


Doctoral Dissertation
Doctoral Program in Electronics and Telecommunications Enginering (34.th cycle)

Machine Learning Algorithms and their Embedded Implementation for Service Robotics Applications

Efficiency and generalization to unlock the power of machine intelligence

Vittorio Mazzia
* * * * * *

Supervisors
Prof. Marcello Chiaberge, Supervisor
Prof. Mario Roberto Casu, Co-supervisor

Doctoral Examination Committee:
Prof. A.B., Referee, University of ...
Prof. C.D., Referee, University of ...
Prof. E.F., University of ...
Prof. G.H., University of ...
Prof. I.J., University of ...

Politecnico di Torino
2021

I hereby declare that, the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Vittorio Mazzia
Turin, 2021

Summary

As the world of the XX century has been reshaped by computers and industrial robotics automation, the XXI century will be progressively transformed by intelligent machines. Among all research fields, the multidisciplinary branch of service robotics works towards a vision of the world where autonomous agents will coexist with humans, assisting them in their daily lives.

Within the last decade, advances in deep learning, coupled with the creation of large available datasets and the boost in computing capabilities, have resulted in remarkable progress in computer perception, natural language processing, machine reasoning, and sensorimotor mapping. Consequently, deep learning (DL) solutions constitute a promising instrument to bring intelligence and endow machines with the necessary level of autonomy to confront our reality. However, most DL models consume vast amounts of power, limiting scalability and applicability to energy-constrained applications. On the contrary, the brain is amazingly efficient, requiring less than 20 W to operate. Moreover, unlike our brain, the generalization of data-driven techniques is limited, and they usually require a well-defined application to maintain their performance. Indeed, domain shifts from training data and affine transformations of target objects constitute an important challenge for current models. All together, represent a real bottleneck for applying deep learning to service robotics which requires generalization and efficiency to live our world.

Within such a context, the presented thesis aims at investigating machine learning algorithms for service robotics, working on a whole data pipeline, from remote sensing data to edge vehicles (e.g., UGV) navigation. It starts proposing DL solutions to extract value from long and short-range remote sensing data levels in support of autonomous agents. Indeed, extracted information from these levels can be a valuable support for navigation and decision-making in practical applications. Subsequently, novel perception and sensorimotor planning solutions for ground vehicles are presented, bringing intelligence at the edge, that is, directly on-board machines. Model accuracy is juxtaposed with power consumption and efficiency at this level, and algorithms build a strict bond with dedicated hardware. However, efficiency is not enough, and generalization always constitutes an essential piece of the problem for service robotics. Consequently, systematic generalization and domain generalization are recurrent themes of the thesis, whether it be bridging simulation to reality gap or overcoming current architecture, training procedure, and data limitations.

Acknowledgements

There are no words to express the immense gratitude I have for all I received during my three years of doctorate. A fantastic path of personal and professional growth that, as with all beautiful things in life, flew fast away to become part of the time. I truly feel extremely lucky and privileged for having had the opportunity to spend three years surrounded by inspiring people full of energy and pure passion. People with the curiosity of a child and the will of true dreamers.

It really felt like being part of a crew onboard a ship, without a specific destination, blown by the wind and ready for the adventure. I felt the complicity of being part of a trusted group, the power of always having someone to rely on, and the awareness of being limited only by our imagination. It has been quite a journey, with beautiful starry nights and days of rough sea, where we felt powerless on our little ship. We lived a pandemic, four removals, one fire, but we were ever better together every time. Our route gave us also many satisfaction and achievements, giving us the sensation of having the potential to make our dent in the world. However, as other crew members before me, it is now my time to get off this ship. It has been an absolute honor to be part of this crew, a crew that has really been a second family to me.

I know that usually, acknowledgments are a long list of influential people for the realization of the presented work. However, being part of two interdepartmental centers, I would have felt the urge to thank too many people. So, I would like to finish this acknowledgments session with the phase that has accompanied all my publications. It is a very formal and cold expression. Still, it encloses in its simplicity the origin of my Ph.D. journey and my gratitude to all ones that have made it possible: "This work has been developed with the contribution of the Politecnico di Torino Interdepartmental Centre for Service Robotics PIC4SeR (https://pic4ser.polito.it) and SmartData@Polito (https://smartdata.polito.it)."

I would like to dedicate this thesis to all that never stop learning.

Life is an endless circle of research for those who want to see the world with the eye of a child.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the world of the XX century has been reshaped by computers and industrial robotics automation, the XXI century will be progressively transformed by intelligent machines. Machines that will manifest a certain level of intelligence and common knowledge of our world, navigating and acting autonomously to solve given tasks. Among all research fields, the multidisciplinary branch of service robotics works towards a vision of the world where autonomous agents will coexist with humans, assisting them in their daily lives. The main focus is to assist human beings, generally performing dull, repetitive, or dangerous tasks, and household chores [230]. Precision agriculture [128], smart cities [138], smart homes [28], underwater [162], and space exploration [19] are only some possible fields of application that are progressively being revolutionized by service robotics, as shown in Figure 1.1. Indeed, contrary to common belief, humanoid robots are only one possible form of service robots: quadrotors (UAV), rovers (UGV), quadrupedal and underwater robots are all examples of embodiment of these future autonomous agents. All machines can work independently or cooperate [251], communicating and extracting value from data to achieve a common goal. Nevertheless, the mechanical part is only the shell of the agent; full autonomy requires effectively perceiving and acting, interacting with the surrounding environment to accomplish goals, and considering the possible long-term consequences of taken actions. Therefore, all that problems that require a certain level of intelligence in taking a right or best choice to achieve a predefined objective.

Within the last decade, advances in deep learning (DL) [141], coupled with the creation of large available datasets, [211], and the boost in computing capabilities [120], have resulted in remarkable progress in computer perception [137], natural language processing (NLP) [246], machine reasoning [215], and sensorimotor mapping [157], that is predicting navigation commands directly from sensor measurements. Consequently, DL solutions constitute a promising instrument to bring intelligence and endow machines with the necessary level of autonomy to face our reality. However, most DL models consume vast amounts of power,

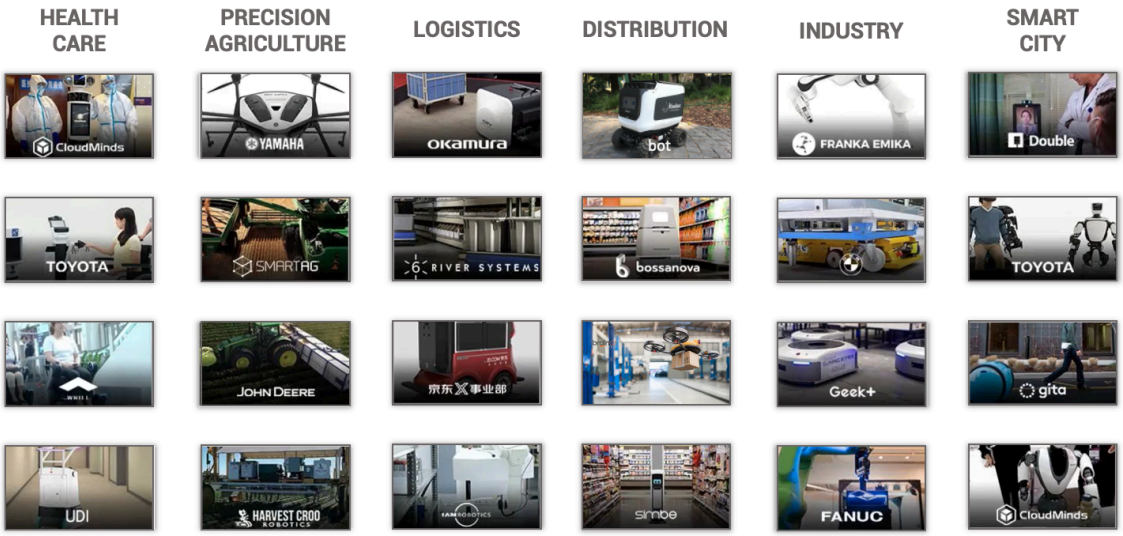


Figure 1.1: Example of already commercialized applications that leverage service robotics in different sectors.

limiting scalability and applicability to energy-constrained applications. Indeed, service robotics has stringent latency, memory usage, storage size, and data protection requirements. So, it is not sufficient to train accurate and precise models to beat benchmarks, but it is necessary to make them energy efficient enough to run onboard machines locally. Architecture $\mathcal{A}$, training procedure $\mathcal{T}$, data $\mathcal{D}$, and optimization/compression methodologies $\mathcal{O}$ have all an essential role in this regard.

Moreover, today, deep neural networks are progressively hitting bottlenecks that limit generalization capabilities. Indeed, although existing artificial intelligence (AI) methods have produced powerful applications that outperform humans in specific bounded domains [103], these techniques have fundamental limitations that hinder the creation of general systems. Since the earliest days of research on neural networks, a recurring point of debate has been whether neural networks exhibit generalization beyond their training experience in a systematic way [98]. The problem of induction has a central role within the learning process. Without generalization, machine learning algorithms would be able to exhibit proper behaviors only in situations identical to the ones previously experienced [241]. However, as a matter of fact, the generalization of data-driven techniques is limited nowadays. Indeed, the specific application usually requires well-defined boundaries and compromises to maintain the required training performance. All together, represent a real barrier for applying deep learning to service robotics which requires systematic generalization and efficiency to live our world. Service robotics applications always face environment variations and unpredictable situations that significantly challenge and affect statistical methods with their generalization capability.

1.1 Research motivation and objectives

Robots dynamics capabilities have drastically evolved in the past few years through performing actuators, providing machines with the necessary agility and dexterity to tackle diverse tasks in challenging and complex situations [157]. Moreover, advancement in sensor research has progressively enabled machines to acquire more rich and diverse perception data. LiDARs [24], RGB-D cameras [22], stereo-cameras [73] are some examples of sensors that have been increasingly becoming common for robotics applications in the last decade. Nevertheless, the development of the required processing algorithms and the supporting hardware is still a long-standing open challenge. It prevents endowing agents with the necessary level of intelligence and autonomy to tackle our variegated world and extrapolate from data information in their support. However, advances in artificial intelligence (AI) solutions, optimization techniques, and hardware accelerators lay the foundations to build truly intelligent machines. Indeed, DL models, supported by optimization methodologies [116] and distillation techniques [99], are not only helpful to analyze and extrapolate information from sensory data, but they are a promising solution to bring intelligence at the edge, that is, directly on-board machines. Edge AI [150] is progressively becoming a growing field of research. Indeed, transfer intelligence on devices is of interest not only for robotics but also for all fields of application that want to bring intelligence and awareness to their products.

