction property can be used to speed-up classification times of dense feature vectors like the ones generated by HOG. In this work, we compared the performance of a reduced set of features obtained by applying PCA over feature vectors generated by our HOG configuration against the pure HOG ones. Both qualitative and quantitative tests (classification accuracy and processing time) are investigated. The PCA implementation used was the one found in OpenCV library (BRADSKI, 2000).

3.5 Experimental results

3.5.1 Methods used for comparison

The methods used for comparison with the OPF approach used the following parameters: SVM implementation is the one from libSVM library (CHANG; LIN, 2011). The kernel used was the linear one, as this type of kernel is the common choice used in pedestrian detection (DALAL; TRIGGS, 2005; KOBAYASHI et al., 2008). The type selected was nu-SVC, using default value for the *nu* parameter. MLP and Random Forest implementation is the one from OpenCV library.

MLP was set to use the Resilient Propagation (RPROP) training method (RIEDMILLER; BRAUN, 1993), with 3 layers in total, an input, a hidden and an output layer. The number of neurons in the first layer was set as the number of features, the hidden layer had 1000 neurons and the output layer, a single one. Random Forest was set to have a maximum number of 100 forests and a maximum depth of 25. The OpenCV version used was built with Intel™ Thread Building Blocks library. As stated in OpenCV documentation, both MLP and Random Forest implementations benefit from the library parallelization. This has to be considered for the processing times comparison, as the other methods implementations do not use any kind of parallelism. Only the classifier performance was evaluated, using a per-window approach, as described by Dollar et al. (2012). The intention is to evaluate the suitability of the classifier to be applied to pedestrian detection applications. The final efficiency of the classifier part is influenced by the detector part, however, evaluating the classifier alone is also important, as it gives us a basis to chose the more suitable classifier for a specific design choice, let say, focusing on classification performance or processing speed.

3.5.2 Data set description

The classifier training was done using a dataset composed of a combination of several publicly available data sets for pedestrian detection: The INRIA Person [1] dataset, the TUD-Motion Pairs[2] dataset, the Caltech Pedestrian Detection[3] dataset and the CVC-01 Pedestrian[4] dataset. The idea is to increase the classifier generalization by combining the different characteristic of this datasets. The final dataset consisted of 6080 pedestrian and 6080 non-pedestrian images. The images were kept on the original scale, with 128×64 pixels resolution, to match the HOG detection window, in PNG format in gray-scale. Figure 3.4 shows a sample of each class. The classifiers were modeled to perform a binary classification, considering the pedestrian images as positive cases and the non-pedestrian as negative cases.

[1] Available at: http://pascal.inrialpes.fr/data/human/
[2] Available at: http://datasets.d2.mpi-inf.mpg.de/tud-brussels/tud-brussels-motionpairs.tar.gz
[3] Available at: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/
[4] Available at: http://www.cvc.uab.es/adas/site/?q=node/7

(a) (b)

Figure 3.4 – Sample of images from the mixed dataset used in this work.
(a) A positive sample, showing a person in an arbitrary pose.
(b) A negative sample, with no person visible.

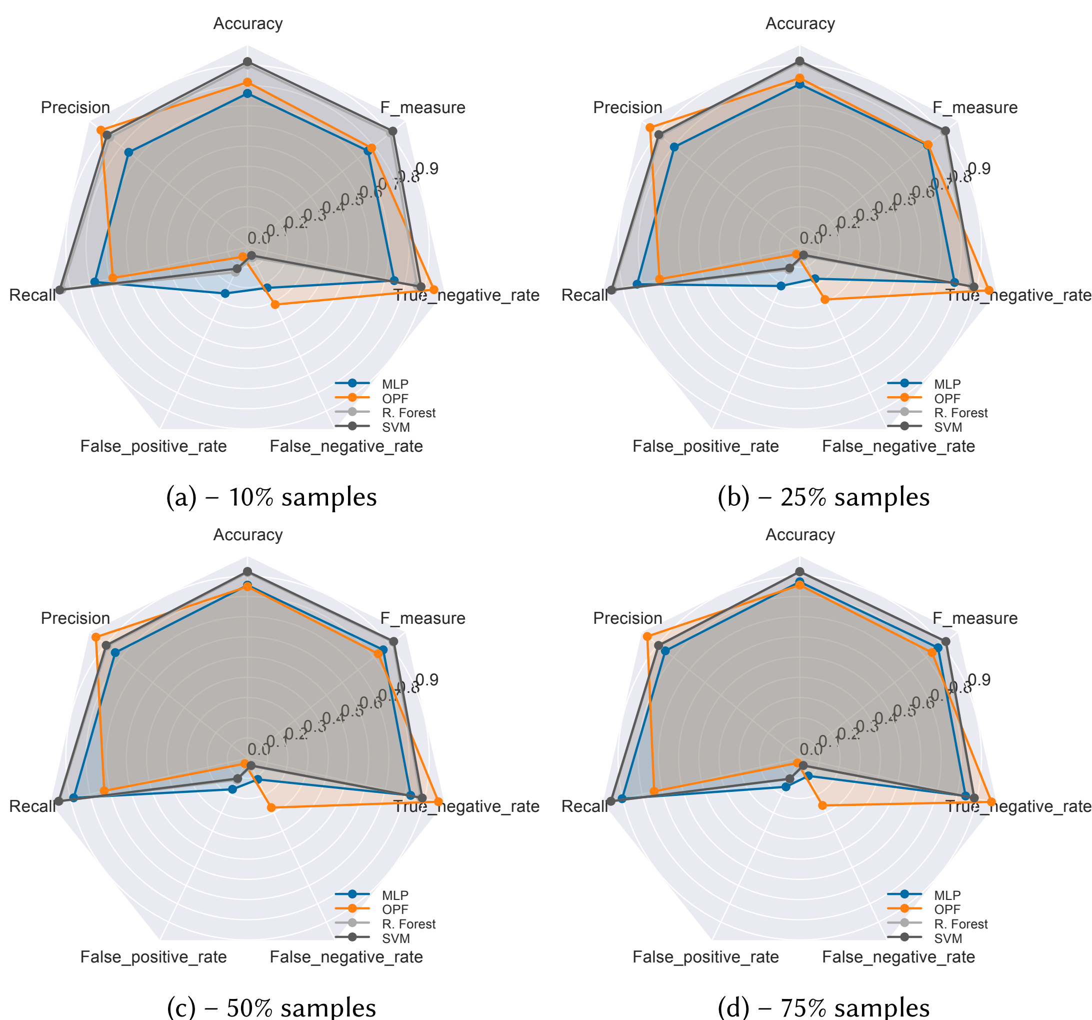
3.5.3 Results

The metrics used to evaluate the method performances were the ones based in the Confusion Matrix (SOKOLOVA; LAPALME, 2009). The time spent in training and testing phases have also been considered. They were measured using Repeated Random Sub-sampling validation with stratified sampling, keeping $50\% - 50\%$ ratio between positive and negative samples in every partition. Each method was executed 100 times with a different randomly chosen set of training and test samples, keeping the same sets for each method in each round. The final results are given by the arithmetic average of all rounds. The equipment used was the same for all methods, a PC equipped with a Intel® Core™ i7-3720QM CPU at 2.600 GHz with 8 GB RAM DDR2 memory, running Ubuntu 14.04 "Trusty Tahr".

All classifiers were evaluated using the two approaches for feature extraction, HOG alone and HOG+PCA. In order to evaluate the influence of the number of samples on classification stability, four configurations were used, with 10%, 25%, 50% and 75% of the dataset respectively. An amount of 40% of the resulting set was used for training the classifiers and other 40% used as test set. The remaining 20% were used for the training of the OPF method, as it requires an extra validation set.

The execution time for applying PCA and projecting the samples to the resulting subspace was, in average, 32 minutes. We chose to keep 95% of the original covariance, reducing the feature space from 3780 to 1271 dimensions.

Figure 3.5 shows the results for HOG descriptors and Figure 3.6 shows the results for HOG+PCA descriptors. For HOG only, SVM and Random Forests had practically the same performance, with OPF and MLP being a little less accurate. With HOG+PCA, all methods showed a drop in performance. We can notice that the OPF method was less affected, showing to be stable and more accurate than the other methods in this configuration. The MLP was stable but significantly less accurate and Random Forest completely degenerates.



(a) – 10% samples

(b) – 25% samples

(c) – 50% samples

(d) – 75% samples

Figure 3.5 – Metrics for classifying HOG descriptors.

Figure 3.7 shows the resulting Receiver Operating Characteristic space using each descriptor. As the OPF is a discrete classifier, the resulting ROC curve is a single point. To be fair, all the other methods were also set as discrete.

Table 3.2 and Table 3.3 show the processing times for training and testing stages. We can notice a significant reduction in testing time with HOG+PCA and an increase in training time, except for OPF, whose training time decreased in all situations. Random Forest showed

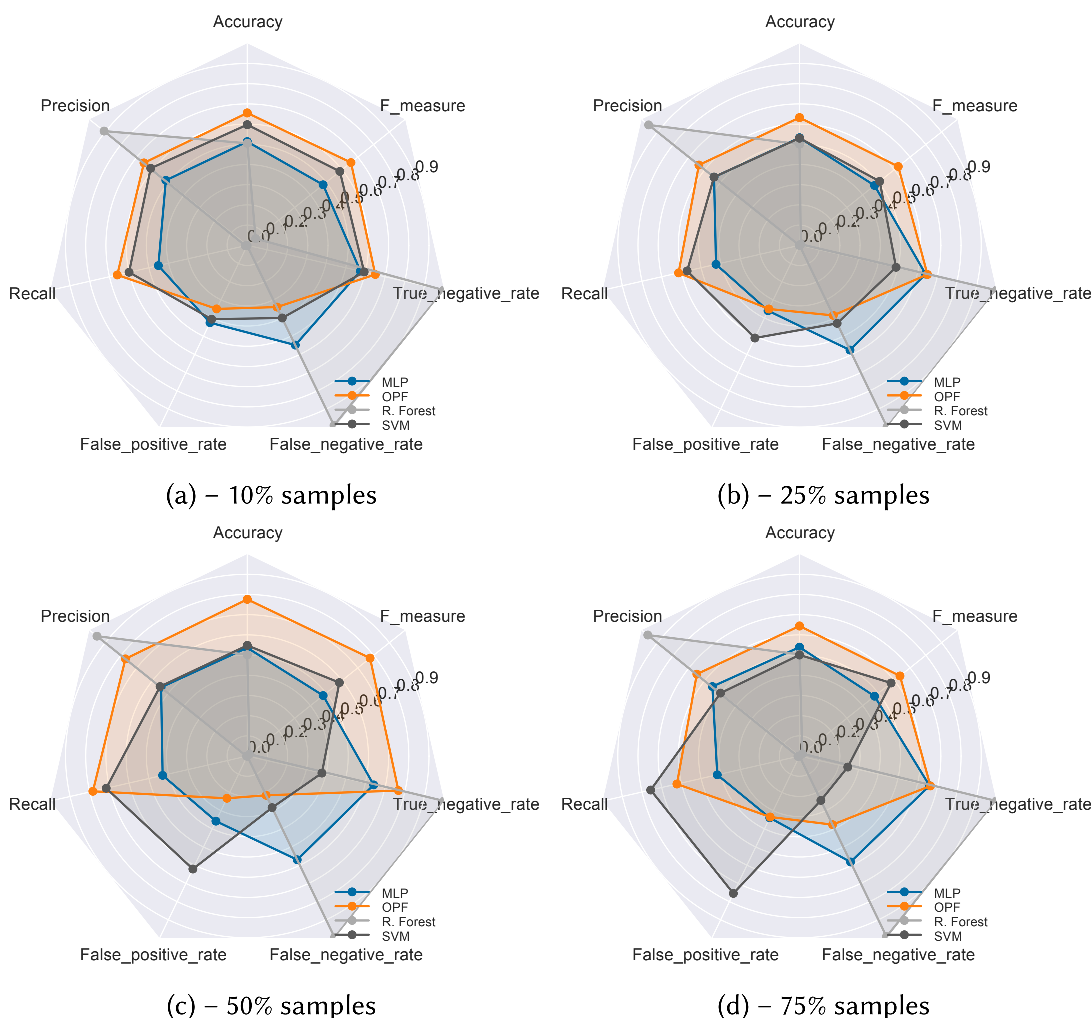


Figure 3.6 – Metrics for classifying HOG+PCA descriptors.

unmatched speed, being the faster with HOG descriptors, but given its poor performance with HOG+PCA, the results for this feature extraction method must be disregarded. OPF and SVM showed close speed results, with OPF being more accurate with HOG+PCA. When the number of samples is increased, the advantage of the TBB library parallelization in OpenCV methods is noticed; MLP and R. Forest became faster. It is also important to remark that the parameter optimization performed by the libSVM with HOG+PCA had some difficulty to converge. This can be an indicative that within HOG+PCA subspace, the linear kernel lost its generalization, bringing the necessity of testing different kernels or doing a deeper parameter optimization. This remarks the advantage of the non-parametric characteristic of the OPF, alongside its stability with dimension reduction by HOG+PCA.

Figure 3.8 shows the Accuracy histogram of each method for HOG+PCA descriptors. OPF shows to be more stable and accurate. We have to disregard Random Forest result, as it is not functional with this descriptor.

Table 3.2 – Training and testing stages processing times using only HOG descriptors.