Therefore, the presented dissertation aims at investigating DL methodologies for service robotics, applying novel solutions to the entire service robotics data ecosystem: from long and close-range data (e.g., satellites, UAVs, UWB) to the machine/edge level, where power consumption and efficiency cannot be overlooked anymore. Indeed, information extracted from the different data ecosystem levels could greatly benefit service robotics applications, supporting decision-making [129] and navigation of ground vehicles [5]. Computational load and latency are not a matter of much concern at these supporting levels, and the cloud can be used to fully exploit the potential of deep learning algorithms. On the other hand, machines have limited computational capabilities, and energy is a major concern for most applications, preventing the direct application of previous-level algorithms. Thus, powered by hardware accelerators of different natures, novel perception and sensorimotor planning solutions for ground vehicles are presented, bringing intelligence directly on-board machines. Model accuracy is juxtaposed with power consumption and efficiency at the machine level, and algorithms build a strict bond with dedicated hardware. Embedded General-Purpose Graphics Processing Units (GPGPU), Vision Processing Units (VPU), and edge Tensor Processing Units (TPU) are hardware accelerators that can be leveraged to bring intelligence at the edge effectively. Indeed, hardware accelerators in conjunction with optimization techniques can deliver solutions that perform 4 trillion operations per second (TOPS) with barely 2 W. Figure 1.2 presents an example of a full stack of data layers for a

Figure 1.2: Data ecosystem of a precision agriculture application. Long and close-range RS acquired data can be processed by ML algorithms to extrapolate valuable information for the overall process. Subsequently, machines at the edge layer, guided by information of previous levels, are driven by similar ML techniques, optimized to run locally and efficiently onboard the machines for navigation, decision-making, and task execution.

precision agriculture application. Long and close-range remote sensing (RS) data can be processed by ML algorithms to extrapolate valuable information for the overall process. Subsequently, machines at the edge layer, guided by information of previous levels, are driven by similar ML techniques, optimized to run locally on-board the machines themselves. However, efficiency is not enough, and generalization always constitutes an essential piece of the problem for service robotics. Consequently, systematic generalization and domain generalization are recurrent themes of the thesis, whether it be bridging simulation to reality gap or overcoming current deep learning architectures, training procedure, and data limitations. Nowadays, simulation is a valuable tool for data-driven robotics: it accelerates development and enables sensorimotor planning models trained with reinforcement learning (RL) or imitation learning. Nevertheless, a policy learned in simulation could not scale to the real world, and the gap between the two realities has to be taken into serious consideration. Zero-shot generalization, domain randomization [239], self-supervised methodologies [44] and capsule-based networks [212] are only some of the arguments developed and elaborated in the dissertation.

1.2 Main contributions

The main contributions of this thesis can be summarized as follows:

- Propose a vision of service robotics in which similar machine learning and deep learning approaches can be exploited to a different level of data sources, ultimately guiding navigation, decision-making, and tasks of autonomous machines.
- Introduce cutting-edge processing algorithms for long-range and short-range remote sensing data in support of ground vehicles navigation.
- Design and develop methodologies that combine long and short-range remote sensing data to enhance navigation and decision-making of edge agents.
- Propose perception and sensorimotor planning novel solutions that coupled with optimization techniques and hardware accelerators bring intelligence directly on board machines.
- Investigate novel deep learning architecture paradigms for machine perception and out-of-domain techniques, working towards generalization of autonomous agents algorithms.

Part of the work included in this thesis was presented through several contributions in international conferences renowned in the AI, and robotics community [5, 230, 41, 3, 130] and peer-reviewed journal [214, 171, 9, 169, 4, 173, 168, 172, 170, 165, 40]. Some contributions were instead part of the DisloMan Dynamic Integrated ShopfLoor Operation MANagement for Industry 4.0 project [167] and technical reports of the MANUNET European project.

Some research contributions included in this work received the best paper award on Agri-Robotics at the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2021) [5] and Best Presentation Award for the 5th SmartData@Polito Workshop [168] and 6th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS) [41].

1.3 How to read this dissertation

The presented dissertation is divided into three distinct parts. All three sections can be read independently by the rest of the thesis and are all introduced by the necessary theoretical concepts.

Part I presents contributions for the data ecosystem surrounding autonomous agents, focusing on long and close-range remote sensing sources of data. Deep learning methodologies and techniques are exploited to propose state-of-the-art solutions to support edge agents to navigation, decision-making, and task delivery.

Computational load and latency are not a matter of much concern at these levels, and the cloud can be exploited to fully take advantage of the potential of present deep learning algorithms. **Chapter 3** first provides a general background to remote sensing. Common RS platforms and spectral indices are introduced and analyzed. **Chapter 4** and **Chapter 5** discusses contributions that exploit long and close-range RS data, respectively. All selected works brought as examples are united by an hypothetical precision agricultural application. However, similar data ecosystems can be found in most service robotics fields of application.

Part II introduces edge AI contributions in support of edge machines. Novel perception and sensorimotor planning solutions for ground vehicles are presented, bringing intelligence directly on-board machines. Model accuracy is juxtaposed with power consumption and efficiency at the machine level, and algorithms build a strict bond with dedicated hardware. **Chapter 6** discusses neural efficiency theoretical concepts, from model optimization and distillation techniques to hardware accelerators. **Chapter 7** provides a series of contributions that leverage on edge AI to provide robust navigation, perception, and planning.

Part III discusses systematic and out-of-domain generalization, introducing novel architectural paradigms, training procedures, and data acquisition techniques. **Chapter 8** provides some concepts to easily follow works proposed in **Chapter 9**. It gives background and concepts regarding single domain and out-of-domain generalization.

Finally, **Chapter 2** first provides a general introduction to deep learning for service robotics. Architectures, supervised, self-supervised training procedures, and state-of-the-art optimization techniques are presented in detail and thoroughly discussed. All introduced concepts of chapter 2 are helpful for all three parts of the dissertation.

The notation used throughout this dissertation is the same of [85], if not otherwise stated. All vectors are column vectors. Finally, to provide more continuity to all links shared with this dissertation, an updated link list can be found at https://vittoriomazzia.com/phdthesislinks.

Chapter 2

Deep Learning for Service Robotics

Chapter two aims at providing the minimal theoretical ML and DL background to understand the presented dissertation. Therefore, only concepts recalled by subsequent chapters are introduced, and each topic is presented with a view of the final application inside the thesis. It is a general theoretical chapter that presents notions common to all three parts of the thesis and useful to build intelligent machines.

The first section introduces deep learning and the computational model at the heart of each artificial neural network (ANN). The following section describes the common models at the foundation of some of the contributions presented in the dissertation. Subsequently, the chapter concludes by introducing essential aspects of training algorithms. Firstly, it describes two families of learning algorithms: supervised and self-supervised. Then, it presents some optimization techniques for ANNs and standard regularization and initialization methodologies to help minimize the highly non-convex loss functions of deep neural networks. On the other hand, unsupervised learning algorithms adopted in the text are not discussed, but we refer an interested reader to [80].

2.1 Going deeper with machine learning

Artificial neural networks were first introduced in 1943 by Warren McCulloch and the mathematician Walter Pitts in their landmark paper "A logical Calculus of the Ideas Immanent in Nervous Activity" [174]. They proposed the first simplified computational model of a biological neuron: a computational unit that takes as input one or more binary signals and produces one binary output. They initiated a field of research that goes under the broader machine learning field of study, which is part of artificial intelligence. AI aims at creating intelligent systems, that is,

agents with the capability to make the right choices in different situations. Different approaches have been investigated during the years, but empirical evidence point to the process of learning as an indispensable feature to achieve intelligent behaviors. Indeed, machine learning is the prominent branch of artificial intelligence that studies how to program machines to learn from experience to make the right choices. The representation of data plays an essential role in ML algorithms, and feature engineering was an important part of the work of AI scientists. Although ANNs, as ML models, had the desirable characteristics to create their own representations, they went through periods of hype and decline. Indeed, after McCulloch and Pitts's publication, ANN underwent a period called "Cybernetics," which brought several contributions to the field as Perceptron [209] from Frank Rosenblatt. Successively, after a period of decay during the 70', ANN came back for a new decade dubbed "Connectionism" pushed by cognitive science inspiration. Backpropagation [210] and multi-layer perceptrons with sigmoid as activation function came from that period. Nevertheless, technology was not ready, and ANNs had to wait until the beginning of the first decade of the XXI century. The advent of powerful and versatile GP-GPU, the increasing availability of a massive quantity of data, and the introduction of better and more efficient training algorithms have risen ANNs again with a third wave that goes under the name "Deep Learning." Excellent results based on ANNs regularly make the headline news, attracting more researchers and organizations and attracting more funding and interest.

Deep learning places its foundations on works of the XX century, leveraging the versatility, powerfulness, and scalability of ANNs as ML models. It focuses its main efforts to optimize networks with several layers placed one on top of each other. That makes it possible to exploit the hierarchy of data that generally characterizes our world. Indeed, deep learning models have the possibility to decompose a problem into simple pieces and increasingly construct on them to create abstract and disentangle representations. They found the biological inspiration of McCulloch and Pitts, but similarities with the functioning of our brain already stop at the modeling of neurons. Indeed, deep learning models employ a slight variation of the threshold logic unit (TLU) proposed by Rosenblatt [209] which in turn takes inspiration from the artificial neuron proposed by McCulloch and Pitts. The main idea is to perform a weighted sum of inputs coming to a unit and successively apply a non-linear activation function that determines the grade of activation of the unit itself. Unlike TLU, the activation function is not a simple step function in order to enable gradient-based learning. The resulting unit is a simplified computational model of how biological neurons work.

Figure 2.1 shows a common representation used to compare biological and artificial neurons. It shows a multipolar neuron called pyramidal cell that is the most common excitatory neuron in the neocortex (the neocortex is the part of the brain responsible for all high-level cognitive ability shown by human beings). It is always suggested that the two systems have very similar working principles in

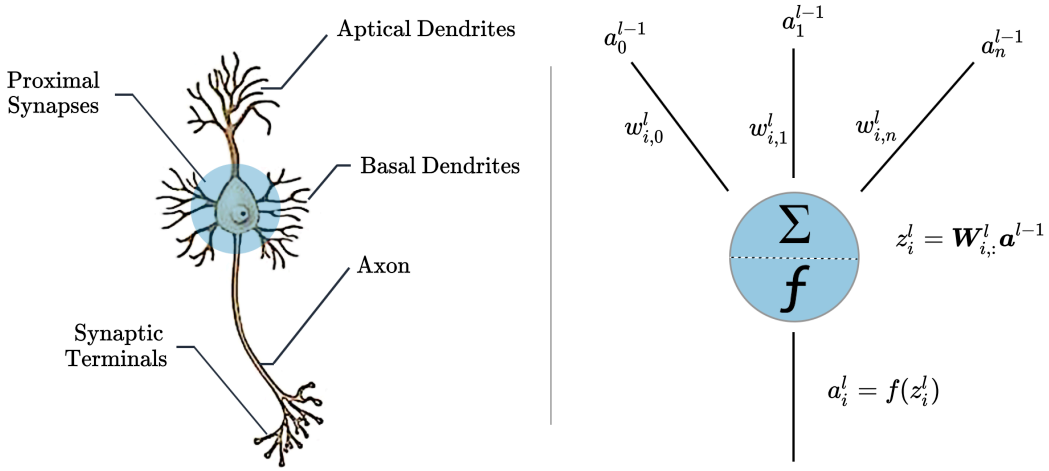


Figure 2.1: Schematic comparison between a biological and artificial neuron. On the left is depicted a pyramidal neuron, the most common excitatory neuron in the neocortex. In light blue is highlighted the area broadly modeled by an artificial neuron. Synapses, modeled as weights $W_{i,j} = w_{i,j}$, perform a weighted sum of input signals and in conjunction with an activation function f is determined the grade of activation of the neuron itself. Therefore, an artificial unit can recognize a pattern of activity on its synapses. On the other hand, neurosciences studies suggest that a single biological neuron can recognize hundreds of different patterns alone.

books. Nevertheless, that is far from being true: only 10% of synapses (comparable to weights $W_{i,j} = w_{i,j}$ of ANNs) are able to generate an action potential (AP) and make the neuron fires. Approximately only synapses located in the region of the proximal synapse, highlighted in light blue in Figure 2.1. On the other hand, our computational model does not model basal zones and aptical dendrites that feature the majority of synapses. Indeed, in deep learning models, it is common to think of a neuron as recognizing a single pattern of activity on its synapses. On the other hand, neurosciences studies suggest that a single biological neuron can recognize hundreds of different patterns alone. That clearly shows how deep learning is only very loosely inspired by cognitive science and neurosciences, placing its foundation on a computational unit that is very distant from its biological counterpart. Nevertheless, the research community does not agree on what should be the degree of inspiration. The majority of research groups work without anymore considering the biological analogy. Instead, other groups try to find inspiration from the biological example to solve some crucial drawbacks of current deep learning models: catastrophic forgetting, memory/computational inefficiency, and lack of broader generalization.

However, regardless of the simplicity of the computational model at the heart of each neural network, highly complex computations can be performed by a network of a reduced number of neurons. Moreover, the universal approximation theorem [104] proves that ANNs are theoretically able to approximate any arbitrarily complex function. Nevertheless, learning algorithms could not find the solution searched. Therefore, many efforts are made to increasingly develop better architectures and learning procedures to solve more challenging and complex problems.

2.2 Common neural network architectures

This section describes some of the common architectures for DL that are at the foundation of the contributions presented in the dissertation.

2.2.1 Multilayer Perceptron

The Multilayer Perceptron (MLP) is a direct descendant of the perceptron proposed by Rosenblatt [209] in 1957. It is the architecture introduced in 1986 by

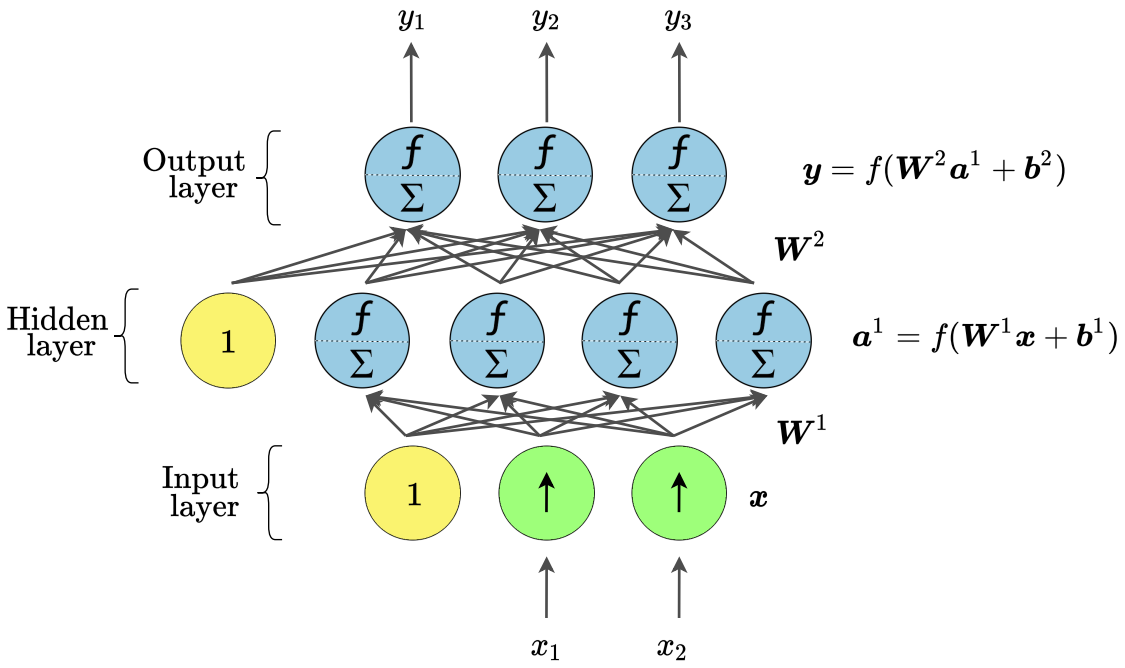


Figure 2.2: The architecture of an MLP with two inputs, one hidden layer of four neurons, and three output neurons. The network is Fully Connected (FC), and signals flow only in one direction (from the inputs to the outputs). Thus, this is an example of a Feedforward Neural Network (FNN).

David Rumelhart, Geoffrey Hinton, and Ronald Williams [210] that stacks a perceptron layer on top of each other and implements a differentiable activation function in order to enable gradient-based learning. Indeed, they replaced the step function with the logistic (sigmoid) function, $\sigma(z) = 1/(1 + \exp(-z)$, because it has a nonzero derivative everywhere, allowing gradient descent to make progress at every step. Several other alternative activation functions have been proposed, each with advantages and disadvantages. Tanh, $\tanh(z) = 2\sigma(2z) - 1$, ReLU, $\text{ReLU}(z) = \max(0, z)$ [181], LeakyReLU [258], $\text{LeakyReLU}_\alpha(z) = \max(\alpha z, z)$, PReLU [94], softplus, $\text{softplus}(z) = \log(1 + \text{ext}(z))$, ELU [50], SELU [134], GELU [97] and Mish [175] are only some of the activations proposed in the literature.

Regardless the specific activation function selected, each unit inside a MLP performs the operation summarized in Figure 2.1. Using the superscript to indicate a layer l and the subscript to indicate a specific neuron i, the weighted sum can be writen as $z_i^l = \boldsymbol{W}_{i,:}^l \boldsymbol{a}^{l-1}$, where $\boldsymbol{W}_{i,:}^{l-1}$ is the array of weights of the i-th neuron. Subsequently, the activation function f is applied in order to produce the new output of the neuron $a_i^l = f(z_i^l)$. On the other hand, a composition of elementary units in layers forms the architecture depicted in Figure 2.2. It is a simple MLP with two inputs, one hidden layer of four neurons, and three output neurons. Cells with "1" are bias neurons, which output one all the time.

It is possible to collect all weights of a layer in a matrix $\boldsymbol{W}$, with the number of rows and columns equal to the number of output and input neurons, respectively. Therefore, it is possible to compute the output of a layer l as

$$\boldsymbol{a}^l = f(\boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l) \tag{2.1}$$

where $\boldsymbol{b}$ is the bias vector containing all weights between the bias neuron and the artificial neurons. On the other hand, $\boldsymbol{a}^{l-1}$ is the output vector of the previous layer. Finally, the output of the network $\boldsymbol{y}$ is computed by equation (2.1) in the same way to other hidden layers.

2.2.2 Convolutional neural network

A multilayer perceptron is a versatile and scalable architecture, but it does not make any assumption of the data it has to model. However, many real-world cases present data in which proximity of features has an important role. For instance, if pixels of an image are treated as features, their location is not irrelevant but carries essential information about the intrinsic structure of data. Therefore, architecture with a prior of the low-level locality of data would facilitate working with such data.

That is what is achieved by Convolutional Neural Networks (CNNs). The main idea is to exploit the hierarchical nature of data as MLP and keep the spatial information of the input image and process it with a network that exploits the image structure. In a CNN, an input image does not to be flattened into a vector,

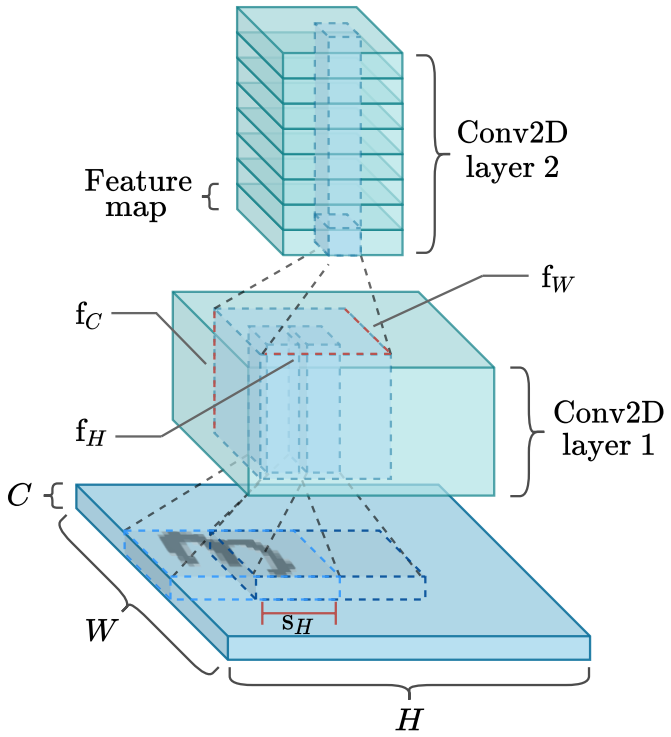


Figure 2.3: Convolutional layers with multiple feature maps. The first and second layer has a stride s_W and s_H greater than one. Therefore, the two spatial dimensions W and H are progressively reduced. On the other hand, the number of channel is increased at each layer in order to capture more high-level features.

but it is a tensor $\mathbf{X}$ with three dimensions $H \times W \times C$, where H, W and C are the height, width and number of channels (e.g., RGB), respectively. Subsequently, an input tensor is processed by a stack of convolutional layers, which is the fundamental building block inside a CNN. Similar to MLP neurons, units perform a weighted sum in a convolutional layer without being fully connected. Neurons are connected only to units that falls in their receptive fields $f_H \times f_W \times f_C$, where f_H, f_W and f_C are the height, width and channels depth of the receptive field. Usually, a receptive neuron field extends across all the previous layer channels with f_C equal to the number of channels of the previous layer. Therefore, a neuron placed in row i, column j of a given layer is connected to the outputs of the neurons in the previous layer located in rows i to $i + f_H - 1$, columns j to $j + f_W - 1$. The term channel is only used for the input tensor. On the contrary, the collection of all neurons in a layer is called a feature map.

It is common to connect a large feature map of a layer to a much smaller one

in the following layer by spacing out receptive fields. The shift from one receptive field to the next is called the stride. Therefore, a neuron located in row i, column j in the upper layer is connected to the outputs of the neurons in the previous layer located in rows $i \times \mathrm{s}_H$ to $i \times \mathrm{s}_H + \mathrm{f}_H - 1$, columns $j \times \mathrm{s}_W$ to $j \times \mathrm{s}_W + \mathrm{f}_W - 1$, where s_H and s_W are the vertical and horizontal strides.

A layer is usually composed of multiple feature maps. All neurons within a given feature map share the same parameters (i.e., the same weights and bias term). On the other hand, units in different feature maps use different parameters. Ultimately, a neuron located in row i, column j of a feature map k in a given convolutional layer l is connected to the outputs of the neurons in the previous layer $l-1$, located in rows $i \times \mathrm{s}_H$ to $i \times \mathrm{s}_H + \mathrm{f}_H - 1$ and columns $j \times \mathrm{s}_W$ to $j \times \mathrm{s}_W + \mathrm{f}_W - 1$, across all feature maps. Therefore, it is possible to summarize the weighted sum performed by a neuron in a convolutional layer with the following formula

$$z_{i,j,k} = b_k + \sum_{u=0}^{\mathrm{f}_H - 1} \sum_{v=0}^{\mathrm{f}_W - 1} \sum_{k'=0}^{\mathrm{f}_{n'} - 1} x_{i',j',k'} \cdot w_{u,v,k',k} \tag{2.2}$$

where $i' = i \times \mathrm{s}_H + u$, $j' = j \times \mathrm{s}_W + v$ and $X_{i',j',k'} = x_{i',j',k'}$.

Figure 2.3 summarizes all concepts discussed in this section so far. Moreover, it highlights the hierarchical nature of CNN. Indeed, a common CNN architecture progressively reduces the spatial dimension of an input tensor (W and H) while increasing the number of channels (feature maps). That is useful to detect more high-level and abstract features, building over low-level extracted features. So, a CNN usually starts extracting color gradients, corners, edges of a scene and progressively builds upon them to detect objects and complex scene structures. Finally, extracted representations can be processed by a further machine learning model that can be jointly trained with the convolutional counterpart. In this way, it is possible to perform different tasks requiring a high-level representation of the signal analyzed: image classification, semantic or instance segmentation, and object detection.

Convolutional neural networks presented so far perform a 2D convolutional operation and are mostly adopted for visual understanding. Indeed, they are much more accurate and efficient than MLP for image processing. Their priors allow them to exploit locality successfully and efficiently detect objects in every part of the scene (translation equivariance). For their success in image analysis, researchers have tried to apply CNNs to different types of signals, adapting convolutions to different input dimensions. For instance, 1D convolutions are very successful with waves [9], and 3D convolutions can be used to process points cloud or temporal video sequences.