% of samples	Method	Training Time (s)	Testing Time (s)	Time per sample (ms)
10	MLP	115.0555	1.1091	2.277
	OPF	2.9102	0.5653	1.160
	R. Forest	10.5303	0.0102	0.020
	SVM	0.8588	0.6133	1.259
25	MLP	134.0223	2.9527	2.428
	OPF	19.4376	3.5941	2.955
	R. Forest	40.4635	0.0398	0.032
	SVM	5.4974	3.8070	3.130
50	MLP	394.9813	5.4359	2.235
	OPF	79.2350	13.6569	5.615
	R. Forest	91.5122	0.0992	0.040
	SVM	20.3603	13.8870	5.710
75	MLP	791.7233	8.5056	2.331
	OPF	182.0430	31.0965	8.524
	R. Forest	1,546.5659	0.1643	0.045
	SVM	47.6002	32.2706	8.846

Table 3.3 – Training and testing stages processing times for PCA+HOG descriptors.

% of samples	Method	Training Time (s)	Testing Time (s)	Time per sample (ms)
10	MLP	22.5206	0.1757	0.360
	OPF	0.7282	0.1429	0.293
	R. Forest	2.4151	0.0047	0.009
	SVM	4.5394	0.1183	0.242
25	MLP	169.3280	0.6864	0.564
	OPF	8.1560	1.4681	1.207
	R. Forest	10.1993	0.0166	0.013
	SVM	32.1660	1.0654	0.876
50	MLP	689.3629	1.6430	0.675
	OPF	49.8310	7.1837	2.953
	R. Forest	21.2510	0.0407	0.016
	SVM	97.8980	4.7310	1.945
75	MLP	1168.1723	2.5782	0.706
	OPF	111.8964	18.5842	5.094
	R. Forest	39.1694	0.0679	0.018
	SVM	164.8542	10.9190	2.993

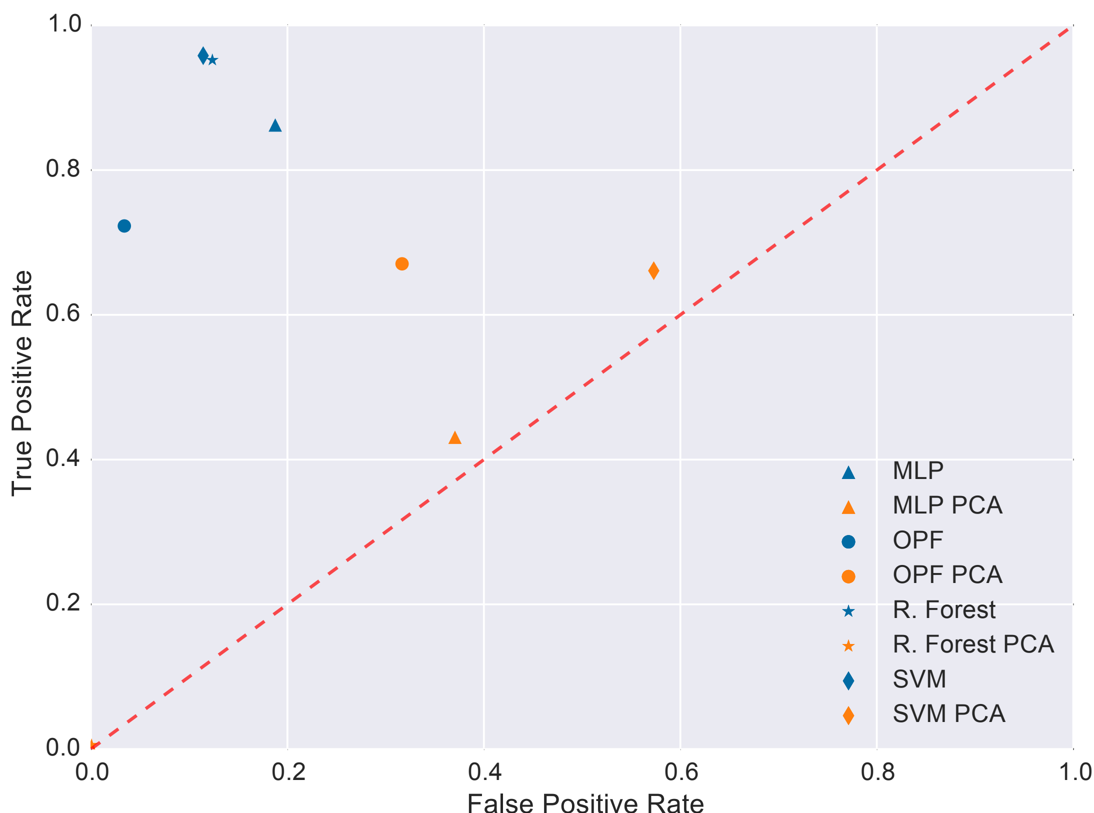
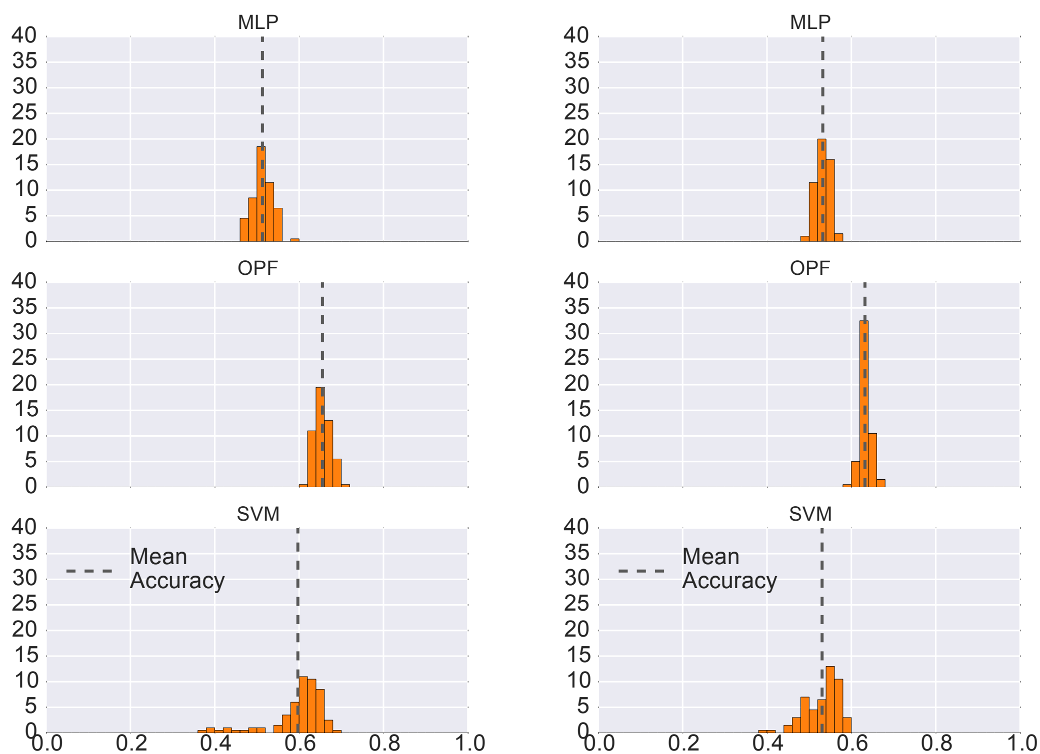


Figure 3.7 – Receiver Operating Characteristic space results for each descriptor. Best viewed in color.

Although other metrics have shown good results, the false negative rate for OPF is a bit higher than expected, with values around 20%. As false negative means that the presence of a pedestrian was not detected, this metric, in particular, is important to improve the safety and efficiency of the system. One can say that this is the most important feature expected in a pedestrian detection system.

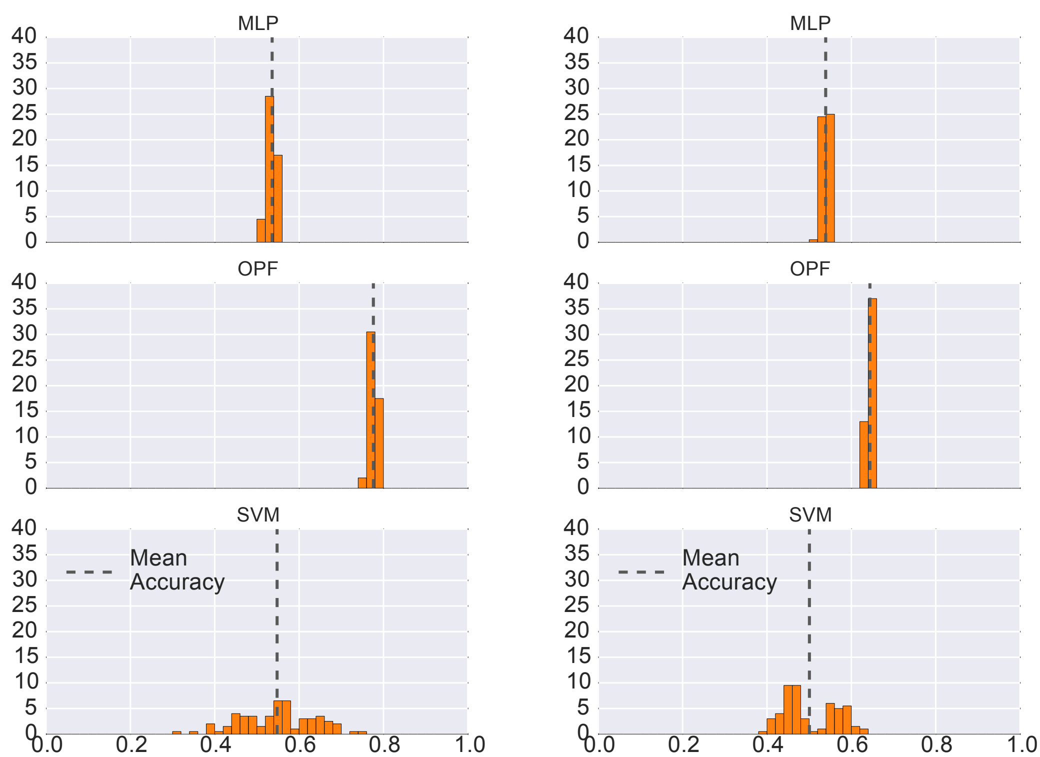
3.6 Conclusion

A novel application of the OPF classifier for pedestrian detection using HOG feature extraction alone and with PCA dimension reduction is here presented and analyzed. Its performance as compared with other methods usually applied for this task. It is important to notice that the dimension reduction done by PCA significantly influences all methods performances, OPF showing to be less sensitive. Therefore, it is possible to take advantage of the reduction in processing time without compromising too much of the classification performance. Its simplic-



(a) – 10% samples

(b) – 25% samples



(c) – 50% samples

(d) – 75% samples

Figure 3.8 – Accuracy histogram showing classification stability with 100 randomly chosen training sets for each method using HOG+PCA descriptors. Random Forest results was disregarded, as it is not functional.

ity of implementation and absence of parameters in training routine and multi-class capability make it a suitable candidate for its use in pedestrian detection applications. OPF algorithm also shows great potential for parallelism, being suitable for implementation in specialized hardware like GPUs or FPGAs, which will permit applications in real-time embedded systems.

Published work derived from this chapter:

DINIZ, W. F. S. et al. Evaluation of optimum path forest classifier for pedestrian detection. In: **2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)**. Zhuhai: IEEE, 2015. p. 899–904. ISBN 978-1-4673-9675-2. Disponível em: <doi://10.1109/ROBIO.2015.7418885>.

Chapter 4

Self-organizing Evolutionary learning for supervised OPF training

" 'Ipsa scientia potestas est'. For also knowledge itself is power."
— Francis Bacon

4.1 Introduction

Although supervised learning procedures are the most common used techniques to prepare a classifier, some cases may present challenges that render this technique less efficient. For example, some applications using Deep Learning need a huge amount of data from which the network will infer its parameters. Image classification, for example, has a relatively easy source of material, since millions of images are already available in public databases. For other applications, labeled data may not be available or very difficult to produce, thus, the training must rely on unsupervised methods such as clustering or data mining. Semi-supervised learning is also an alternative for cases where partial knowledge is available. Also, during its classification phase, a sample belonging to an unknown class may be wrongly classified as one of the previously known ones, leading to errors and potentially dangerous consequences.

Regardless of the learning procedure nature (supervised, unsupervised of semi-supervised), introducing new classes to a previously trained classifier, without implying a complete retraining, is a challenging task. Some methods were introduced to provide incremental learning, i.e., the ability to learn new classes and redefining the decision function while in classification mode. A true incremental learning method acts by modifying the classifier whenever it founds a sample that it not recognizes, so this new knowledge is added, modifying the previously trained data.

An autonomous robotic system, moving in real environments, needs to process the perception signals, acquired from its sensors, turning the data into knowledge to be used to continuously increment its classification machine. The diversity of stimuli from real environments demands more sophisticated methods than what is usual in the traditional laboratory training and testing. A generic approach for an autonomous robot learning from dynamic changing scenarios is proposed by Keysermann & Vargas (2015), to cope with unknown elements in order to create new classes, adopting clustering of data and associative incremental learning.

Nonetheless, in some cases, modifying the original classifier is not possible. Hu et al. (1997) proposed an approach for improve Electrocardiogram (ECG) classification using a technique called Mixture-Of-Experts (MOE). It consists of a Global Expert (GE) classifier response combined with a Local Expert (LE) classifier to reach a consensus over the actual classification. The GE is a classifier trained with available ECG data while the LE is produced from an specific patient. The goal is to enable automatic adaptability to patient specific variations in the ECG signal that may cause failure to identify an anomaly. The advantage of this approach is that it does not modify the original classifier data, allowing its use in situations in which it is not possible to modify the actual classifier.

Whatever is the adopted online configuration, the system will begin with a regular training whose parameters and classes are going to be updated later when in effective use. Then an efficient training method is necessary to face the real time restrictions. Both types of incremental training must be performed online when the classifier is already in use, so the impact of this procedure may affect the whole system. Depending on the type of system, such impact may cause undesirable situations, so it is desirable that the online training has a high performance. FPGAs, with their reconfiguration ability, may be the suitable devices to such tasks. The new training procedure for an OPF classifier proposed in this chapter is the first step in a broader goal to achieve a highly adaptable and reliable classifier framework for embedded systems.