Due to their success, after their first practical usage for recognition of digits [143] (LeNet), and their raising in 2012 with the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [137], many works on the literature have contributed at

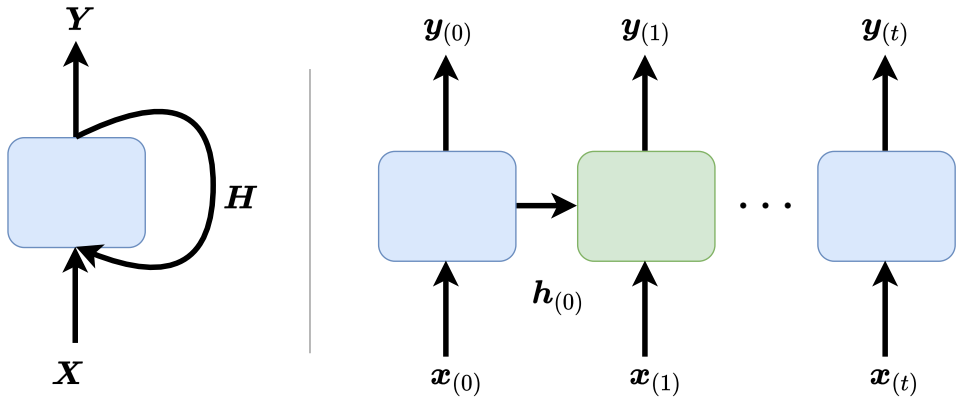


Figure 2.4: A recurrent layer and its unrolled through time representation. A temporal sample $\boldsymbol{X}$ is made by a sequence of time steps, $\boldsymbol{x}_{(t)}$, that along the previous output $\boldsymbol{h}_{(t)}$ feed the next iteration of the network. Either $\boldsymbol{x}$ and $\boldsymbol{h}$ are vectors which bring representations of the sample at the instance t.

creating better and more elaborated architecture using the convolution operation as a building block. Except for the different pooling layers to reduce spatial dimension already introduced with the first LeNet architecture, global average pooling [152], inception blocks [231], residual connections [93], attention blocks [106], 1×1 convolutions [152] and depth-wise convolutions [47] are only some of the operations proposed during the years. Each of those operations will be presented once adopted as a building block in the different architectures presented in the following chapters of the dissertation.

2.2.3 Long-short term memory network

As previously stated, 1D CNNs have also been applied to temporal sequences, showing good results and generalization capabilities. Especially for short temporal sequences. However, the most classical architectures to analyze time-series data are Recurrent Neural Networks (RNNs). Figure 2.4 presents a simple RNN model, composed of just one layer, where all neurons are in the depth dimension. It looks very similar to the feedforward neural shown in Figure 2.2, rotating the network around the vertical axis. However, there are also connections going backward. Indeed, the layer is not only fed by an input vector $\boldsymbol{x}_{(t)}$, but it also receives $\boldsymbol{h}_{(t)}$ (cell state), which is equal to the output neuron itself, $\boldsymbol{y}_{(t)}$. So, at each time step t, this recurrent layer receives an input 1-D array $\boldsymbol{x}_{(t)}$ as well as its own output from the previous time step, $\boldsymbol{y}_{(t-1)}$. In general, since the output of a recurrent neuron at time step t is a function of all inputs from previous time steps, it has, intuitively, a sort of memory that influences all successive outputs. Therefore, considering the

similarity with an MLP, it is straightforward to compute a cell output for a certain time step t, as

$$\boldsymbol{y}_{(t)} = \phi(\boldsymbol{W}_x \cdot \boldsymbol{x}_{(t)} + \boldsymbol{W}_y \cdot \boldsymbol{y}_{(t-1)} + \boldsymbol{b}) \tag{2.3}$$

where, $\boldsymbol{W}_x$, $\boldsymbol{W}_x \boldsymbol{W}_y$ and $\boldsymbol{b}$ are the two weight matrices and the bias array, respectively. For the sake of clarity, it is important to highlight that $\boldsymbol{y}_{(t)}$ as $\boldsymbol{x}_{(t)}$ are vectors and they can have an arbitrary number of elements, but the representation Figure 2.2 does not change. Simply, all neurons are hidden in the depth dimension.

Unfortunately, simply adapting the basic cell of Figure 2.1 with a backward connection results in a significant instability issue with the training and the gradient flow. Consequently, other types of cells have been proposed in the literature with the shared aim to contrast vanishing or exploding gradient issues and construct more resilient and stable recurrent-based architectures. Most notably, the Long Short-Term Memory (LSTM) cell proposed by Sepp Hochreiter and Jurgen Schmidhuber in 2000 [81], Gated Recurrent Unit (GRU) introduced by Cho, et al. [45] in 2014 and their variations and improvements.

For instance, Figure 2.5 shows a peephole LSTM that is an improved variation of original LSTM. However, the main concept remains the same: it provides the network with a mechanism to easily forget and retain information. Therefore, the network can learn what to store in a long-term state, $\boldsymbol{c}_{(t)}$ what to throw away and

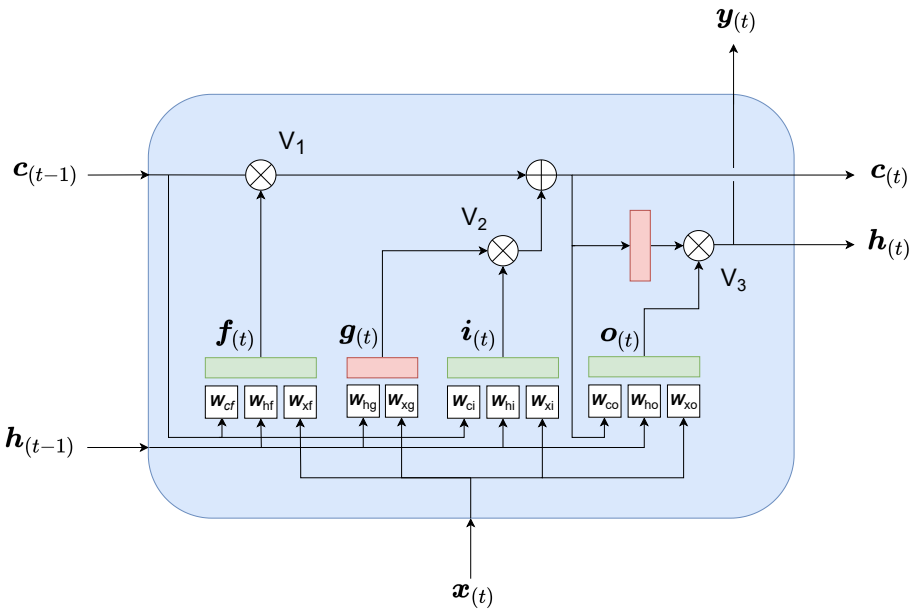


Figure 2.5: LSTM cell with peephole connections. A time step t an input $\boldsymbol{x}_{(t)}$ is processed by the memory cell which decides what to add and forgot in the long-term state $\boldsymbol{c}_{(t)}$ and what discard for the present state $\boldsymbol{y}_t$.

what to use for the current state $\boldsymbol{h}_{(t)}$ and $\boldsymbol{y}_{(t)}$ that, as for the basic unit, are equal. That is performed with simple element-wise multiplications working as "valves" for the fluxes of information. Those elements, V_1, V_2 and V_3 are controlled by fully connected layers that have as input the current input state $\boldsymbol{x}_{(t)}$ and the previous short-term memory term $\boldsymbol{h}_{(t-1)}$. In addition, in the peephole LSTM cell, the long-term state $\boldsymbol{c}_{(t-1)}$ is added as an input to the FC of the forgot gate,V_1, and the input gate, V_2. Finally, the current long-term state $\boldsymbol{c}_t$ is added as an input to the FC of the output gate. All "gates controllers" have sigmoid as activation functions (green boxes. Instead, tanh activations to process directly the signals (red boxes). In conclusion, a LSTM block has three input/output signals; two are the standard input state $\boldsymbol{x}_{(t)}$ and cell output $\boldsymbol{y}_{(t)}$. Instead, $\boldsymbol{c}$ and $\boldsymbol{h}$ are the long and short-term state, respectively. The unit has the capability to utilize its internal controllers and valves to add and discard information. Formally, equations in (2.4) summarize how to compute the cell long-term state, its short-term state, and its output at each time step for a single instance.

$$
\begin{aligned}
\boldsymbol{i}_{(t)} &= \sigma(\, \boldsymbol{W}_{ci}^{T} \cdot \boldsymbol{c}_{(t-1)} + \boldsymbol{W}_{hi}^{T} \cdot \boldsymbol{h}_{(t-1)} + \boldsymbol{W}_{xi}^{T} \cdot \boldsymbol{x}_{(t)} + \boldsymbol{b}_i) \\
\boldsymbol{f}_{(t)} &= \sigma(\, \boldsymbol{W}_{cf}^{T} \cdot \boldsymbol{c}_{(t-1)} + \boldsymbol{W}_{hf}^{T} \cdot \boldsymbol{h}_{(t-1)} + \boldsymbol{W}_{xf}^{T} \cdot \boldsymbol{x}_{(t)} + \boldsymbol{b}_f) \\
\boldsymbol{o}_{(t)} &= \sigma(\, \boldsymbol{W}_{co}^{T} \cdot \boldsymbol{c}_{(t)} + \boldsymbol{W}_{ho}^{T} \cdot \boldsymbol{h}_{(t-1)} + \boldsymbol{W}_{xo}^{T} \cdot \boldsymbol{x}_{(t)} + \boldsymbol{b}_o) \\
\boldsymbol{g}_{(t)} &= \tanh(\, \boldsymbol{W}_{hg}^{T} \cdot \boldsymbol{h}_{(t-1)} + \boldsymbol{W}_{xg}^{T} \cdot \boldsymbol{x}_{(t)} + \boldsymbol{b}_g) \\
\boldsymbol{c}_{(t)} &= \boldsymbol{f}_{(t)} \otimes \boldsymbol{c}_{(t-1)} + \boldsymbol{i}_{(t)} \otimes \boldsymbol{g}_{(t)} \\
\boldsymbol{y}_{(t)} &= \boldsymbol{h}_{(t)} = \boldsymbol{o}_{(t)} \otimes \tanh(\boldsymbol{c}_{(t)})
\end{aligned}
\tag{2.4}
$$

2.2.4 Self-attention based neural network

At the time of writing, an architecture that is becoming ubiquitous is the Transformer model, presented by Vaswani et al. [246] in their seminal paper "Attention is all you need." Firstly, proposed as an alternative architecture to the RNN encoder-decoder for language translation, it is finding application in diverse fields such as computer vision [68, 170], decision making, and time series analysis. It is primarily based on the self-attention mechanism, but as demonstrated by some researches [66] also the remaining part of the network significantly contributes to the success of this model. The original architecture is composed of two distinct parts: encoder and decoder. However, employing only the encoder module as a feature extractor is increasingly common. Only that part will be readjusted for some presented contributions in the dissertation. Therefore, the encoder part, adapted for classification or feature extraction tasks, is discussed in this section. We refer the interested reader to the well-written original paper [246].

The Transformer encoder is made of L layers with alternating H multi-head self-attention and feed-forward blocks. Dropout [226], Layernorm [14], and residual

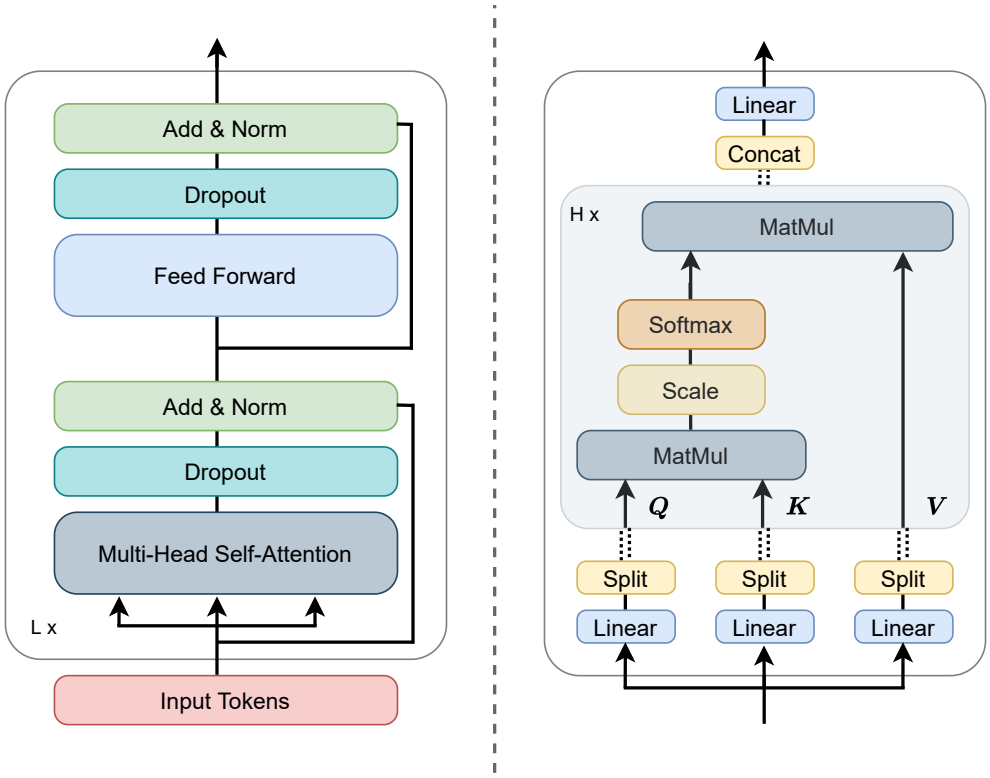


Figure 2.6: Transformer encoder layer architecture (left) and schematic overview of a multi-head self-attention block (right). Input tokens go through L encoder layers and H self-attention heads.

connections are applied after every block. The overall sequence of blocks of a Transformer encoder is summarized on the left of Figure 2.6.

Each feed-forward block is a multi-layer perceptron with two layers and GELU [97] non-linearity. The first layer expands the dimension from D_{model} to $D_{mlp} = 4 \cdot D_{model}$ and applies the activation function. On the other hand, the second layer reduces the dimension back from D_{mlp} to D_{model}.

Instead, the multi-head $\boldsymbol{QKV}$ self-attention mechanism (MSA) is based on a trainable associative memory with key-value vector pairs. For the l-th layer of the Transformer encoder and the h-th head, queries ($\boldsymbol{Q}$), keys ($\boldsymbol{K}$) and values ($\boldsymbol{V}$) are computed as $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K$ and $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V$ respectively, where $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$ and $\boldsymbol{W}_V$ belong to $\mathbb{R}^{D_{model} \times D_h}$, being D_h the dimension of the attention head. So, for each self-attention head (SA) and element in the input sequence $\boldsymbol{X} \in \mathbb{R}^{T \times D_{model}}$ we compute a weighted sum over all values $\boldsymbol{V}$. The attention weights A_{ij} are derived from the pairwise similarity between two elements of the sequence and their respective query Q_i and key K_j representations. Therefore, it

is possible to compute the attention weights $\boldsymbol{A} \in \mathbb{R}^{T \times T}$ for the l-th layer as

$$\boldsymbol{A} = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{D_h}}\right) \tag{2.5}$$

and the final weighted sum SA as

$$\text{SA}(\boldsymbol{X}) = \boldsymbol{A}\,\boldsymbol{V} \tag{2.6}$$

We perform this SA operation for all the H heads of the l-th layer. We concatenate the results and linearly project the output tensor to the original dimension model as

$$\text{MSA}(\boldsymbol{X}) = [SA_1(\boldsymbol{X}); SA_2(\boldsymbol{X}); ...; SA_{\text{H}}(\boldsymbol{X})]\,\boldsymbol{W}_{MSA} \tag{2.7}$$

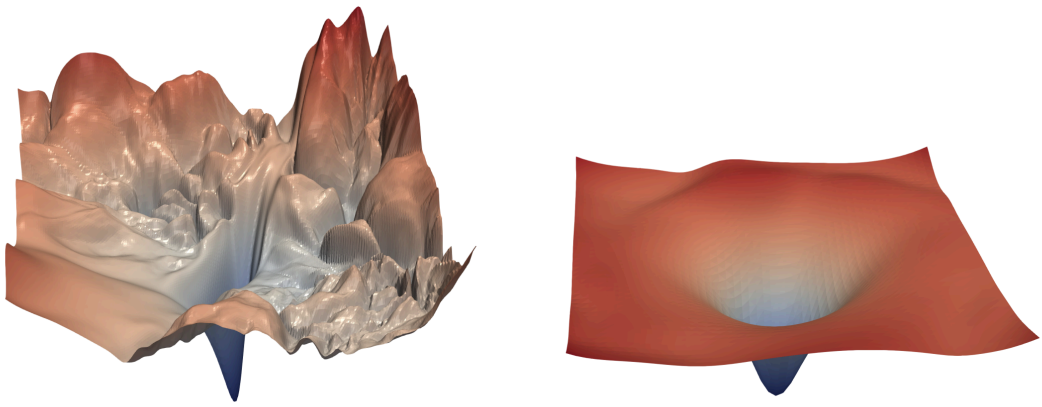
where $\boldsymbol{W}_{MSA} \in \mathbb{R}^{\text{H}\cdot D_h \times D_{model}}$. All operations are schematized on the right side of Figure 2.6. The input tensor $\boldsymbol{X}$ is firstly projected to $T \times \text{H} \cdot D_h$, and then a SA operation is performed to all H splits of the resulting projected tensor. Finally, all head outputs are concatenated and linearly projected back to D_{model}.

2.3 Training algorithms

The architecture is only part of the problem. In order to acquire knowledge from data, a complete learning algorithm requires a training procedure. Indeed, like other machine learning algorithms, ANN necessitates a mechanism to find the correct variable values for the selected task. The problem is always framed as an optimization problem. A cost/target function C or $\mathcal{L}$ provides measurable feedback of the performance of the model, and an optimization procedure provides a concrete way to update model variables in order to minimize the overall loss function and reduce the error. Data, architecture, and training procedures (e.g., regularization, initialization) are tightly bound, and they all concur at shaping the cost function and defining the possibility of reaching a global minimum. For instance, Figure 2.7 highlights the impact of the architecture on the resulting shape of the loss. All other training ingredients bring a similar contribution.

The difficulty of training deep neural networks or any statistical model is not only limited to minimizing the cost function given some training data but to generalizing with new samples not exploited during the optimization process. This vital problem and all its facets go under the name generalization. It is an additional degree of complexity in constructing architectures and training algorithms; therefore, it will be discussed in a dedicated chapter of the dissertation.

On the other hand, this section aims to provide insight into important aspects of training procedures and how minimizing a deep neural network cost function on a given training data. Firstly, for a loss to provide error feedback, it is in function of model parameters. It is parameterized by predictions of the model



(a) without skip connections (b) with skip connections

Figure 2.7: The loss surfaces of ResNet-56 with/without skip connections visualized with a methodology proposed by [148]. It clearly highlights how already the architecture can greatly influence the shape of the target loss function. Image taken from [148].

and by target outputs. Targets can be provided differently, with more or less supervision. Therefore, machine learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning (self-supervised learning), semisupervised learning, and reinforcement learning. The two first sections discuss the first two because necessary for the rest of the dissertation. For deep learning, there is a tendency to rename unsupervised learning as self-supervised learning to differentiate with more classical algorithms such as clustering, dimensionality reduction, or association rule learning. However, only self-supervised learning will be discussed, and as previously stated, so-called classical unsupervised learning algorithms adopted in the text are not discussed, but we refer an interested reader to [80].

Subsequently, the chapter introduces some standard optimization techniques for ANNs and procedures that update model variables toward the direction of minimum loss. Then, it shortly introduces backpropagation [210], the efficient algorithm that allows to propagate error information to all variables of the model and consequently enable networks optimization. Finally, some regularization and initialization techniques are introduced. They heavily influence the overall problem, and they usually determine the actual trainability of a deep neural network.

2.3.1 Training models with supervised learning

A cost function C is an essential ingredient for all training algorithms. It takes different forms, but the main aim is always to provide quantitative feedback on the performance of the network for a specific task. It takes as input the current

parameters value $\boldsymbol{w}$ and is parametrized by the architecture of the network h and the target y of the chosen task. Targets can be provided differently, with more or less supervision.

In supervised learning, the training set $\mathcal{D} = (x_i, y_i)_{i=1}^{N}$ completely includes the desired solution y_i. So, the aim is to tune variables in order to find the function that connects inputs and outputs. Supervised learning imposes an elementary feedback signal to models but is responsible for the current success of deep learning in the industry. Indeed, the AI field has made tremendous progress in developing AI systems that learn from massive amounts of carefully labeled data.

There are two popular supervised tasks: classification and regression. The first one classifies given data points in different classes/groups. Depending on the number of outputs and classes, it can be binary, multiclass, multilabel, and multioutput. Binary and multiclass have only one output, but two classes or more than two, respectively. On the other hand, multilabel and multioutput have more than one output but with a binary or multiclass for each output, respectively.

The most adopted supervised loss function for classification is by far cross-entropy. As regard binary problems, it can be written for a single sample in the following form:

$$C(\boldsymbol{w}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \tag{2.8}$$

where $\hat{y}$ is the model prediction. Instead, equation (2.8) can be rewritten to take into account multiple classes.

$$C(\boldsymbol{w}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{2.9}$$

If equation (2.8) is usually used in conjunction with sigmoid as non-linear activation functions, categorical cross-entropy is usually directly applied to the linear outputs (logits) of the last layer.

On the other hand, regression can be single, multiple, univariate, or multivariate regression. The first two are defined by the number of features/predictors of the input. Conversely, the latter two depend on the number of values to predict. A distance metric between model predictions and target values is the preferable choice of loss in all cases. For instance, most problems make use of a certain grade of the Minkowski distance

$$C(\boldsymbol{w}) = (|y - \hat{y}|^p)^{1/p} \tag{2.10}$$

where p is the order of the Minkowsky distance. Usually, the higher is the norm index, the more it focuses on large values and neglects small ones. For instance, $p = 2$ is more sensitive to outliers than $p = 1$. However, when outliers are rare, $p = 2$ performs very well and is generally preferred.

Even if supervised signals are very naive, they can train excellent deep neural networks with high generalization capabilities. This paradigm has a proven track

record for training specialist models that perform exceptionally well on the task they were trained to. Nevertheless, the major drawback is the necessity of the targets. Indeed, ground-truth for most applications are very time-consuming to collect or sometimes even not available. Consequently, more and more researchers are investigating unsupervised techniques where labels are not needed to construct a loss function.

2.3.2 Training models with self-supervised learning

Self-supervised learning (SSL) is one of the most promising areas of AI research today. Indeed, learning effective representations without labels is a long-standing problem. In contrast to supervised learning, it does not require human supervision and opens the possibility of learning incrementally with a large amount of data. Moreover, supervised learning losses provide weak learning signals more concerned with the given task than in learning semantically rich representations. On the other hand, a self-supervised algorithm can train strong network encoders and then fine-tune or transfer their knowledge for a specific task. Indeed, self-supervised objective functions provide much richer learning signals, and, consequently, they produce representations with much more generalization power.

Most mainstream approaches fall into one of two classes: generative or discriminative. The first class of algorithms [86] is computationally expensive because it forces work with data-level generation. On the other hand, similar to supervised learning algorithms, discriminative approaches learn representation through an objective function. However, both the inputs and labels are derived from an unlabeled dataset. Hence, the name "self-supervised." Moreover, the term self-supervised is more accepted than the previously used term "unsupervised learning, " which suggests that the learning uses no supervision.