Recalling the classic algorithm for training a supervised OPF classifier, it uses three sets of samples divided from a universe of known labeled instances of the problem to be classified. Then, from one of these sets, the training set, a classifier is formed by constructing a complete graph using the samples as nodes and their dissimilarity as edges. The resulting classifier graph will have as many nodes as samples in the training set. The other two sets are used in a learning

procedure to enhance the classification performance. It is clear to conclude that the number of nodes in the classifier affects the classification processing speed: More nodes means more items to compare.

After publishing the original OPF algorithm (PAPA, 2008; PAPA et al., 2009), efforts were made to increase OPF processing speed. In the first one, as already presented, Papa et al. (2010) used a theoretical property of OPF classification. Introducing a step to construct the classifier as an ordered set by their associated costs, the comparison is done until the associated cost of the next node is greater than the current minimum. Therefore, it increases the chances of finding the correct label without having to compare all nodes. The same author also proposed a second enhancement (PAPA et al., 2012), a pruning algorithm that selects the most relevant samples in the training set and cuts out irrelevant ones, thus decreasing the number of nodes in the classifier.

Evolutionary Computation is an AI subfield that mostly uses iterative progress on a population of candidate solutions to find its goal. It can be understood as global optimization methods with meta-heuristics or stochastic characteristics. Many methods are inspired by biological evolution mechanisms, hence the name. It was introduced in the 50s and consolidated in the next decades with works in the now called Evolutionary Computation *dialects*: *Evolutionary Programming* (FOGEL et al., 1966), *Genetic Algorithms* (HOLLAND, 1975), and *Evolution Strategies* (RECHENBERG, 1971; SCHWEFEL, 1977). In the 90s, a new dialect, *Genetic Programming* was introduced, as well as nature-inspired algorithms started to play an important role in the field.

This chapter presents the Self-Organizing Evolutionary Learning (SOEL), a new proposition for building an OPF classifier; instead of constructing a complete graph with the training samples and then finding the MST and finally pruning nodes, we propose that the graph starts with few nodes and then, using an evolutionary algorithm, grows while fitting to the decision boundaries. The goal is to increase the processing speed in the classification stage using a classifier with a smaller number of nodes. The growth is directed in a manner that keeps the graph small enough to enhance its processing speed compared to the classical OPF without compromising classification performance. The inspiration comes from a known Evolutionary Computation algorithm, the SOM. It is expected that the SOM capacity of revealing the structure of the feature space combined with the same ability of the MST structure used in OPF

allows minimizing or even to prevent loss of accuracy in the smaller graph. In next sections, an overview of SOM presents the characteristics that inspired SOEL development, followed by the details of the new algorithm implementation and its analysis in comparison with the classical OPF training algorithm.

4.2 Self-organizing maps

Self-organizing Maps or Self-organizing Feature Maps, were introduced as an alternative method for building ANNs (KOHONEN, 1990). It uses an unsupervised competitive learning procedure to produce a low-dimensional (usually two-dimensional) representation of the input space called map. A SOM is composed of several elements called nodes or neurons. Each node has an associated weight vector with the same dimensionality as the input space and is represented by a point in the map space.

The evolutionary training procedure consists of presenting samples from the data space and finding the node whose weight vector is closest, i.e. have the smallest distance, to the sample. This node is called Best Matching Unit (BMU). The weight vectors of the BMU and neighboring nodes inside a region are adjusted towards the sample input vector. The intensity of this adjustment depends on the iteration step and distance of the nodes in relation to the BMU, decreasing over each iteration. The update formula for a weight vector $\vec{w}$ of a node with index v is given by:

$$\vec{w}_v(t+1) = \vec{w}_v(t) + \theta(b,v,t) \cdot \alpha(t) \cdot (\vec{u}_s - \vec{w}_v(t)), \tag{4.1}$$

where t is the iteration index, b is the BMU index in relation to the sample s, $\vec{u}_s$ is the sample input vector, $\theta(b,v,t)$ is the neighborhood function that returns the intensity factor in relation to the distance of b and v in the iteration t, and $\alpha(t)$, called *Learning Rate*, is a monotonically decreasing coefficient that restrains the amount of learning over time. According to the author, the learning process is partially inspired by how the human brain organizes sensory information in different parts of the brain. In these structures, sensory information is almost directly mapped to the neural network. The brain learning capacity is simulated by the neighborhood and learning rate functions. At the beginning, there is no knowledge, thus the neighborhood is

broad and the adjustments happen on a global basis. The learning rate is also big, then the adjustment intensity is high. Over time, the neighborhood shrinks to have local influence over the BMU itself and its closest nodes, while the learning rate is smaller and so the adjustment magnitude. Figure 4.1 illustrates the learning procedure.

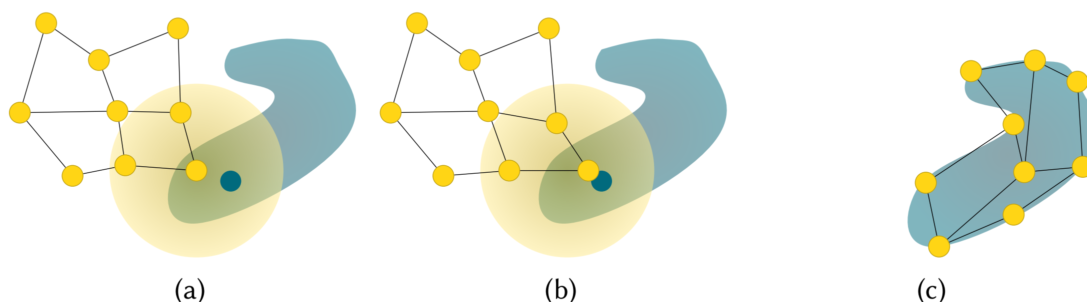
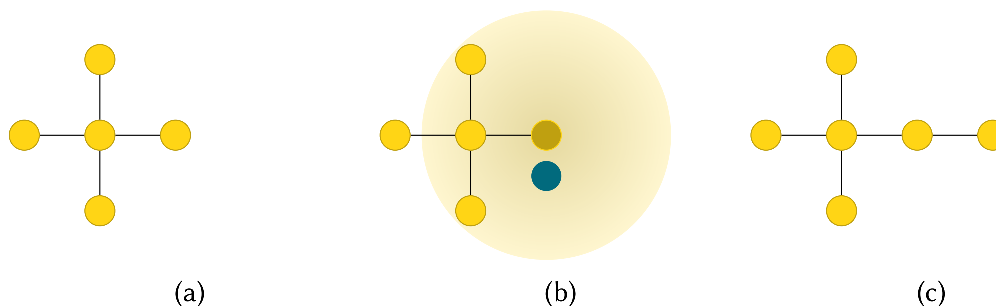


Figure 4.1 – The SOM's training procedure is a competitive learning that modifies the network's shape as long as the training goes.
(a) An input vector, represented by the blue dot, is selected from data space and presented to the network. The BMU, represented by the hollow yellow dot, is determined, as well its neighborhood function, represented by the yellow shadow.
(b) The BMU and the nodes inside its neighborhood region are adjusted towards the input vector, according to their distance and the learning rate.
(c) As the learning proceeds, the nodes assume a distribution close to the data space shape.

Originally, (KOHONEN, 1990) used randomly generated weight vectors for the initial placement of the nodes. Later, it was discovered that using evenly sampled vectors from the subspace formed by the two largest principal component eigenvectors speeds up the learning because it is a good approximation of the final weights, also making possible exact reproducibility of results (CIAMPI; LECHEVALLIER, 2000). However, the advantages of principal component initialization are not universal, comparisons with the stochastic method have shown that for nonlinear datasets, the latter performs better while principal component initialization is best suited for linear and quasilinear sets (AKINDUKO et al., 2016).

SOM classification performance is highly influenced by the number of nodes in the network. Growing Self-organizing Map (G-SOM) were proposed, aiming to solve the issue of determining an optimal map size (ALAHAKOON et al., 2000; ALAHAKOON et al., 1998). In this method, the network is started with a small number of nodes and, as the learning progresses, the network grows over the boundaries according to a defined heuristic. A parameter called Spread Factor is introduced, to control the network growth. The training is similar to SOM to a certain degree. Input vectors are presented to the network, and then a BMU is found and the weight vectors of the nodes within a neighborhood centered at the BMU are adjusted, including

the BMU itself. The nodes have an error value that is increased every time a node is chosen as BMU. This error is the L_1 normalized difference between the input vector and BMU weight vector. When this error surpasses a previously defined threshold value the last presented node is added to the network, growing the network. The growing process is repeated until it reaches a minimum value. The process is exemplified in Figure 4.2.



(a) (b) (c)

Figure 4.2 – The G-SOM learning procedure is also a competitive learning based-one, but now the network is initialized with a small set of nodes, growing as necessary.
(a) A map is initialized with 5 nodes.
(b) As a input is presented, the BMU is determined and its weight vector and their neighboors' are adjusted. A error value is accumulated with the difference between the input's vector and BMU's weight vector.
(c) If the error value surpasses a threshold, a new node is spawned to the network. The amount of growing is controlled by a Spread Factor parameter.

The number of nodes that are spawned is controlled by the spreading factor and also depends on the adopted topology for the network. G-SOM has been used in data mining, classification, and clusterization applications.

4.3 Method description

The main idea of the proposed SOEL learning algorithm is to get advantage of the OPF ability to infer decision boundaries from data through the MST structure and combine it with the clusterization capacities of SOM and G-SOM. Like the latter, it starts the classifier with few seed nodes, instead of a complete graph from the training set that classical OPF uses. Then, applying an evolutionary learning process, it lets the graph growing, while fitting the graph to the decision boundaries in feature space. This is the difference between the methods, there is no mapping, the structure used is the embedded feature space within the graph.

The algorithm is divided into four phases, each one covering an aspect of the evolutionary process, as indicated in Figure 4.3. Algorithm 4.1 shows the complete algorithm pseudocode for the evolutionary learning. The algorithm phases are briefly introduced below:

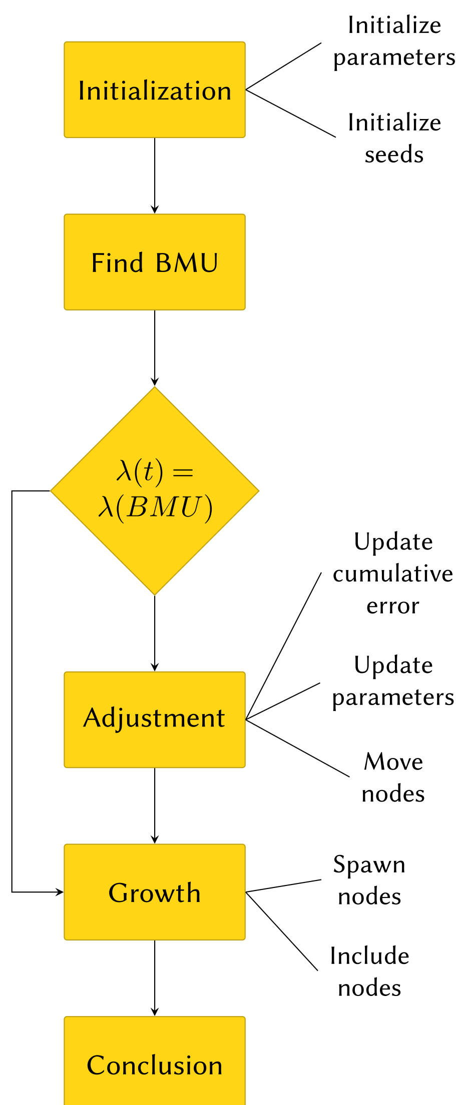


Figure 4.3 – Overview of the SOEL algorithm structural hierarchy.

Initialization: Consists of the seeds determination, according to a defined policy. After the seeds are determined, the training process starts applying to OPF fitting algorithm (Algorithm 2.1) to initialize the classifier graph, containing only the seeds at this point.

Adjustment: After initializing the seeds, the training procedure follows to the Adjustment phase. Each sample in the training set will be presented and classified using the OPF classification algorithm. The node that wins the classification is marked as the current BMU in the iteration. The classifier nodes of the same class of the BMU lying inside the neighborhood have their feature vectors adjusted in function of the distance between the stimulus and the BMU as

well considering the learning rate for the current iteration.

Growth: After the adjustment of the nodes, the current BMU accumulated error is verified against a previously defined threshold. The accumulated errors give a measure of the graph responsiveness. If the threshold is exceeded the graph grows by spawning a new node according to a set of rules. The new set of nodes is then fitted again with the OPF fitting function, before proceeding to the next iteration.

Conclusion: After all training samples are presented to the classifiers, one last fitting process is performed to guarantee that any new node that was spawned in the last moments gets correctly connected to a prototype.

The *Initialization* phase cover the lines 2 to 6 of Algorithm 4.1. It starts with the seed determination which is important because it affects how many nodes the graph will have in its final form. Once the seeds are determined, the remaining phases are the same for both policies. Two different policies are here proposed: Random Seeds Policy (RSP) and Prototypes as Seeds Policy (PSP).

RSP consists of randomly picking samples from the training set until the classifier graph contains exactly one node for each class. The PSP variation consists of using the prototypes found by a run of the OPF fitting algorithm over the training set as seeds. As the prototypes are nodes close to the decision boundary, starting the network from them may lead to a more precise classifier in some particular cases, a behavior also shown by SOMs. Each variation has their own characteristics that are discussed in Section 4.4.

Once seeds are determined, next phase starts by fitting them to obtain an initial classifier C. Now the algorithm is ready to present the training set to the classifier and proceed with the *Adjustment* phase. This phase comprises lines 9 to 21 in Algorithm 4.1. The first step is to determine the BMU, which happens in the function *Find_BMU*. This function is a modified version of the OPF classification function that returns the label $\lambda(t)$ assigned to the sample and also the index of the node that had won the competition process and consequently transmitted its label. This is the BMU node.

Next, the sample true label is compared with the assigned one, to assess if the BMU is of the same class, marking a correct classification, or not. The algorithm branches at this point. The positive case indicates that in the BMU region, the representativity is existent, just needing an adjustment to register the new information. The adjustment is applied to all nodes

Algorithm 4.1 The SOEL algorithm applies an evolutionary learning procedure inspired by G-SOM algorithm. Differently of the traditional OPF training, the graph is started with a few seed nodes, instead of a complete graph. Then, the nodes in the training set are presented to the classifier as stimuli that will force the graph to grow and evolve over time, fitting the graph to the decision boundaries in the process.

Require: Training set T, label map λ, cumulative error map E, seed determination policy, neighborhood radius σ, learning rate ϕ, distance function $d(.,.)$, error function $\epsilon(.,.)$, error threshold ξ.
Output: Classifier C.
Auxiliary: Counter i.

```
 1: function SOE_LEARNING(T)
 2:     Determine Seeds set (S) according to the policy
 3:     Initialize C from S
 4:     Initialize φ_0
 5:     Initialize σ_0
 6:     Initialize E with all values equal to 0
 7:     i ← 0
 8:     for all t ∈ T do
 9:         i ← i+1
10:         OPF_Fitting(C)
11:         BMU ← Find_BMU(t,C)
12:         if λ(t) = λ(BMU) then
13:             E(BMU) ← E(BMU) + ε(t,BMU)
14:             Update φ_i
15:             Update σ_i
16:             for all c ∈ C do
17:                 if λ(c) = λ(BMU) then
18:                     β = d(c,BMU)
19:                     if β < σ_i then
20:                         α ← d(t,BMU)
21:                         Calculate ψ using α and σ_i
22:                         Adjust c, using φ_i and ψ
23:             if E(BMU) > ξ then
24:                 E(BMU) = 0
25:                 Spawn a new node in C
26:         else
27:             Insert t in C
28:     OPF_Fitting(C) return C
```

of the same class of the BMU that are inside the current iteration neighborhood radius σ_i. The neighborhood radius is centered on the BMU and all nodes inside this region, BMU itself included, have their feature vectors adjusted, reflecting that they move in the stimulus direction. The adjustment intensity is controlled by two components, ϕ and ψ that are, respectively, the learning rate and the distance decay factor. The adjustment process is illustrated in Figure 4.4.

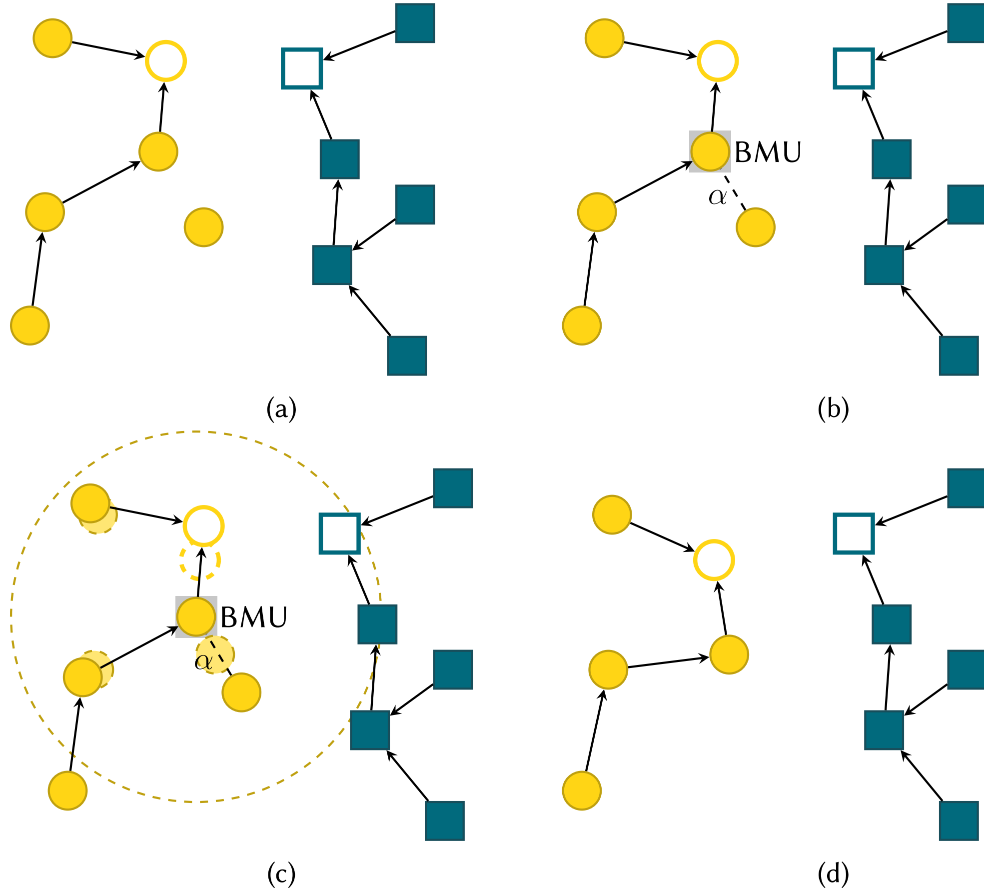


Figure 4.4 – The node adjustment propagates the knowledge acquired by a stimulus to the nodes of the same class of the BMU that lie inside a neighborhood.
(a) A stimulus is presented to the graph.
(b) The BMU node is determined and the stimulus strength α is calculated.
(c) The nodes of the same class of the BMU inside the neighborhood radius move towards the stimulus.
(d) The new graph with the adjusted positions.

The classification error ϵ is represented by the difference between the L^1 norm of the sample and BMU feature vectors difference:

$$\epsilon(\vec{v},\vec{w}) = \sum_{i=1}^{N} |\vec{v}_i - \vec{w}_i|, \tag{4.2}$$

where $\vec{v}$ and $\vec{w}$ are the respective N-dimensional feature vectors. The nodes have an accumulated error value that is updated whenever they are selected as BMU. This express the fact that it is desirable to keep the classification error low, that is, have a minimal quantity of nodes in the classifier graph that is able to connect to unknown samples in an optimal way. Therefore, whenever a node is selected as BMU the error is accumulated. If during a classification the accumulated error exceeds a previously defined threshold, ξ in the algorithm, it means that the region needs new nodes to increase its representativity. When this happens, a new node is added to the classifier. In the next iteration, the classifier is fitted again, thus ensuring that the trees are always optimum-path ones. Notice that in this process, a node that was a prototype in the previous iteration may lose this status, when a new node being spawned closer to the decision boundary. Remember that the definition of prototypes in OPF is nodes closer to the boundary. Also, notice that the determination of ξ value affects how much the graph grows. Smaller values lead to more nodes being spawned. This concludes the positive case of the branch. The node spawning process is illustrated in Figure 4.5.

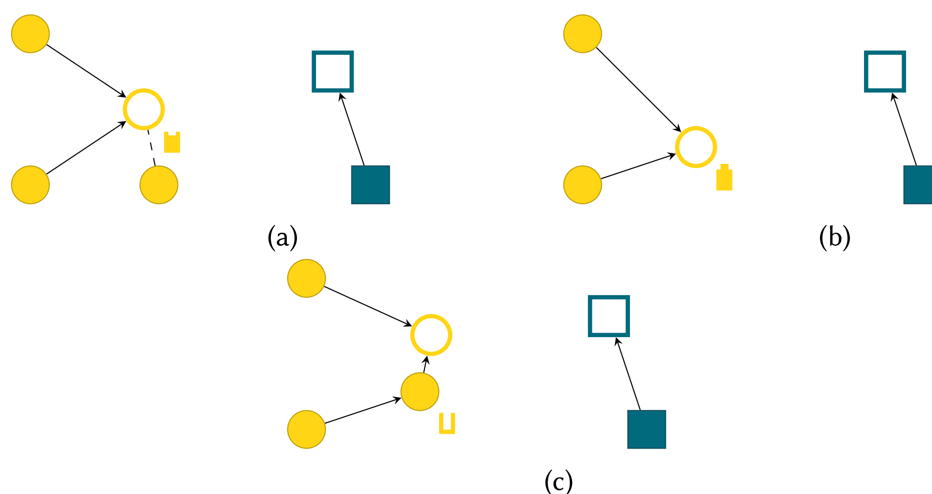


Figure 4.5 – New nodes are spawned as the error threshold is exceeded.
(a) The node selected as BMU has already accumulated errors (the small *bucket* besides the node). Nonetheless, it is selected as BMU one more time.
(b) After the adjustment phase, the cumulative error now exceeds the threshold (the bucket overflows). A new node is to be spawned in the region near the node so they can share the representativity thus decreasing the classification errors in the next iterations.
(c) The new node is closer to the decision boundary, so in the next iteration, it gains prototype status while the previous BMU loses it. Its cumulative error is also reset to 0.

Notice that all nodes in the classifier have an associated cumulative error. They were not shown in the figure for simplification.

The negative case, when the selected BMU is not of the same class of the sample, means that there is a boundary between the stimulus and the BMU that was not discovered yet. The

procedure is simply to include the sample in the network because this stimulus is closer to the true boundary than the current BMU. This step together with the new node spawning discussed before comprises the algorithm *Growth* phase.

Finally, the *Conclusion* phase applies the last fitting after all samples have been presented to the classifier. If this phase is not performed, any node that was included or spawned in the last iteration will be incorrectly linked or left alone. With the end of the Conclusion phase, the classifier is ready to identify new samples. Next sections will detail the parameters that affect the graph evolution.

4.3.1 Node adjustment determination

The adjustment intensity takes into consideration the strength of the stimulus, represented by the distance between the sample and the BMU. The further they are, more adjustment is necessary, so it is directly proportional to the distance magnitude. The distance of the node to the BMU also is taken into consideration, but now inversely proportional, meaning that the nodes closer to the stimulus move more than the further ones.

The adjustment of a node with a feature vector $\vec{v}$ in relation to an stimulating sample with feature vector $\vec{w}$, in a given iteration, is given by the following equation:

$$\vec{v}_{t+1} = \vec{v}_t + \psi \cdot \phi \cdot (\vec{w}_t - \vec{v}_t), \tag{4.3}$$

where ψ is the adjustment factor in relation to stimulus strength and neighborhood radius and ϕ is the learning rate value in the current iteration.

Neighborhood radius determination

The neighborhood radius is used to find which nodes must be adjusted. Whenever an adjustment is made, only the nodes inside this radius, centered on the BMU, are adjusted. The

initial value of the radius is determined at the Initialization phase and progresses as an ordinary exponential decay of the form:

$$\sigma_i = \sigma_0 \cdot \exp(-\frac{i}{\tau}), \tag{4.4}$$

where τ is the decay mean lifetime defined as:

$$\tau = N \cdot \ln(\sigma_0) \tag{4.5}$$

Figure 4.6 shows how the functions σ and τ behave in a given configuration. The decaying of the neighborhood radius over time simulates the learning progress. At the beginning, the graph does not have any knowledge, so any stimulus influences a high number of neighboring nodes to the BMU. As the learning progresses, prior acquired knowledge represented by stimulus in a region already trained influences less and less nodes, until full saturation when just the BMU itself is adjusted.

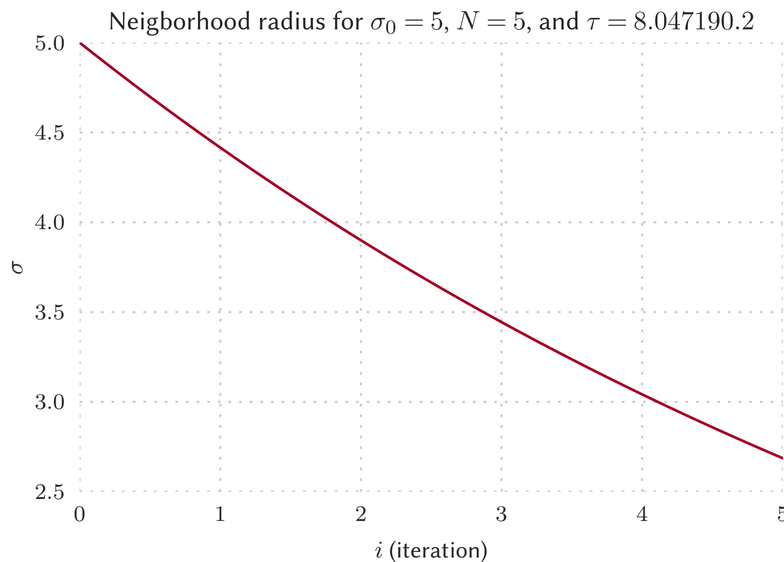


Figure 4.6 – The neighborhood radius function σ is an exponential decay tied to the time constant τ.

Once the neighborhood radius is determined, the adjustment intensity ψ to be applied to nodes within BMU neighborhood radius is given by a Gaussian function in the form below:

$$\psi = \exp(\frac{-d^2}{2 \cdot \sigma^2}), \tag{4.6}$$

where d is the Euclidean distance of the node to be adjusted to the BMU. Figure 4.7 shows how the adjustment value changes in function of σ and distance.