Self-supervised learning has had a particularly profound impact on NLP and significantly advanced the field. However, the same techniques cannot be directly applied to different domains such as computer vision. The main reason is that it is considerably more challenging to represent uncertainty in image prediction than for words. In fact, we do not know how to represent suitable probability distributions over high-dimensional continuous spaces, such as the set of all possible videos.

Nevertheless, at the time of writing, discriminative approaches based on contrastive or non-contrastive learning in latent space have shown great promise, achieving state-of-the-art results. In particular, both approaches can be modeled under the unified framework of an energy-based model (EBM). An EBM is a trainable system that takes two inputs and outputs a compatibility score between the two. Therefore, the model would tell us to which extent the two inputs are in accordance, providing low or high energy if there is compatibility or not, respectively. The training procedure of an energy-based system comprises two parts: showing compatible and incompatible examples. However, the difficulty relies on

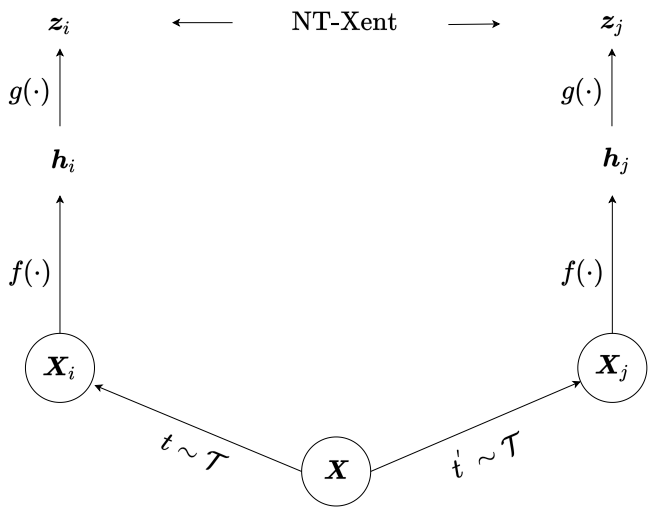


Figure 2.8: SimCLR graphical summary. A pair of inputs are created with two separate data augmentation operators sampled from the same family of augmentations. A base encoder $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. Image inspired from [44].

obtaining high energy when the two inputs are different. Indeed, a model could collapse to a trivial solution without a specific way of doing so and produce the same representation for all inputs.

There are two categories of techniques to avoid collapse: contrastive and regularization methods. In this section are presented one example for each of the two categories that will be important for the rest of the dissertation: SimCLR [44] and self-Distillation with No labels (DINO) [36]. Despite their simplicity, both techniques show promising results, exhibiting interesting properties of representation learned with self-supervision. Indeed, after the unsupervised pretraining, self-supervised techniques leave an encoder network that projects input data in a semantically rich latent space. Output representations and model inner activations can also be inspected and exploited for a downstream task. For instance, train a linear classifier on top of the frozen base network.

SimCLR: a simple framework for contrastive learning

Contrastive methods are based on the simple principle of constructing pairs of inputs that are not compatible and adjusting the parameters of the model so that the corresponding output energy is large. In 2020, Ting et al. presented SimCLR [44], a simple framework for contrastive learning of visual representations that inspired successive research.

A visual representation of SimCLR is depicted in Figure 2.8. SimCLR learns representations by maximizing agreement between differently augmented views of

the same data example, while minimizing agreement of negative data example. In particular, a stochastic data augmentation module $\mathcal{T}$ transforms any given data example in two related views of the same example, $\mathbf{X}_i$ and $\mathbf{X}_j$. These two are considered positive pair and they are usually the product of a certain pipeline of transformations. For instance, random cropping, random color distortions and random Gaussian blur. Successively, a base encoder $f(\cdot)$ is used to form a joint embedding architecture also known as Siamese network. It is an architecture composed with two copies of the same network. One network is fed with $\mathbf{X}_i$ and the other with $\mathbf{X}_j$. Consequently, the two representations $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ generated are further processed by a projection head $g(\cdot)$ (MLP with one hidden layer). It maps representations to the space where the contrastive loss is applied, $\boldsymbol{z}$. In particular, $\boldsymbol{z} = g(\boldsymbol{h})$ is trained to be invariant to data transformation. Thus, without g many important information could be removed from $\boldsymbol{h}$, resulting in less rich and poor representations. Finally, given a set $\{\boldsymbol{X}_k\}$, including a positive pair of examples, $\mathbf{X}_i$ and $\mathbf{X}_j$, the following contrastive objective is applied

$$\mathcal{L} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)\tau)} \tag{2.11}$$

where $\mathbb{1}_{[k\neq i]} \in \{0,1\}$ is an indicator function evaluating to 1 if $k \neq i$ and τ denotes a temperature parameter. Equation (2.11) is named normalized temperature-scaled cross entropy loss and it has the simple effect to bring near the positive pair at the numerator while pushing away negative samples.

Despite the simplicity of this formulation, using the linear classifier evaluation protocol, SimCLR achieved approximately 70% top-1 accuracy with ResNet-50 as a base encoder. On the other hand, it is possible to achieve 75% with the same architecture with traditional supervised algorithms. However, it is not only impressive how SimCLR largely closed the gap with supervised learning, but representations learned with it demonstrate better transferability.

DINO: self-distillation with no labels

Contrastive objectives are simple to frame but are also very inefficient to train. Indeed, in high-dimensional spaces such as images, one image can be different from another in many ways. On the other hand, non-contrastive methods require regularization to avoid collapsing into a trivial solution. In 2021, Caron et al. [36] proposed a simple self-supervised approach, DINO, that shares various tricks introduced by previous notable works: BYOL [89], SwAV [37], MoCO [96]. Momentum encoder, multi-crop augmentation, centering, and sharpening are all effective solutions to stabilize training and avoid collapse.

The overall architecture is very similar to Figure 2.8. Indeed, DINO features two base encoders with the same architecture and a projection head on top of the overall framework. However, one of the two encoders, called teacher from distillation [99], is

built with a momentum encoder. Therefore, only one network, the student network, is trained with backpropagation. Conversely, the teacher network is updated with an exponential moving average of the student parameters, $\theta_s \leftarrow \lambda\theta_t + (1-\lambda)\theta_s$.

More precisely, in DINO the student network f_{θ_s} has to match the output of a given teacher network f_{θ_t}, parameterized by θ_s and θ_t respectively. From a given image $\mathbf{X}$, it is generated a set V of different views of the same image. The set contains two global views, $\mathbf{X}_1^g$, $\mathbf{X}_2^g$ and several local views of smaller resolution. As for SimCLR, a data augmentation operator transforms each view (e.g., color jitter, Gaussian noise or blur, rotations), avoiding collapse. Subsequently, both networks, with the help of a projection head $g(\cdot)$, output a probability distribution over K dimensions denoted by $\boldsymbol{p}_s$ and $\boldsymbol{p}_t$. Probabilities are obtained by normalizing the output of the projection head with a softmax function

$$\boldsymbol{p}(\mathbf{X}) = \frac{\exp(f_\theta(\mathbf{X})/\tau)}{\sum_{k=1}^{K}\exp(f_\theta(\mathbf{X})_k/\tau)} \tag{2.12}$$

where τ is a temperature parameter that controls the sharpness of the output distribution. Each network has a distinct temperature parameter. Finally, the student is optimized using backpropagation and cross-entropy loss between the two distributions

$$\min_{\theta_s} \sum_{\mathbf{X}\in\{\mathbf{X}_1^g,\mathbf{X}_2^g\}} \sum_{\substack{\mathbf{X}^{'}\in V \\ \mathbf{X}^{'}\neq\mathbf{X}}} H(\boldsymbol{p}_t(\mathbf{X}), \boldsymbol{p}_s(\mathbf{X}^{'})) \tag{2.13}$$

where $H(a,b) = -a\log b$. In order to avoid collapse, besides augmentation of the different views and output sharpening with τ, teacher predictions are centered using first-order batch statistics: $g_t(x) \leftarrow g_t(x) + c$. The center c is updated with an exponential moving average, which allows the approach to work well across different batch sizes. Centering prevents one dimension to dominate but encourages collapse to the uniform distribution, while the sharpening has the opposite effect.

Despite again the simplicity of this self-supervised approach, DINO is able to achieve 75.3% top-1 accuracy on ImageNet with ResNet50, training a linear classifier on top of frozen features. Moreover, DINO highlighted properties of ViT not emerged with supervised training. For instance, it is possible to demonstrate that ViT features explicitly contain the scene layout and, in particular, object boundaries. Finally, ViT features perform particularly well with a basic nearest neighbors classifier without any fine-tuning, linear classifier, or data augmentation, achieving 78.3% top-1 accuracy on ImageNet with a ViT-S with patches of dimension 8^2.

2.3.3 Optimizing networks

A generic objective function C provides a straightforward and measurable mechanism to receive feedback on the performance of a model, taking model variables as input. Indeed, it is parameterized by model predictions and target outputs, and

it provides a score of the training status of a model. Therefore, without taking into account generalization, the general aim of learning is to obtain $C(\boldsymbol{w}) \approx 0$, where $\boldsymbol{w}$ is the array that contains all n weights of a network. However, minimizing highly non-convex neural loss functions is not so straightforward, and it usually requires elaborated optimization techniques that do not assure the convergence to a global minimum. Nevertheless, even if the literature of ANNs forecasted difficulty in convergence, it turns out that due to the nature of the high-dimensional space, it is rare to get stuck in local optima, and when it is the case, they are usually reasonably close to the global optimum.

In order to find a minimum of the cost function, it is helpful to reason about the effect of variable variations on the cost function itself. Calculus tells us that C changes as follows:

$$\Delta C \approx \frac{\partial C}{\partial w_1}\Delta w_1 + \frac{\partial C}{\partial w_2}\Delta w_2 + \frac{\partial C}{\partial w_3}\Delta w_3 + \cdots + \frac{\partial C}{\partial w_n}\Delta w_n \tag{2.14}$$

All different Δw_i can be merged in an array $\Delta \boldsymbol{w} = (\Delta w_1, \Delta w_2, \ldots, \Delta w_n)^{\mathrm{T}}$. The same is true for all partial derivatives, $\nabla C = (\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}, \ldots, \frac{\partial C}{\partial w_n})^{\mathrm{T}}$. ∇C is the gradient vector of C, containing all partial derivatives of the loss function respect to its inputs $\boldsymbol{w}$. With those definitions, equation (2.14) can be rewritten as the following equation

$$\Delta C \approx \nabla C^{\mathrm{T}} \cdot \Delta \boldsymbol{w} \tag{2.15}$$

where $\cdot$ is the dot product. What is very useful of equation (2.15) is that it allows to choose $\boldsymbol{w}$ in a way to make ΔC negative and therefore minimizing it. Indeed, if we choose

$$\Delta \boldsymbol{w} = -\eta \nabla C \tag{2.16}$$

where, η is a small, positive parameter it is possible to obtain

$$\Delta C \approx -\eta \nabla C^{\mathrm{T}} \cdot \nabla C = \eta \parallel \nabla C \parallel^2 \tag{2.17}$$

where $\parallel \nabla C \parallel^2$ is greater or equal to zero and therefore $\Delta C \leq 0$. Equation (2.16) is called gradient descent. It provides a simple and straightforward way to update model variables and converge towards the minimum. It gives us a way of repeatedly changing values of $\boldsymbol{w}$ in order to find a minimum of the function C.