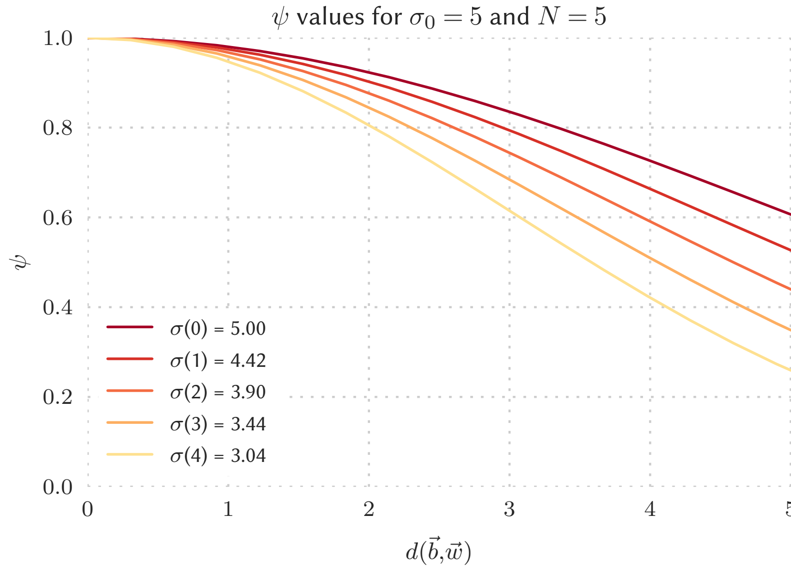


Figure 4.7 – The node adjustment factor is given by a gaussian function clipped at the σ value. It depends on the iteration as well as the corresponding σ for that iteration and the distance between the node and the BMU.

Learning Rate determination

The learning rate also decreases exponentially as the learning progresses. This reflects the fact that in later stages of the training process, the amount of adjustment will be smaller, as the network has already acquired knowledge in the previous stages. Therefore, the learning capacity decreases in time. The learning rate is defined, like the neighborhood radius, as:

$$\phi_i = \phi_0 \cdot \exp(-\frac{i}{N}) \tag{4.7}$$

The initial value is arbitrarily defined at the beginning, as a small positive real number. It controls the amount of adjustment that a graph will face as the learning progresses. Figure 4.8 shows an example of learning rate variation.

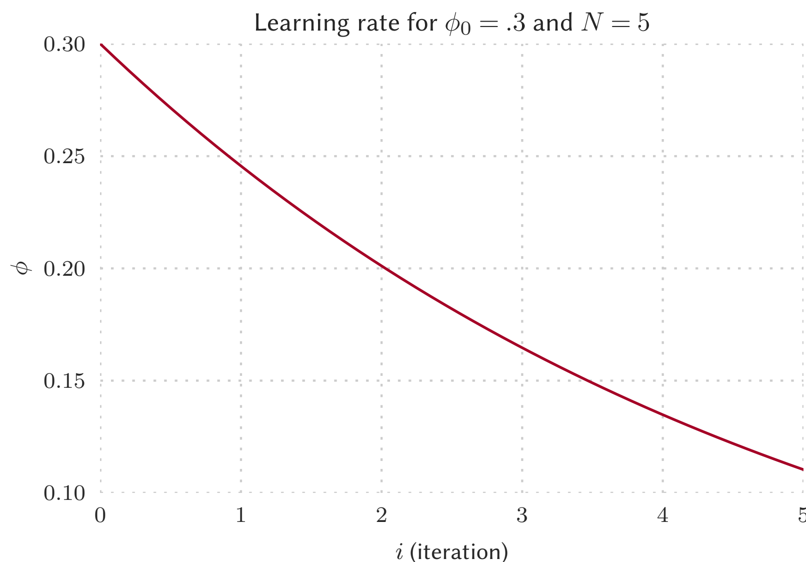


Figure 4.8 – The learning rate controls the amount of adjustment over an entire learning process. It is arbitrarily defined at the begining and decays exponentially. This behavior is intended to mimic the brains ability to learn more of a given subject at the begining stages.

4.4 Experimental results

4.4.1 Metrics and performance indicators

The experimental validation was done using both qualitative and quantitative metrics based on Confusion Matrix, for multi-class classification Sokolova & Lapalme (2009). For statistical significance, each method was evaluated using Repeated Random Sub-sampling, using 100 instances, randomly choosing samples for training, test and evaluation sets, using the same set for each method in the same instance. The distribution used was 40% of the samples for the training set, 40% for the test set and 20% for the evaluation set with stratified sampling, i.e., keeping the classes with the same distribution as the universe set, to prevent bias in cases of highly unbalanced class distributions.

The comparisons were made using two learning methods provided by the libOPF library for the classical training algorithm and the two initialization policies for the SOEL algorithm. The focus is to achieve processing time reduction in classification phase, thus, the processing time was measured to provide the quantitative data. The qualitative data is given by the comparison of the performance metrics, to observe if and how much compromise in classification

performance the new algorithm imposes. The number of nodes for the reduced graphs was also compared. A PC equipped with an Intel® Core™ i3-550 at 3.200 GHz CPU, 4 GB RAM DDR2 running Ubuntu 16.04 “Xenial Xerus” was the configuration used to evaluate the new algorithm.

4.4.2 Datasets description

The system was tested by running the classification in five different datasets. The first five were picked from the publicly available Machine Learning repository of University of California Irvine (LICHMAN, 2013). The data sets D1, D3, and D5 are originally from Computer Vision (CV) applications, with different descriptors used to generate the feature vectors. Using these data sets will permit to analyze the performance of the OPF training algorithms in diverse CV scenarios. Table 4.1 compiles each dataset characteristics. Datasets with a different number of classes and attributes are used to assess how this variation affects both the software and hardware versions.

Table 4.1 – Dataset descriptions

Id	Name	# attr.	# classes	# samples
D1	Brest Cancer Winsc. (Diag.)	9	2	569
D2	Glass Identification	9	6	214
D3	Image Segmentation	19	7	2310
D4	Iris	4	3	150
D5	Parkinsons	22	2	197

4.4.3 Results

Table 4.2 shows the compilation of results of the four methods, showing the achieved accuracy, the training, the testing time and the number of nodes in the classifier.

We can observe a drastic increase in training time, what is expected as the SOEL algorithm is more complex, and a drastic decrease in testing time. The performance loss due to using reduced graphs was very small, rarely surpassing a variation of 2%, except for dataset D5, in which performance dropped by approximately 5%. Dataset D1 showed actually better

Table 4.2 – OPF learning algorithms comparison

Dataset	Method	Accuracy avg (max)	Training time (ms)	Testing Time (ms)	# of nodes
D1	Classical OPF	0.949 (0.974)	13.79	1.24	272.00
	Aggl. Learning OPF	0.950 (0.978)	10.12	1.32	280.82
	SOEL Random Seeds	0.955 (0.989)	91.49	0.20	31.20
	SOEL Proto. Seeds	0.961 (0.985)	83.20	0.30	48.72
D2	Classical OPF	0.893 (0.919)	1.51	0.26	83.00
	Aggl. Learning OPF	0.895 (0.923)	2.05	0.30	98.18
	SOEL Random Seeds	0.876 (0.919)	7.38	0.14	54.00
	SOEL Proto. Seeds	0.884 (0.912)	20.36	0.28	86.83
D3	Classical OPF	0.984 (0.990)	238.64	24.02	924.00
	Aggl. Learning OPF	0.984 (0.989)	180.65	24.54	955.95
	SOEL Random Seeds	0.976 (0.982)	6144.15	12.76	425.76
	SOEL Proto. Seeds	0.979 (0.984)	8517.56	17.66	549.07
D4	Classical OPF	0.962 (0.988)	0.62	0.07	60.00
	Aggl. Learning OPF	0.962 (1.0)	0.43	0.07	61.95
	SOEL Random Seeds	0.958 (1.0)	2.98	0.03	29.84
	SOEL Proto. Seeds	0.966 (0.988)	2.97	0.04	34.23
D5	Classical OPF	0.823 (0.925)	1.81	0.24	77.00
	Aggl. Learning OPF	0.824 (0.925)	1.39	0.25	85.14
	SOEL Random Seeds	0.758 (0.875)	1.80	0.05	14.14
	SOEL Proto. Seeds	0.794 (0.887)	8.44	0.182	49.55

performance with the new algorithm. These results show that the primary objective of reducing processing time in classification without significant performance degradation was achieved. Regarding the number of nodes in the classifiers graphs, the reduction varied, from a minimum of 42% to a maximum of 88%. Given the random characteristics of the SOEL algorithm, inherent to evolutionary methods, this variation is also expected.

Regarding the initialization policies, PSP generated bigger networks and showed slightly greater classification performance than RSP again in the 2% range, except for D5, where the difference where near 3.6%.

4.5 Conclusions

This chapter presented a new learning algorithm for an supervised OPF classifier, inspired by evolutionary methods such as SOMs and G-SOMs. The main objective was to reduce the classification time by generating smaller networks than the ones generated by the classical

OPF learning algorithms.

The reduction achieved in the number of nodes in the final classifier was up to 88%, which reflected in classification proportionally smaller. The degradation in performance was very small, in order of 2% except for on dataset, with another dataset showing an improvement in performance.

Chapter 5

FPGA based framework for classification with OPF

"Educating the mind without educating the heart is no education at all."
— Aristotle

5.1 Introduction

Advances in integrated circuits technology has lead to miniaturization and consequently, an increase in the number of transistors in a single chip, as stated in Moore's law. However, as the size of the elements progressively gets smaller, the law limits seem to be near reached. The energy consumption is another consideration that can limit the advancement in the performance of microprocessors and other integrated circuits. Pipeline enhancements and clever memory access design also helped but the main firepower currently relies on multiple cores for parallelism. Microprocessors are now built with several computational cores, giving them the ability to process many tasks in parallel. Even GPUs started to be used to perform general computation, pushing hardware parallelism limits even more, increasing the computational power/energy consumption ratio.

As FPGAs are composed of several logic blocks linked by a series of reconfigurable interconnection, they possess remarkable parallelism potential. They are also a class of integrated circuits whose main feature is the possibility of being reconfigured by the user to perform specific tasks in a customized hardware logic. All these characteristics make them interesting candidates to act as performance accelerators, especially for embedded systems.

FPGAs have some unique advantages over general-purpose CPUs. Firstly, instead of a fixed architecture interpreting commands from a programming language, they are reconfigurable

hardware logic. Therefore, when an algorithm is transcripted to an FPGA description, it will be effectively a custom circuit uniquely designed for that specific task, thus, if well designed, able to perform better than a software running on a generic CPU. The newest FPGA models offer computational power that can even rival those provided by GPUs, with a fraction of energy consumption. The newest model from Altera™ now Intel® , the Stratix® 10, for example, is able to achieve up to 10 Tera Floating-point Operations per Second (TFLOPS) of computing performance, with a peak consumption of 48 W (PARKER, 2014; Altera Corporation, 2015), while the newest NVIDIA® GPU of comparable computational power has a TDP of 300 W (NVIDIA, 2016).

The recent adoption of OpenCL for FPGA development brought a new perspective for algorithm development, introducing a change of paradigm in which the focus is over the algorithm design rather than digital circuit design, allowing a more straightforward transition from code to device. As a side effect, a number of existing code bases can benefit from FPGA acceleration.

With these motivations in mind, this chapter describes an implementation of a framework for classification aimed to primarily help to build significant embedded vision systems, but with enough flexibility to adapt it to any embedded classification need. The acceleration at low power consumption of the FPGA is explored and evaluated, as well as the suitability and characteristics of implementing the system with a new high-level approach by using the respective OpenCL workflow.

5.2 High level system design

The proposed architecture was conceived to use a SIMD-based auxiliary Parallel Processor (PP) associated with a Host Processor (HP), with a high-performance bridge allowing communication between them processors, as well as a double-access global memory module accessible from both processors, using a shared region. Figure 5.1 shows an overview of the main system. In the adopted configuration, the host processor controls the main application execution flow, managing the data access, and dispatching commands to the parallel processor through an interconnection bridge. HP processed data written on the Global Memory (GM)

is distributed by the PP to be processed in parallel by the several processors that are inside the Elementary Processors Arrays (EPAs). Each EPA is further divided in several Elementary Processors (EPs), that are the most refined processing units, which effectively carry on the SIMD processing.

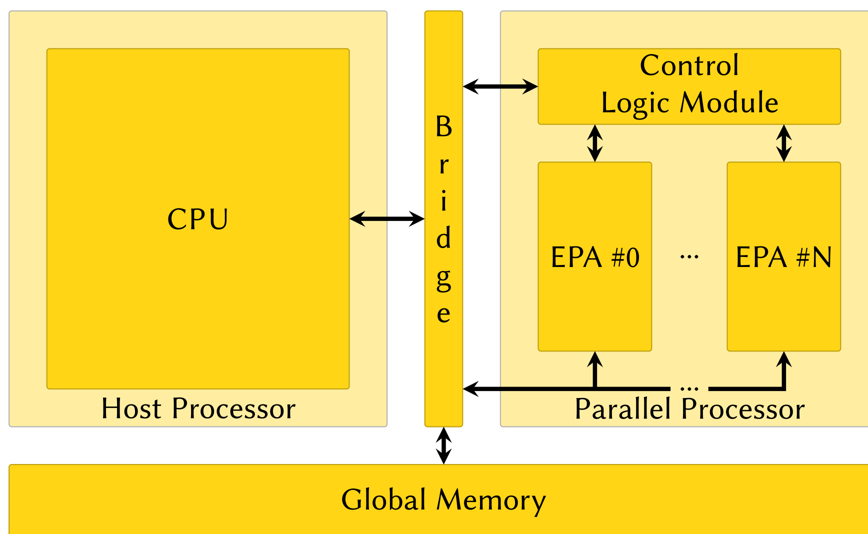


Figure 5.1 – Proposed architecture general overview. The two processing elements communicate through a bridge that also grants memory access. The parallel processor can receive parameters and data directly from the host processor through a bridge DMA channel.