The loss function has the variables of the model to evaluate and usually produces a scalar. Data samples are the parameters and, in conjunction with the architecture and training procedures, determine the overall loss shape. However, especially for deep learning, evaluating the cost function with all data is not efficient. On the contrary, due to the highly non-convex nature of the problem, several studies point in the direction that estimating C with small batches of data and updating the network more frequently can largely improve the performance of the optimization algorithm [223]. Consequently, in practical cases, only a subset of the whole training

rate is adopted in order to estimate the gradient of the loss function, as shown in the following equation

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_j \tag{2.18}$$

where m is a subset of the available dataset, usually named mini-batch size, and $\nabla \boldsymbol{C}_j$ is the gradient of the cost function computed with the j-th sample. Therefore, the updating rule of equation (2.16) becomes

$$w_i \rightarrow w_i^{'} = w_i - \frac{\eta}{m} \sum_{j} \frac{\partial C_j}{\partial w_i} \tag{2.19}$$

where $w_i^{'}$ is the value of the i-th weight at the next iteration. In the particular case in which m is equal to one the algorithm takes the name of stochastic gradient descent.

With most practical problems, equation (2.19) is too slow to converge with a deep neural network. A huge speed boost could come from using alternatives to gradient descent presented in the literature. The following paragraphs briefly present some of the most popular alternative optimizers.

Momentum-based gradient descent

Using Taylor's theorem, the cost function can be approximated near a point $\boldsymbol{w}$ by the following equation

$$C(\boldsymbol{w} + \Delta \boldsymbol{w}) = C(\boldsymbol{w}) + \sum_{i} \frac{\partial C}{\partial w_i} \Delta w_i + \frac{1}{2} \sum_{ij} \Delta w_i \frac{\partial^2 C}{\partial w_i \partial w_j} \Delta w_j + \cdots + R(\Delta w) \tag{2.20}$$

where $R(\Delta w)$ is the remainder of the series. Equation (2.20) can be rewritten in a more compact form as

$$C(\boldsymbol{w} + \Delta \boldsymbol{w}) = C(\boldsymbol{w}) + \nabla \boldsymbol{C}^{\mathrm{T}} \dot{\Delta} \boldsymbol{w} + \frac{1}{2} \Delta \boldsymbol{w}^{\mathrm{T}} \boldsymbol{H} \Delta \boldsymbol{w} + \cdots + R(\Delta w) \tag{2.21}$$

where $\boldsymbol{H}$ is the Hessian matrix. Discarding term of higher order, using calculus it is possible to demonstrate with calculus that in order to minimize the right-hand side of equation (2.21), $\Delta \boldsymbol{w}$ should be equal to equation (2.22).

$$\Delta \boldsymbol{w} = -\boldsymbol{H}^{-1} \nabla C \tag{2.22}$$

That is known as Hessian technique or Hessian optimization. However, even if theoretical and empirical results show that Hessian methods converge on a minimum in fewer steps than standard gradient descent, it is much more inefficient than the latter. Therefore, it is never adopted for deep learning. Nevertheless, it gives an important intuition of what can be done to improve convergence. Indeed, Hessian

optimization incorporates information about the gradient and information about how the gradient is changing. Proposed by Boris Polyak in 1964 [198], momentum-based gradient descent is based on a similar intuition but avoids large matrices of second derivatives. It simply makes use of gradients as acceleration and not speed anymore. It adds new gradients to a new vector that keeps part of previous ones. That gives a kind of inertia of previous weights, speeding up convergence and being more resilient to local minima. Therefore, equation (2.19) can be rewritten as

$$\begin{aligned} v_i \to v_i^{'} = \beta v_i - \frac{\eta}{m} \sum_j \frac{\partial C_j}{\partial w_i} \\ w_i \to w_i^{'} = w_i + v_i^{'} \end{aligned} \tag{2.23}$$

where β is hyper-parameter which controls the amount of damping or friction in the system. It is easy to verify that if the gradient remains constant, the terminal velocity equals the gradient multiplied by $\eta/(1-\beta)$. Therefore, if $\beta = 0.9$, the terminal velocity is ten times faster than plain gradient descent.

Gradient descent with Nesterov acceleration

In 1983, Yurii Nesterov [183] proposed a small variation for the momentum-based gradient descent algorithm. The main idea is to measure the gradient of the cost function not at the local position $\boldsymbol{w}$, but slightly ahead in the direction of the momentum, at $\boldsymbol{w} + \beta \boldsymbol{v}$. Therefore, equation (2.23) can be written as

$$\begin{aligned} v_i \to v_i^{'} = \beta v_i - \frac{\eta}{m} \sum_j \frac{\partial C_j(\boldsymbol{w} + \beta \boldsymbol{v})}{\partial w_i} \\ w_i \to w_i^{'} = w_i + v_i^{'} \end{aligned} \tag{2.24}$$

where $C(\boldsymbol{w} + \beta \boldsymbol{v})$ is the cost function evaluated at $\boldsymbol{w} + \beta \boldsymbol{v}$. Nesterov accelerated gradient is almost always faster than the classical algorithm because it can peek a bit farther in the direction of the gradient and adjust the overall updating direction. Finally, Nesterov variation can be easily combined with all other optimization techniques and only to momentum-based gradient descent.

Root Mean Squared Propagation gradient descent

Root Mean Squared Propagation or RMSProp gradient descent has been firstly proposed by Geoffrey Hinton and Tijmen Tieleman in 2012[1] in order to better adapt AdGrad [69] for deep learning problems. Indeed, AdaGrad dynamical adapts the learning rate of each variable using the magnitude of gradients. However, even if

[1] https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

it performs well for simple quadratic problems, it often scales down the learning rate so much that the algorithm does not reach an acceptable solution. That happens because the square of the gradients are always accumulated: $s_i \rightarrow s_i^{'} = s_i + (\partial C/\partial w_i)^2$. They instead proposed to adopt and exponential decay sum in order to exploit the benefit of an adaptive learning rate, while keeping the training run. Therefore, RMSProp is defined by the following two equations written in vectorial form

$$\begin{aligned} \boldsymbol{s} \rightarrow \boldsymbol{s}^{'} &= \beta \boldsymbol{s} + (1-\beta)\left(\nabla C \odot \nabla C\right) \\ \boldsymbol{w} \rightarrow \boldsymbol{w}^{'} &= \boldsymbol{w} - \eta \frac{\nabla C}{\sqrt{\boldsymbol{s}^{'} + \epsilon}} \end{aligned} \tag{2.25}$$

where $\odot$ is is the Hadamard product, ϵ is a small value to avoid zero division and ∇C is the estimated gradient computed with equation (2.18). In conclusion, RMSProp decays the learning rate looking at the magnitude of the gradients of the most recent iterations. It does faster for steep dimensions than for dimensions with gentler slopes.

Adaptive moment estimation

Adaptive moment estimation, better known as Adam, has been proposed by Diederik P. Kingma and Jimmy Ba in 2014 [133]. It is by far the most adopted optimization algorithm and it is based on a simple idea: adding moment-based gradient descent to RMSProp. Indeed, just like momentum optimization it keeps track of an exponentially decaying average of past gradients (gradients mean, first moment) and as RMSProp it keeps track of an exponentially decaying average of past squared gradients (gradients variance, second moment). Finally, it adds two correction factors in order to avoid a biased toward 0 of $\boldsymbol{v}$ and $\boldsymbol{s}$ at the beginning of the training. Adam can be summarized by the following equations

$$\begin{aligned} \boldsymbol{v} \rightarrow \boldsymbol{v}^{'} &= \beta_1 \boldsymbol{v} - (1-\beta_1)\nabla C \\ \boldsymbol{s} \rightarrow \boldsymbol{s}^{'} &= \beta_2 \boldsymbol{s} + (1-\beta_2)\left(\nabla C \odot \nabla C\right) \\ \hat{\boldsymbol{v}} &= \frac{\boldsymbol{v}^{'}}{1-\beta_1^t} \\ \hat{\boldsymbol{s}} &= \frac{\boldsymbol{s}^{'}}{1-\beta_2^t} \\ \boldsymbol{w} \rightarrow \boldsymbol{w}^{'} &= \boldsymbol{w} - \eta \frac{\hat{\boldsymbol{v}}}{\sqrt{\hat{\boldsymbol{s}} + \epsilon}} \end{aligned} \tag{2.26}$$

where β_1 and β_2 are the momentum and scaling decay hyperparameter, respectively. On the other hand t represents the iteration number. Many alternatives to Adam have been proposed, such as Nadam, AdaMax, to improve stability and convergence speed. However, it depends on the dataset, and most of the time, experimentations are the only method of discernment.

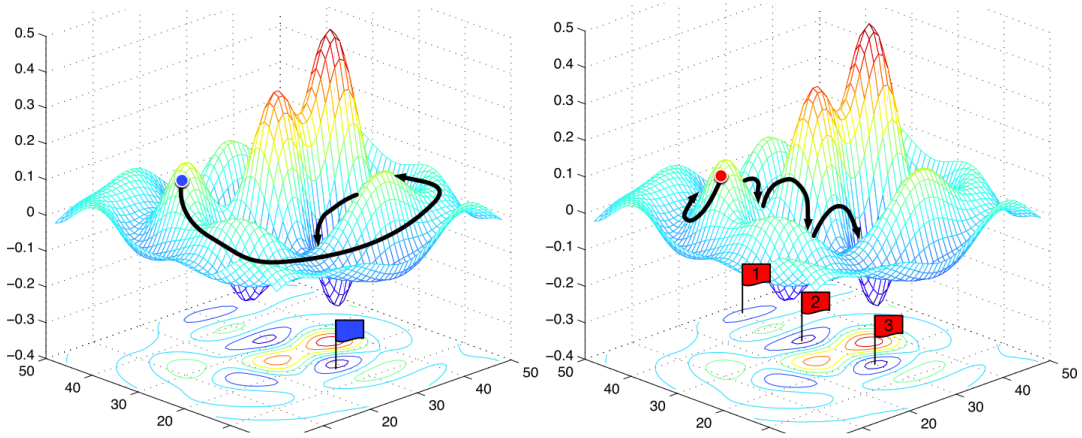


Figure 2.9: Example of a loss surface with a toy model with only two parameters. Both examples highlight how a learning rate scheduler can escape local minima and perform model selection. Image taken from [108].

Finally, even if Adam is an adaptive learning rate algorithm requiring minor tuning of this important hyperparameter, learning rate schedulers still play an essential role in training deep neural networks. Many learning rate schedulers have been proposed in the literature. However, as for choosing the best optimizer to use, the learning schedule selection is also more an art than science. The most adopted are linear scheduling, exponential scheduling, cosine scheduling with warmup [246], picewise constant scheduling, 1cycle scheduling [222] and performance scheduler. Moreover, as shown in Figure 2.9, learning schedulers such as cyclic learning rate do not only help to escape local minima but provide a simple way to perform model selection, taking parameters snapshot at the end of each cycle. Huang Gao et al. [108] further shows how it is possible to use all snapshots to create with a simple training a performant ensemble.

2.3.4 Backpropagation

The previous section presented how minimizing highly non-convex neural loss functions require inevitably computing the gradient of the target function with respect to all trainable variables. However, how can it be done with deep neural networks? With reference to Figure 2.2, given a loss function that has as inputs weights ($\mathbf{W}$) and biases ($\boldsymbol{B}$) of the model $C(\mathbf{W}, \boldsymbol{B})$, we need an efficient mechanism to compute the contribution of each variable to it. Successively, knowing that information different optimization techniques can be adopted.