5.2.1 Host Processor

In every parallel application, there are pieces of code that do not require running in specialized parallel hardware. Tasks like data preparation, data flow control, interaction with the user, and commanding the parallel hardware can be performed by a general purpose coordinator processor. In the proposed architecture, this coordinating entity is the HP. It can be based on any kind of all-purpose device. It is usually programmed using a high-level language, like C/C++, interacting with the parallel hardware through dedicated interfaces. This adds flexibility to designing or choosing a hardware platform to implement the architecture. External communication is managed by the HP, whose routines are more readily established using the platform facilities.

5.2.2 Parallel Processor

The auxiliary parallel processor is designed to execute the computationally intensive application tasks. It consists of a Control Logic Module (CLM) and one or more EPAs. Communication with HP is done via the interconnected bridge, which also grants access to GM, which is the main interface for data exchange between the processors. The bridge implements a shared access policy, which contributes to reducing data access latency, and a Direct Memory Access (DMA) channel (not shown in Figure 5.1) grants the parallel processor the ability to read and write data directly from and to GM, making better use of the available bandwidth. The host processor controls the DMA channels, avoiding racing conditions that could corrupt data through an arbiter, which orders access requests to the shared regions and prevents wrong access to reserved areas. This policy also allows the parallel processor to run asynchronously, leaving the host processor free to run other tasks in the meantime.

The CLM provides an interface with the host processor for receiving commands and the memory addresses to read and write back. It also distributes the data among EPAs, controls their execution and coordinates memory access for writing back the results.

Elementary Processors Array

Inside the PP, there is a defined number of units that take data to be processed in parallel, controlling the distribution of this data among the elementary processing units. As already mentioned, they are called Elementary Processors Array or EPA. Figure 5.2 details their configuration. Each EPA is composed of a number of EPs, internal memories, and a control module.

Each EPA has its own Local Shared Memory (LSM) with an associated Memory Management (MM) module. This memory is used for fast data exchanging between EPs. The MM module implements the same protection policy to avoid memory corruption found in the host processor. It is also responsible for coordinating the access to the global memory for EPs. There is also a Local Private Memory (LPM) module for each EP exclusive use. The LPM access is

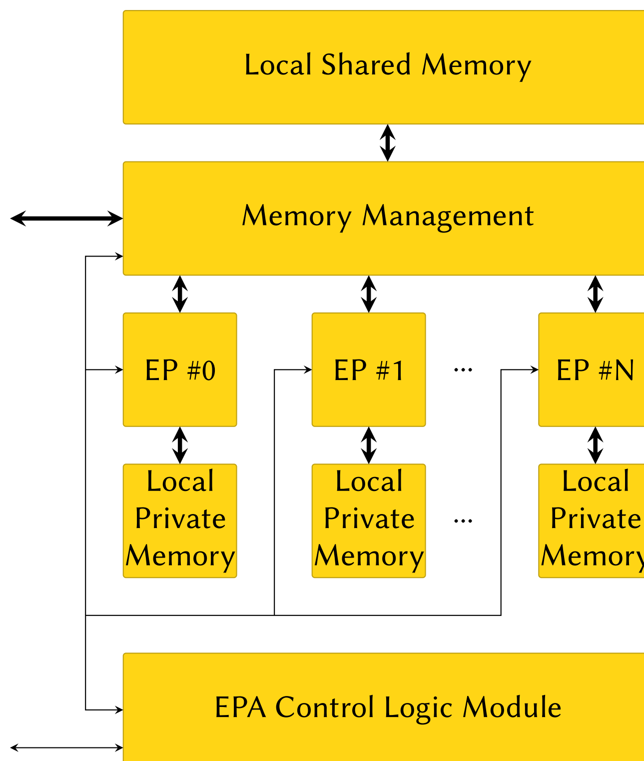


Figure 5.2 – Elementary Processors Array block diagram. Each EPA has a Memory Management module that controls access to external global memory and local shared memory. Each Elementary Processor can access these memories through the MM. Individual local private memory blocks are accessed and controlled by their respective EPs. A Control Logic Module manages the application flow through parameters received by the external controller.

managed by the EP itself, which uses it to store intermediary data.

Elementary Processor

The EPs provide the core functionality of the application. They are responsible to effectively execute the computationally intensive task to be accelerated by parallelization. All EPs are identical, executing the same operation in different blocks of data, complying with a hardwired SIMD architecture. Notice that the proposed architecture corresponds to a SIMD implementation, however the *instruction* here has a more abstract concept, meaning that it represents the full funcionality of the EP, executed concurrently as it is indeed a hardware circuit. Figure 5.3 shows how it is organized.

The EPs were designed to implement the OPF classification algorithm shown in Al-

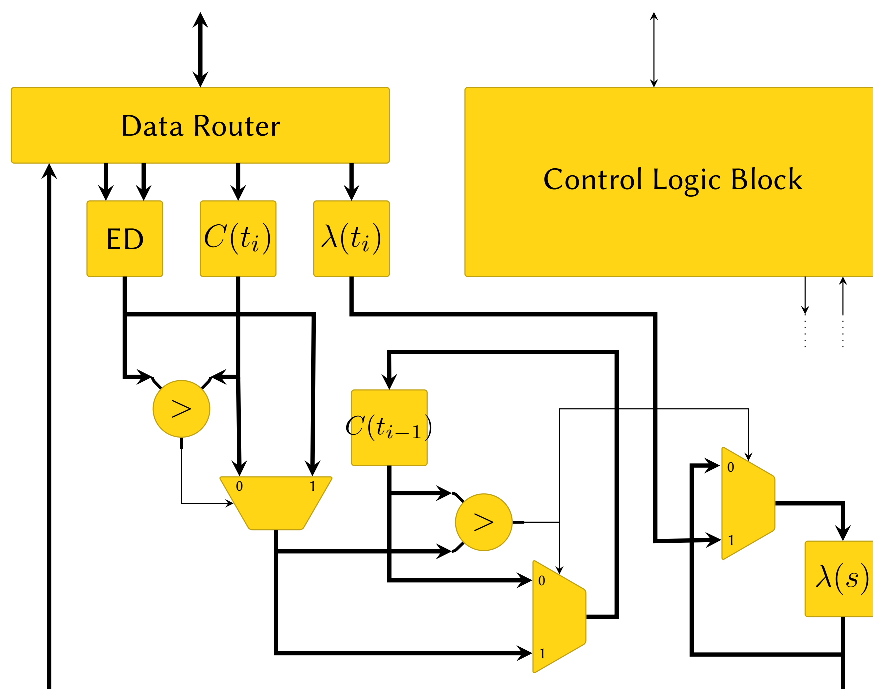


Figure 5.3 – Elementary Processor block showing the hardwired OPF algorithm. The control block receives the parameters from the external controller to command the data flow between the processing components. The Euclidean Distance (ED) block perform the calculation using floating point hardware in the FPGA device. The comparisons are them processed in combinational logic, with the controller generating a synchronization signal to update the registers and then writing the result back to the previously assigned memory address.

gorithm 2.2. The dissimilarity function is the most computationally expensive step in the algorithm, therefore a specific hardware module implements the respective process. Alongside the natural speed gain by implementing the function in a dedicated hardware, the architecture explores parallelism, enabling several data chunks to be distributed to the several EPs in each EPA to be processed simultaneously. Additionally, the parallel architecture was designed to be flexible enough to be adapted to different algorithms simply by redesigning the EP, as long as they comply with the fine-grain parallelism model adopted in this framework.

5.3 System realization

5.3.1 Host Processor code organization

This section details the HP design transcription for implementation on the board. The ARM processor at the SoC board represents the HP and as such it executes the host code, which is written in C/C++. The code is organized into sub-tasks: Input file reading, data preparation, buffer preparation, kernel configuration, kernel launching, presentation of results and resources deallocation. Figure 5.4 presents the execution flow of each sub-task, which is explained in the following:

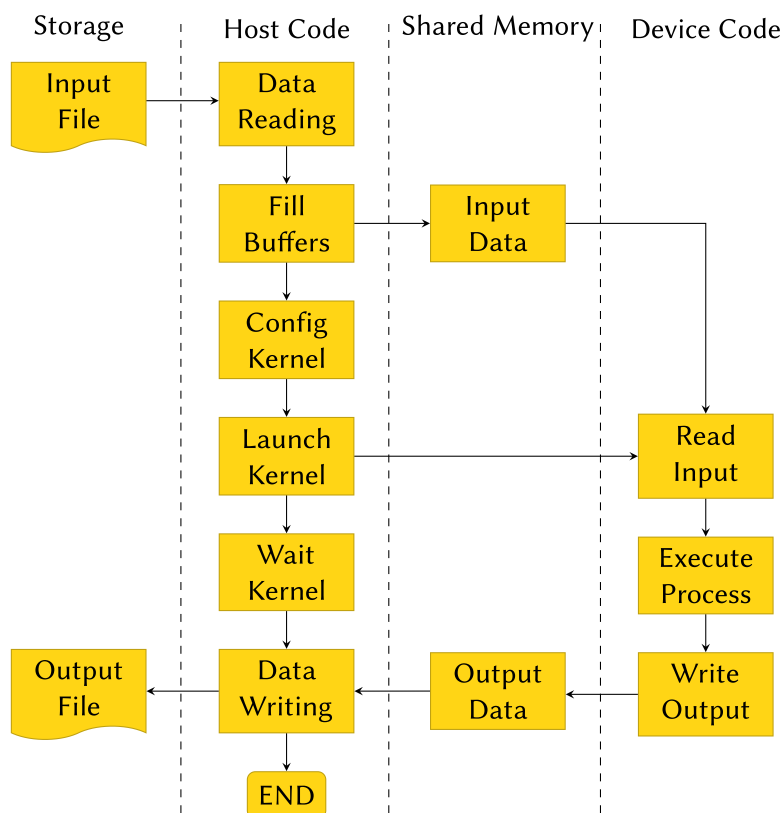


Figure 5.4 – Host code sub-task execution flow, showing the interactions between the processing elements through the shared memory space.

Input file reading: The data sets are organized as two input files, one containing the classifier itself and another with the test data to be classified. Both the files are in the OPF binary format provided by the library.

Buffer preparation: The OpenCL API uses a specific data structure as buffers to communicate

data between Host and Device memory spaces. Therefore, it is necessary to prepare these buffers before moving data around them. As the adopted board uses a shared memory space between the ARM processor and the FPGA fabric, there is no need to make an explicit call to write and read functions. The shared memory control was implemented using an internal controller present in the hard processor system of the chosen device, that takes care of protecting the memory regions to avoid data corruption. Once the buffers are defined, both the Host and Device can access them. The host writes the input data into the corresponding buffers and the device will be responsible for writing its processing results into the output buffers.

Kernel configuration: Once all the buffers are correctly set, the kernel interface is read and configured to run. During the compilation process, the kernel code is stored into a binary file that holds the image to be configured into the FPGA fabric and its interface description. Each buffer is associated with its corresponding argument in the kernel interface. These steps prepare the kernel to execute.

Kernel launching: At this point, the execution is transferred from the host code running on the ARM processor to the device synthesized in the FPGA fabric. The FPGA execution is asynchronous, that is, the host code will continue to run independently of the parallel hardware. It is possible, in the case of very complex parallel code, that the host finishes its execution before the kernel finishes. The API provides barrier function calls to prevent this behavior. Once the kernel completes its execution, the results are written into the output buffers and are ready to be accessed by the host.

Presentation of results: The host code can finally present the results in the manner the user chooses to do so.

Resources deallocation: Once the application finishes, the buffers must be freed to let the device ready for a new task, if it is the case.

The efficiency of the classifier was evaluated using offline-trained OPF data stored in libOPF format (PAPA et al., 2014). The final system is flexible enough to permit the use of different datasets, with diverse feature vector dimensions and multiple classes. Therefore, it can be adapted for different classification tasks just changing the data acquisition and feature extraction methods to a more suitable one to the application in question.

5.3.2 Parallel Processor code organization

Algorithm 2.2 is implemented as a hardwired SIMD architecture in the EP, as shown in Figure 5.3. The algorithm classifies each sample executing two loops. The outer loop iterates over the set of samples to be classified and the inner loop iterates over the classifier nodes to identify the correspondent tree offering the minimum path-cost to its prototype, taking into account the euclidean distance between the sample and the associated cost of each node of the forest. Each EP corresponds to an OpenCL work-item and the EPAs correspond to work-groups. The EPs hardwired SIMD code is implemented in an OpenCL kernel. Following these directives, the kernel is organized such that each EP/work-item loads one sample from the test set and perform the inner loop over the classifier nodes. Notice that the number of samples to be classified in parallel is equal to the total number of EPs considering all EPAs. The currently available compiler does not support more than one kernel instance, nor to call a kernel inside another kernel, thus restricting the inner loop to run sequentially inside each EP/work-item.

Figure 5.5 shows how the kernel is organized. The classifier data is shared among the EPs/work-items that belong to the same EPA/work-group, while the input data is divided among all EPs/work-items. The most computationally expensive operation in the OPF classification algorithm is the Euclidean distance calculation. The compiler optimization capabilities are able to build the internal configuration of EPs/work-items as a pipelined structure, finely tuned with the memory access timing, which contributes to increasing the system throughput by a better employment of the memory bandwidth.

The resulting architecture was synthesized containing two groups of computing units, corresponding to EPAs, which will run in parallel but sharing the memory interface. Because of this sharing, the memory bandwidth is also divided between the units. The number of EPAs to synthesize is an important design decision because it may cause a throughput drop due to serialized memory access; even if there are enough FPGA resources to synthesize several EPAs, doing so may not be the optimal solution. In this work, two units configuration showed to not impact the memory bandwidth significantly and it also is the maximum number possible to synthesize, given the available resources on the adopted FPGA model, so it was done this way. Each EPA was configured to have a maximum of 512 SIMD units/EPs, but the actual number depends on the available resources of the FPGA platform.

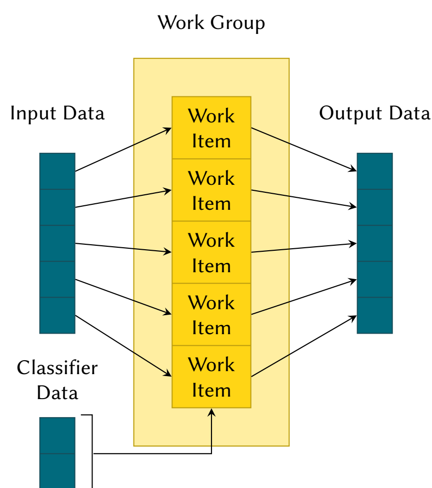


Figure 5.5 – Data distribution for execution by the OpenCL kernel converted in parallel hardware. Input elements are distributed among the EPs while classifier data is shared for all. Then, an output element is written on the memory after processing. Each EPA processes a number of elements equal to the number of EPs it possesses at same time. Then it cycles to the next batch until finished.

5.3.3 Hardware platform specifications

In the last years, FPGA makers have introduced SoC/FPGA devices. They offer in a single encapsulation, a discrete embedded processor associated to the FPGA fabric and dedicated Input/Output (IO) banks for communication between them. This devices introduce a new layer of flexibility to implement efficient and powerful embedded systems, exploring FPGA acceleration capabilities. SoC/FPGA systems like these comply exactly with the proposed HP/PP framework, therefore, adopting a SoC/FPGA-equipped board conducts to a natural function distribution between the embedded microprocessor and the FPGA chip.

Keeping these characteristics in mind led to adopting the Arrow SoCKit, shown in Figure5.6, as the development platform. It is built around a Cyclone V SoC/FPGA chip featuring, in a single encapsulation, a dual-core A9-Cortex ARM processor, an FPGA device and an internal high-speed bridge connecting the processor to the FPGA fabric, as well as a second low-speed bridge to complement the first. The ARM processor enables the implementation of stand-alone embedded systems, eliminating the need for an external host computer.

Figure 5.6 – The Arrow SoCKit development board features a Cyclone® V SoC with a 117k elements FPGA and a dual core ARM Cortex-A9 microprocessor. SoCs have the advantage of not requiring an aditional processor chip, simplifying board design.

5.4 Experimental results

5.4.1 Hardware and software specifications

The development board host processor was set up to run at 800 MHz. The operational system used was the image provided by the manufacturer, consisting of a Linux distribution based on the Yocto Project. The distribution was installed to the board with the addition of the OpenCL runtimes libraries that expose the FPGA side interfaces to the Linux Hardware Abstraction Layer. A micro-SD card hosts the operational system and acts as mass storage device.

The offline training of the different datasets used a PC equipped with an Intel® Core™ 2 Quad Q8400 CPU 2 GHz with 8GB DDR2 RAM memory running Ubuntu 14.04 Trusty Tahr operating system. The resulting classifiers were saved in a file and then transferred to the development board micro-SD card.

The comparison was made with the software-only classification running in the ARM-based processor and its accelerated implementation on the FPGA. The OPF implementation came from version 2.1 of libOPF.

5.4.2 Metrics and performance indicators

The comparison focus is the processing speed gain obtained by measuring the acceleration provided by the FPGA parallel hardware against its software-only counterpart. The execution times for classifying the whole dataset was measured and the average duration to classify an individual sample was calculated.

The chosen performance metric for evaluating the quality of the classification was the Average Accuracy. As the classifier function used for the software and hardware versions are the same, there is no reason to perform a complete qualitative analysis of the results. This metric gives us a general idea of the classification quality and can be used to assess the divergence (if any) in the results of the two versions caused by design decisions.

As defined in (SOKOLOVA; LAPALME, 2009), the Average Accuracy measures the average per class effectiveness of a multi-class classifier and is calculated as:

$$Acc = \frac{\sum_{i=1}^{l} \frac{tp_i + tn_i}{tp_i + fn_i + fp_i + tn_i}}{l}, \tag{5.1}$$

where l is the number of different classes of the problem and tp, fp, tn and fn stand for, respectively, true positives, false positives, true negatives and false negatives for the $i\text{-}th$ sample in the testing set.

5.4.3 Dataset descriptions

For uniformity, the datasets used for the architecture experimental evaluation is the same ones used in Chapter 4, described in Table 4.1, reproduced here for convenience.

Table 5.1 – Dataset descriptions

Id	Name	# attr.	# classes	# samples
D1	Brest Cancer Winsc. (Diag.)	9	2	569
D2	Glass Identification	9	6	214
D3	Image Segmentation	19	7	2310
D4	Iris	4	3	150
D5	Parkinsons	22	2	197
D6	Pedestrian	3780	2	12160

The first five datasets are from the publicly available Machine Learning repository of University of California Irvine (LICHMAN, 2013). The last one is composed of HOG descriptors taken from a compilation of several popular pedestrian detection datasets generated from the work described in Chapter 3 (DINIZ et al., 2015).

The classifiers were generated by Repeated Random Sub-sampling, choosing the best instance of 100 different randomly generated collections of training/evaluation/testing sets. The training set used 40% of the total samples, and the evaluation set 20%. The remaining 40% constitutes the testing set.

5.4.4 Performance analysis

Table 5.2 presents the accuracy observed for every data set running on each version of the classifier. Also, the number of samples in the test set, the total time spent in the classification in milliseconds, the average classification time per sample and the speed-up obtained by using the hardware implementation against its corresponding software version.

It is important to remark that the final PP clock frequency achieved by the synthesis was 101.9 MHz. Even with the HP running at 800 MHz, we can still observe from the results in Table 5.2 that, for the same datasets, the hardware execution times were in average, 2.5 to 10 times faster than the pure software counterpart. This variation is expected, given the nature of the OPF classification algorithm.

When combined with the new SOEL learning, the results are even better, as shown in Table 5.3.

Table 5.2 – Accuracy and classification times for software (S) and hardware (H) versions of the OPF classifier

Id/Version	# of samples	Accuracy	Total time (ms)	Avg. time per sample (ms)	Speed-up
D1/S	276	0.902	5.537	0.020	-
D1/H	276	0.902	0.638	0.002	10.0
D2/S	91	0.882	3.996	0.044	-
D2/H	91	0.882	0.625	0.007	6.285
D3/S	924	0.927	220.817	0.239	-
D3/H	924	0.927	39.408	0.042	5.690
D4/S	60	0.955	0.346	0.005	-
D4/H	60	0.955	0.129	0.002	2.5
D5/S	80	0.812	4.416	0.055	-
D5/H	80	0.812	0.812	0.010	5.5
D6/S	4864	0.801	1,531,090	314.780	-
D6/H	4864	0.801	367,112	75.475	4.170

Table 5.3 – Processing time reduction with combined SOEL+hardware acceleration

Dataset	Test time SW Original (ms)	Test time SOE+HW (ms)	Speed Up
D1	9.917	0.453	21.891
D2	2.050	0.548	3.740
D3	215.867	26.371	8.185
D4	0.570	0.104	5.480
D5	1.831	0.607	3.016
D6	616,433	104,306	5.909

Table 5.4 shows the peak power consumption for the architecture implementation for the chosen FPGA model for the assumed clock configuration. The values are calculated using tools provided by the manufacturer.

Table 5.4 – Final peak power consumption for the implemented architecture. As the FPGA ocupation was near 100%, the peak consumption was near the theoretical maximum of the FPGA model used

Unit	Max Thermal Power (mW)
FPGA	1,663.45
HPS	1,392.92
Total	3,056.37

5.5 Conclusions

This chapter proposed an architecture for embedded systems parallel processing comprising a host processor and a parallel multiprocessor array. Its implementation applied to a classification application algorithm in a SoC/FPGA board using the OpenCL language and workflow is also presented. Adopting OpenCL brings, in general, a shorter time development, considering that it implies the use of higher level abstraction and verified IPs, and consequently less programming error correction effort.

A software version running on the dual-core ARM host processor is used to assess the acceleration provided by the hardware implementation. The comparison shows that the hardware implementation was able to execute 2.5 to 10 times faster than the software version. Also, by combining the parallel hardware architecture with the new training algorithm presented in Chapter 4 we are able to increase the acceleration, making the creation of real-time compliant applications possible.

Published works derived from this chapter:

_____. FPGA accelerated Optimum-Path Forest classifier framework for embedded systems. **Microprocessors and Microsystems**, 2017. Under review.

Chapter 6
General conclusion

"The best way to predict the future is to invent it."
— Alan Kay

6.1 Key contributions

This thesis presented a study of classification applied to embedded systems, aiming performance acceleration using FPGA-based architecture. An example application using a supervised OPF classifier was proposed and the system implemented and tested in a development board. The main contributions can be summarized as follows:

Evaluation of OPF classification applied to pedestrian detection: A new application for OPF supervised classification was implemented and evaluated, comparing its performance with commonly used classifiers applied in pedestrian detection. A dimension reduction technique using PCA, aiming to reduce processing time in classification stage, was also evaluated. OPF showed to be less sensible to the loss of accuracy imposed by PCA, recommending its choice when a fast classification is the main goal. Its accuracy is close to classical methods but still behind them, for this specific scenario.

Proposition of a new training algorithm for OPF supervised classification: A new training algorithm, the Self-Organizing Evolutionary Learning was proposed. The objective is to increase the classification speed by reducing the number of nodes necessary to form the classifiers graph. The proposed method relies on an evolutionary learning approach to adapt a growing self-organizing graph to find a suitable graph representation for the classifier. Two strategies for graph initialization were proposed, one stochastic based and other using an initial guess from the OPF prototypes. The resulting acceleration was of approximately 50%, with a reduction in the number of nodes in the classifier ranging from 42% to 88% in relation

to the classical OPF approaches. The degradation of accuracy was negligible, around 2%, even being higher in some cases.

Proposal and realization of an FPGA-based architecture for classification: An FPGA-based architecture classification performance acceleration for embedded systems, was designed, implemented and tested. The proof of concept was done with OPF-based supervised classifier, using both the classical training algorithm and the new one proposed in this thesis. The architecture proposes a multiprocessor configuration with one main host processor and an auxiliary parallel processor, applying an SIMD strategy to process data in parallel. This configuration takes advantage of fine-grain parallelism which FPGAs excels. The architecture was implemented on a board equipped with an SoC/FPGA. The architecture was able to accelerate the classification 2.5 to 10 times. Further acceleration was obtained combining a classifier generated by the new training algorithm and the hardware-accelerated classifier, reaching 3 to 20 times the original performance.

6.2 Future perspectives

The methods and implementations proposed in this thesis were validated on a thorough experimental framework that allowed to identify several points that can inspire further development. The main ones are described bellow.

Concerning OPF-based classification, future extensions of this work may consider applying a feature selection method alongside PCA, to improve accuracy and false-negative rate. Furthermore, investigating other dimension reductions techniques may also be a valid option to improve the classifier performance for pedestrian detection. As the performance of candidate objects detection in the first phase affects the classification results in the last, a further extension can consider re-evaluating the OPF classifier within a complete system, thus permitting per frame evaluation, a more recent benchmarking methodology for pedestrian detection systems, which also permits the use of standard datasets.

On OPF classifier itself, the new SOEL training algorithm showed promising prospectives, being capable of improving classification times without big compromises in accuracy. One future extension could be to extend it for unsupervised applications as well. To accomplish this

extension, studies of common techniques for clusterization and label propagation are required. The techniques can be derived from the current ones used in the classical OPF unsupervised variation itself and also the ones used from SOMs. On the algorithm itself, current works are applying a smoothing stage at the end of the growing phase, as used in G-SOMs, which can generate better decision boundaries. Further development can also be made on investigating new node adjustment strategies. Finally, the new algorithm introduced new parameters that can influence the classifier performance and so, a strategy for their optimization is necessary. Using K-fold validation may be a solution. Considering the possibility to expand the number of recognized classes using operational data, using an incremental learning approach, an online training associated to classifier parameters update is necessary to be developed. This new incremental method can adopt the learning technique proposed here, which is expected to fit real-time constraints.

Regarding the embedded FPGA architecture, one consideration could be testing the substitution of the floating point operations by fixed point ones, which generally grants performance improvement against a compromise in precision. For many applications, this precision reduction does not represent a significant loss, considering that the achieved acceleration can be very attractive, or even the viable solution for a hard real-time constrained embedded system. Yet, the overall good performance of modern FPGA devices equipped with hard floating-point multipliers is able to meet the requirements of applications in which precision is prevalent over speed. Additionally, new classification methods can be adapted to the architecture, providing a complete framework with flexibility to be applied in a bigger selection of applications. Furthermore, online reconfiguration capabilities presented by some FPGA devices can help to implement the incremental learning method commented before, contributing to meet real-time constraints.

The ensemble of the technologies and techniques presented in this thesis provides an interesting perspective for future applications. As an example, the lower energy consumption in comparison with GPUs, even the embedded ones, might be applied to Unmanned Aerial Vehicles (UAVs) helping them to acquire the needed computational power and, as a consequence, improving time of flight. Autonomous and semi-autonomous vehicles would also profit, in fact, some initiatives are currently in course, as examples presented in this thesis testify. The natural following of this work would be to improve the framework with the addition of new techniques, aiming to develop new significant applications using FPGA-based technology.

Bibliography

AKINDUKO, A. A.; MIRKES, E. M.; GORBAN, A. N. SOM: Stochastic initialization versus principal components. **Information Sciences**, v. 364, p. 213–221, 2016. ISSN 00200255. Disponível em: <doi://10.1016/j.ins.2015.10.013>.

ALAHAKOON, D.; HALGAMUGE, S.; SRINIVASAN, B. A self-growing cluster development approach to data mining. In: **IEEE International Conference on Systems Man and Cybernetics**. IEEE, 1998. v. 3, p. 2901–2906. ISBN 0780347781. ISSN 1062922X. Disponível em: <doi://10.1109/ICSMC.1998.725103>.

ALAHAKOON, D.; HALGAMUGE, S. K.; SRINIVASAN, B. Dynamic Self-Organizing Maps with Controlled Growth for Knoledge Discovery. **IEEE Trans. on Neural Networks**, v. 11, n. 3, p. 601–14, 2000. ISSN 1045-9227. Disponível em: <doi://10.1109/72.846732>.

Altera Corporation. **Leveraging HyperFlex Architecture in Stratix 10 Devices to Achieve Maximum Power Reduction**. [S.l.], 2015. 10 p.

ANANDTECH. **NVIDIA Announces DRIVE PX 2 - Pascal Power For Self-Driving Cars**. 2016.

BRADSKI, G. {The OpenCV Library}. **Dr. Dobb's Journal of Software Tools**, 2000.

CARTER, W. et al. A user programmable reconfigurable gate array. In: **Proceedings of the IEEE in Custom Integrated Circuits**. [S.l.: s.n.], 1986. p. 233–235.

CHANG, C.-C.; LIN, C.-J. {LIBSVM}: A library for support vector machines. **ACM Transactions on Intelligent Systems and Technology**, v. 2, n. 3, p. 27:1—-27:27, 2011.

CHIACHIA, G. et al. Infrared Face Recognition by Optimum-Path Forest. In: **2009 16th International Conference on Systems, Signals and Image Processing**. IEEE, 2009. p. 1–4. ISBN 978-1-4244-4530-1. Disponível em: <doi://10.1109/IWSSIP.2009.5367752>.

CIAMPI, A.; LECHEVALLIER, Y. Clustering Large, Multi-level Data Sets: An Approach Based on Kohonen Self Organizing Maps. In: . Springer Berlin Heidelberg, 2000. p. 353–358. Disponível em: <doi://10.1007/3-540-45372-5_36>.

COLLOBERT, R.; BENGIO, S. Links between perceptrons, MLPs and SVMs. In: **Twenty-first international conference on Machine learning - ICML '04**. New York, New York, USA: ACM Press, 2004. p. 23. ISBN 1581138285. ISSN 1581138385. Disponível em: <doi://10.1145/1015330.1015415>.

CONTE, D. et al. Thirty Years of Graph Matching in Pattern Recognition. **International Journal of Pattern Recognition**, v. 18, n. 3, p. 265–298, 2004. ISSN 0218-0014. Disponível em: <doi://10.1142/S0218001404003228>.

DALAL, N.; TRIGGS, B. Histograms of Oriented Gradients for Human Detection. **2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)**, Ieee, v. 1, p. 886–893, 2005. Disponível em: <doi://10.1109/CVPR.2005.177>.

DINIZ, W. F. S. et al. Evaluation of optimum path forest classifier for pedestrian detection. In: **2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)**. Zhuhai: IEEE, 2015. p. 899–904. ISBN 978-1-4673-9675-2. Disponível em: <doi://10.1109/ROBIO.2015.7418885>.

_____. FPGA accelerated Optimum-Path Forest classifier framework for embedded systems. **Microprocessors and Microsystems**, 2017. Under review.

DOLLAR, P. et al. Pedestrian detection: an evaluation of the state of the art. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 34, n. 4, p. 743–761, apr 2012. ISSN 1939-3539. Disponível em: <doi://10.1109/TPAMI.2011.155>.

FALCÃO, A. X.; STOLFI, J.; De Alencar Lotufo, R. The Image Foresting Transform: Theory, Algorithms, and Applications. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 26, n. 1, p. 19–29, jan 2004. ISSN 01628828. Disponível em: <doi://10.1109/TPAMI.2004.1261076>.

FLYNN, M. Very high-speed computing systems. **Proceedings of the IEEE**, v. 54, n. 12, p. 1901–1909, 1966. ISSN 0018-9219. Disponível em: <doi://10.1109/PROC.1966.5273>.

FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. Intelligent decision making through a simulation of evolution. **Behavioral Science**, John Wiley & Sons, Ltd., v. 11, n. 4, p. 253–272, jul 1966. ISSN 00057940. Disponível em: <doi://10.1002/bs.3830110403>.

FOGGIA, P.; PERCANELLA, G.; VENTO, M. Graph matching and learning in pattern recognition in the last 10 years. **International Journal of Pattern Recognition and Artificial Intelligence**, v. 28, n. 01, p. 1450001, feb 2014. ISSN 0218-0014. Disponível em: <doi://10.1142/S0218001414500013>.

GELAS, J.; HILUY, J.; MOTA, J. **Anales du III Forum BRAFITEC: CAPES/CDEFI - Google Livros**. [S.l.: s.n.], 2007.

GUIDO, R. C. et al. Spoken emotion recognition through optimum-path forest classification using glottal features. **Computer Speech & Language**, v. 24, n. 3, p. 445–460, 2010.

HABINC, S. **Suitability of reprogrammable FPGAs in space applications**. Goteborg, 2002. 44 p.

HENNESSY, J. L.; PATTERSON, D. a. **Computer Architecture, Fourth Edition: A Quantitative Approach**. [s.n.], 2006. 704 p. ISSN 00262692. ISBN 0123704901. Disponível em: <doi://10.1.1.115.1881>.

HILL, K. et al. Comparative analysis of OpenCL vs. HDL with image-processing kernels on Stratix-V FPGA. In: **2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)**. IEEE, 2015. v. 2015-Septe, p. 189–193. ISBN 978-1-4799-1925-3. ISSN 10636862. Disponível em: <doi://10.1109/ASAP.2015.7245733>.

HOLLAND, J. H. **Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence**. 1992. ed. [S.l.]: MIT Press, 1975. 211 p. ISBN 9780262082136.

HOTELLING, H. Analysis of a complex of statistical variables into principal components. **Journal of Educational Psychology**, Warwick & York, v. 24, n. 6, p. 417–441, 1933. ISSN 0022-0663. Disponível em: <doi://10.1037/h0071325>.

HU, Y. H.; PALREDDY, S.; TOMPKINS, W. A patient-adaptable ECG beat classifier using a mixture of experts approach. **IEEE Transactions on Biomedical Engineering**, v. 44, n. 9, p. 891–900, 1997. ISSN 00189294. Disponível em: <doi://10.1109/10.623058>.

HWANG, K.; KAI. **Advanced computer architecture : parallelism, scalability, programmability**. [S.l.]: McGraw-Hill, 1992. 771 p. ISBN 0070316228.

IBM. **IBM - What is big data?** 2014.

KEYSERMANN, M. U.; VARGAS, P. A. Towards Autonomous Robots Via an Incremental Clustering and Associative Learning Architecture. **Cognitive Computation**, v. 7, n. 4, p. 414–433, aug 2015. ISSN 1866-9956. Disponível em: <doi://10.1007/s12559-014-9311-y>.

Khronos Group. **OpenCL Specification**. [S.l.]: Khronos Group, 2009. 1–385 p.

KOBAYASHI, T.; HIDAKA, A.; KURITA, T. Selection of Histograms of Oriented Gradients Features for Pedestrian Detection. In: ISHIKAWA, M. et al. (Ed.). **Neural Information Processing**. 4985. ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, (Lecture Notes in Computer Science, v. 4985). p. 598–607. ISBN 978-3-540-69159-4. Disponível em: <doi://10.1007/978-3-540-69162-4>.

KOHONEN, T. The self-organizing map. **Proceedings of the IEEE**, v. 78, n. 9, p. 1464–1480, 1990. ISSN 0018-9219. Disponível em: <doi://10.1109/5.58325>.

KUNG, H. T.; LEISERSON, C. E. **Systolic Arrays for (VLSI).** [S.l.], 1978. Disponível em: <http://oai.dtic.mil/oai/oai?verb=getRecord{&}metadataPrefix=html{&}identifier=ADA>.

KUON, I.; TESSIER, R.; ROSE, J. FPGA Architecture: Survey and Challenges. **Foundations and Trends® in Electronic Design Automation**, v. 2, n. 2, p. 135–253, 2007. ISSN 1551-3939. Disponível em: <doi://10.1561/1000000005>.

KWIATKOWSKI, J. Evaluation of Parallel Programs by Measurement of Its Granularity. In: **Parallel Processing and Applied Mathematics: 4th International Conference, PPAM 2001 Poland, September 9–12, 2001 Revised Papers**. Springer, Berlin, Heidelberg, 2002. p. 145–153. ISBN 978-3-540-43792-5. Disponível em: <doi://10.1007/3-540-48086-2_16>.

LI, Q.; YAO, C. **Real-Time Concepts for Embedded Systems**. [s.n.], 2003. v. 2003. 218 p. ISSN 10916490. ISBN 4159476015. Disponível em: <doi://10.1073/pnas.1018260108>.

LICHMAN, M. **{UCI} Machine Learning Repository**. 2013.

MILLER, R.; STOUT, Q. F. **Parallel algorithms for regular architectures: Meshes and pyramids**. MIT Press, 1997. v. 33. 134 p. ISSN 08981221. ISBN 9780262132336. Disponível em: <doi://10.1016/S0898-1221(97)90055-9>.

MINNICK, R. C. A Survey of Microcellular Research. **Journal of the ACM**, ACM, v. 5118, n. 2, p. 392–396, 1997. ISSN 00045411. Disponível em: <doi://10.1145/321386.321387>.

NVIDIA. **NVIDIA Tesla P100 Whitepaper**. [S.l.], 2016. 45 p.

PANDA, N.; CHANG, E. Y.; WU, G. Concept boundary detection for speeding up SVMs. **Proceedings of the 23th International Conference on Machine Learning (ICML-06)**, ACM Press, New York, New York, USA, p. 681–688, 2006. Disponível em: <doi://10.1145/1143844.1143930>.

PAPA, J. P. **Classifica{ç}{ã}o Supervisionada de Padr{õ}es Utilizando Floresta de Caminhos {Ó}timos**. 75 p. Tese (Doutorado) — Universidade Estadual de Campinas, 2008.

PAPA, J. P. et al. Optimizing Optimum-Path Forest Classification for Huge Datasets. In: **2010 20th International Conference on Pattern Recognition**. Ieee, 2010. p. 4162–4165. ISBN 978-1-4244-7542-1. ISSN 1051-4651. Disponível em: <doi://10.1109/ICPR.2010.1012>.

_____. Efficient supervised optimum-path forest classification for large datasets. **Pattern Recognition**, v. 45, n. 1, p. 512–520, jan 2012. ISSN 00313203. Disponível em: <doi://10.1016/j.patcog.2011.07.013>.

PAPA, J. P.; FALCÃO, A. X.; SUZUKI, C. T. N. Supervised pattern classification based on optimum-path forest. **International Journal of Imaging Systems and Technology**, v. 19, n. 2, p. 120–131, jun 2009. ISSN 08999457. Disponível em: <doi://10.1002/ima.20188>.

PAPA, J. P.; SUZUKI, C.; FALCÃO, A. X. **LibOPF A library for the design of optimum path forest classifiers**. 2014.

PARKER, M. **June 2014 Altera Corporation Understanding Peak Floating-Point Performance Claims**. [S.l.], 2014. 4 p.

PEARSON, K. On lines and planes of closest fit to systems of points in space. **Philosophical Magazine Series 6**, Taylor & Francis Group, v. 2, n. 11, p. 559–572, nov 1901. Disponível em: <doi://10.1080/14786440109462720>.

PINGREE, P. J. Advancing NASA's On-Board Processing Capabilities with Reconfigurable FPGA Technologies. In: ARIF, T. T. (Ed.). **Aerospace Technologies Advancements**. [S.l.]: InTech, 2010. v. 1, n. January, cap. 5, p. 69–86. ISBN 9789537619961.

PISANI, R. et al. Land use image classification through Optimum-Path Forest Clustering. In: **2011 IEEE International Geoscience and Remote Sensing Symposium**. IEEE, 2011. p. 826–829. ISBN 978-1-4577-1003-2. Disponível em: <doi://10.1109/IGARSS.2011.6049258>.

RECHENBERG, I. **Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution**. 337–337 p. Tese (Doutorado) — Technische Universität Berlin, 1971. Disponível em: <doi://10.1002/fedr.19750860506>.

RIEDMILLER, M.; BRAUN, H. A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In: **Neural Networks, 1993., IEEE International Conference on**. IEEE, 1993. v. 1, p. 586–591. ISBN 0-7803-0999-5. Disponível em: <doi://10.1109/ICNN.1993.298623>.

Safe Car News. **Audi selects Altera for Piloted Driving**. 2015.

SCHWEFEL, H.-P. Numerical Optimization of Computer Models. Birkhäuser Basel, Basel, 1977. Disponível em: <doi://10.1007/978-3-0348-5927-1>.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. **Information Processing & Management**, Elsevier Ltd, v. 45, n. 4, p. 427–437, jul 2009. ISSN 03064573. Disponível em: <doi://10.1016/j.ipm.2009.03.002>.

SPADOTTO, A. A. et al. Oropharyngeal dysphagia identification using wavelets and optimum path forest. In: **2008 3rd International Symposium on Communications, Control and Signal Processing**. IEEE, 2008. p. 735–740. ISBN 978-1-4244-1687-5. Disponível em:

<doi://10.1109/ISCCSP.2008.4537320>.

TANG, B.; MAZZONI, D. Multiclass reduced-set support vector machines. **Proceedings of the 23rd international conference on Machine learning - ICML '06**, ACM Press, New York, New York, USA, p. 921–928, 2006. Disponível em: <doi://10.1145/1143844.1143960>.

TORRES, R.; FALCÃO, A. X.; COSTA, L. Shape description by image foresting transform. **2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)**, v. 2, 2002. Disponível em: <doi://10.1109/ICDSP.2002.1028280>.