The problem solved by backpropagation is the efficient computation of $\frac{\partial C}{\partial w_{i,j}^{l}}$

and $\frac{\partial C}{\partial b_i^l}$, where $w_{i,j}^l = W_{i,j}^l$ and b_i^l are the weight i, j and bias i of layer l. It has been named backpropagation because, starting from the output layer of a network, it can compute all partial derivatives going backward through the network. Each sample first requires a forward pass, making a prediction. Then, it measures the error with a loss function, and it goes through each layer in reverse to measure the error contribution from each connection. Using backpropagation is possible to derive the following equations:

$$\boldsymbol{\delta}^L = \nabla_a C \odot \sigma'(\boldsymbol{z}^L) \tag{2.27}$$

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^{\mathrm{T}} \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \tag{2.28}$$

$$\frac{\partial C}{\partial w_{i,j}^l} = a_j^{l-1} \delta_i^l \tag{2.29}$$

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l \tag{2.30}$$

where $\odot$ is the Hadamard product and σ' is the derivative of the activation function. It is clear from the equations that it is possible to perform the same operations simultaneously with more samples. However, that requires memorizing all activations for each sample, drastically increasing the overall memory requirement. Moreover, looking at the four equations, it is intuitive to understand the underline mechanism: firstly, error values are computed for the output layer $\boldsymbol{\delta}^L$ and then are backpropagated up to the input layer through equation (2.28). Finally, equations (2.29) and (2.30) relate computed errors with the partial derivatives.

2.3.5 Initialization techniques

Initialization plays an essential role in the overall optimization process: it determines the initial position over the loss function and the convergence speed of the training process. Indeed, techniques such as transfer learning [156, 236] and unsupervised pretraining can significantly affect the final performance of a network. Moreover, parameters initialization can significantly affect the magnitude of signals going through the network. Indeed, even feedforward ANNs have signals traveling in both directions: the forward direction when making predictions and the reverse direction when backpropagating error signals. An ANN needs signals to flow properly in both directions; otherwise, instability issues could arise, also known as vanishing and exploding gradients. Indeed, if signals variance is not preserved from one layer to another, the compositionality nature of ANNs could make magnitude shrink (vanishing gradient) or saturate (exploding gradient), resulting in unstable predictions or training.

Since this problem has been observed, many initialization techniques, activation functions, or magnitude clipping (gradient clipping [191]) strategies have been

presented to alleviate this issue. In the following are presented three of the most adopted initialization schemes.

Glorot, He and LeCun initializations

In 2010, Xavier Glorrot and Youshua Bengio [84] demonstrated that logistic regression and weight initialization with normal distribution with zero mean and unitary variance, two popular ingredients at the time, were the two principles responsible for the vanishing and exploding gradient problem. Indeed, they showed that with this activation function and initialization scheme, the variance of the outputs of each layer is much greater than the variance of its inputs. Therefore, going forward in the network, the variance keeps increasing until the activation function saturates at the top layers.

In order to alleviate the unstable gradients problem, Glorot and Bengio mathematically searched the correct initialization scheme imposing equal input and output variance for signals flowing in both directions. They proved the impossibility of guaranteeing both constraints when input and output layers have different neurons. However, they proposed an excellent compromise to mitigate the problem. Defining fan_{avg} as the mean of input and output neurons $(fan_{in} + fan_{out})/2$, they proposed the initialization scheme summarized in Table 2.1: weights value sampled from a normal distribution with zero mean and $1/fan_{avg}$ variance.

However, the initialization scheme introduced in [84] was developed targeting the activation functions popular at that time: tanh, sigmoid, and softmax nonlinear activations. Successive papers demonstrated that the same scheme is not optimal for more recent activation functions. In Table 2.1 are reported other two common initializations [94, 185] for activations such as SELU and ReLU and variants.

2.3.6 Regularization techniques

Neural networks losses are highly non-convex functions, and the effectiveness of optimization techniques largely depends on the overall shape of the loss function itself. Data, architecture, and regularization techniques are all essential variables for the problem, and they all contribute to modeling the surface of the cost function to a large extent.

Initialization	**Mean** (μ)	**Variance** (σ^2)	**Activation Functions**
LeCun [185]	0	$1/fan_{\text{in}}$	SELU [134]
He [94]	0	$2/fan_{\text{in}}$	ReLU [181], PReLU [94], LeakyReLU [258]
Glorot [84]	0	$1/fan_{\text{avg}}$	Tanh, logistic, softmax

Table 2.1: Different initialization schemes with related normal distribution mean and variance.

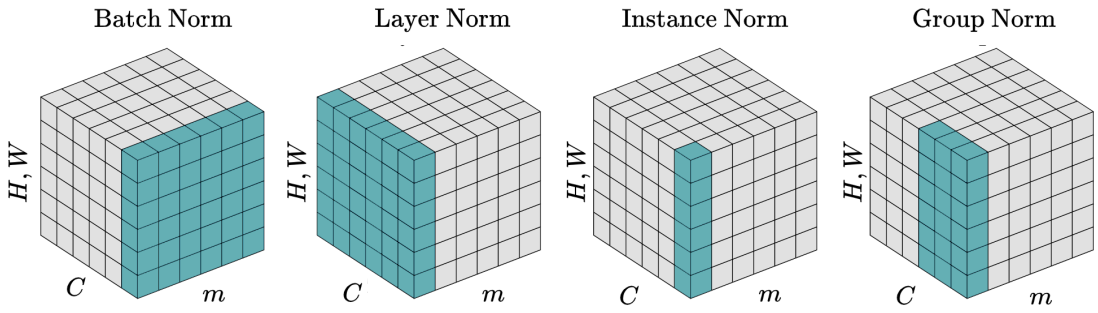


Figure 2.10: Graphical comparison between different normalization methods. It takes into account an image as a data sample. However, concepts are also valid for all other types of input. m is the batch axis, C are the number of channels, and H, W are the spatial axes. The highlighted small cubes are normalized by the same mean and variance, computed by aggregating their values. Image inspired by [256].

Some regularization techniques affect graph computations and more classical ones that directly intervene on the actual loss function. In the following subsections are presented a series of techniques of both families.

Network signals normalization

In a 2015 paper, Sergey Ioffe and Christian Szegedy [114] proposed Batch Normalization (BN), one of the techniques that progressively became pervasive in every deep learning architecture. It is so common to adopt BN that it is often omitted in the architecture diagrams, as it is assumed that BN is added after every layer.

The technique is inspired by initialization techniques and consists of adding an operation in the model just before or after the activation function of each hidden layer. BN simply zero-centers and normalizes each input, then it scales and shifts the result using two new parameter vectors per layer: one for scaling, the other for shifting. The following equations summarizes the simple computations performed by a BN layer

$$\begin{aligned} \boldsymbol{\mu}_B &= \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}^{(i)} \\ \boldsymbol{\sigma}_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_B) \\ \hat{\boldsymbol{x}}^{(i)} &= \frac{\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2+\epsilon}} \\ \boldsymbol{z}^{(i)} &= \boldsymbol{\gamma}\odot\hat{\boldsymbol{x}}^{(i)}+\boldsymbol{\beta} \end{aligned} \tag{2.31}$$

where $\boldsymbol{x}^{(i)}$ is a generic representation inside the network. Indeed, BN, depending from the application, can be applied before or immediately after the activation

function. The vector of input means and variances are estimated with the m samples of the mini-batch during training. At the same time, a moving average of the two arrays is updated and exploited during inference.

Batch Normalization considerably improves convergence speed and final accuracy. Vanishing gradient problems are strongly mitigated, making the network less subject to weight initialization and adopting larger learning rates. Moreover, the continuous estimation of the mean and variance acts as a strong regularizer. Finally, even if each epoch requires more computation with BN, the wall time is usually shorter, and during inference, BN can be merged with other layers.

Inspired by the improvements brought by BN, different literature works proposed different normalization schemes. A summary of the most popular ones is reported in Figure 2.10. In general, they differ from the origin of the statistics. Most notably, Layer Normalization [14] normalizes a whole layer using statistics of a single instance. On the other hand, Instance Normalization [240] normalizes with only statistics of a single instance too, but it also normalizes each feature separately. Both techniques do not make use of batches of information. Therefore, the inference phase is facilitated. However, the regularization effect is far more pronounced. Finally, Group Normalization [256] puts itself in the middle between these last two. It normalizes using statistics of a single sample, but not using a single feature but a group of features.

Dropout

Similar to BN, Dropout is a regularization technique that modifies the architectural graph of the network. Proposed in 2012 [102], it has proven to be highly

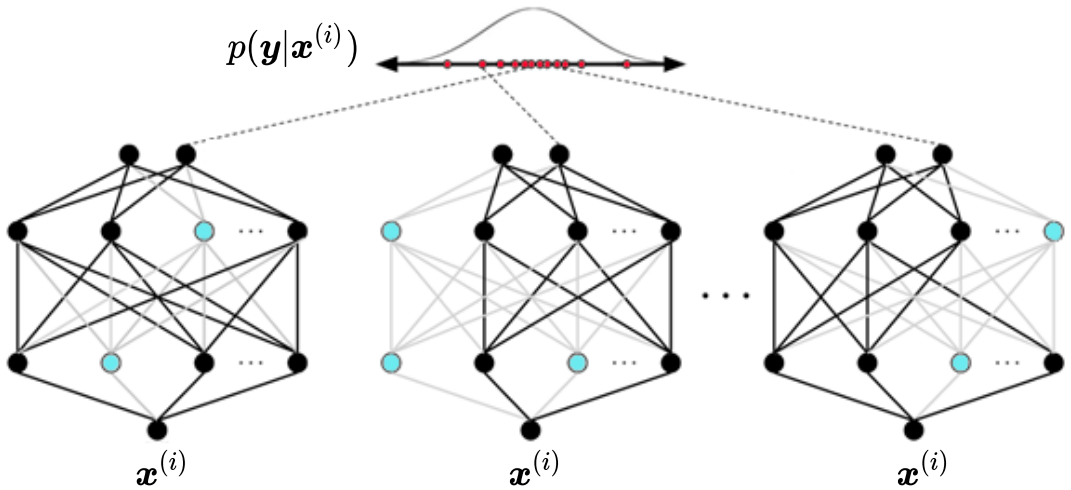


Figure 2.11: The number of forward passes through a data sample $\boldsymbol{x}^{(i)}$ should be evaluated quantitatively, but 30-100 is an appropriate range to consider [78].

effective, usually providing a performance boost to all deep learning architectures. Moreover, the algorithm is relatively simple: at every training step, each neuron of a dropout layer has a probability p of being temporarily "turned off." That avoids co-adaptation of feature detectors and makes each neuron "self-sufficient": neurons of a layer cannot rely on nearby neurons to make predictions. Therefore, they have to be as helpful as possible on their own.

Dropout is very simple to adopt and does not impact the computational training request. It only requires to compensate for the difference between training and inference. Indeed, if a layer has a drop probability p during training, at testing time, a neuron of the successive layer would be connected to a much higher number of neurons. Therefore, it is required to multiply each input connection weight by the inverse probability $1 - p$ after training. Alternatively, it is possible to divide each neuron output by $1 - p$ during training. These two alternatives are not equivalent, but they produce very similar results.

Finally, in 2016 Yarin Gal and Zoubin Ghahraman [78] showed that training a dropout network is mathematically equivalent to approximate Bayesian inference in a specific type of probabilistic model known as Deep Gaussian Process. The predictive distribution can be denoted as $p(\boldsymbol{y}|\boldsymbol{x}, \mathcal{D})$, with $\boldsymbol{y}$ the ground-truth, $\boldsymbol{x}$ the input array and $\mathcal{D}$ the training samples. They introduced Monte Carlo Dropout, a technique that provides a much better measure of the model uncertainty, estimating the predictive distribution. It simply maintains dropout during inference, making a certain number d of predictions, as shown in Figure 2.11. Therefore, at each inference step, a different number of neurons are active, producing slightly different predictions at each step and opening the possibility to compute some statistics regarding the uncertainty of the network. Indeed, each dropout configuration corresponds to a different sample from the approximate parametric posterior distribution $q(\boldsymbol{w}|\mathcal{D})$, $\boldsymbol{w}_t \sim q(\boldsymbol{w}|\mathcal{D})$. Therefore, sampling from the approximate posterior $q(\boldsymbol{w}|\mathcal{D})$ enables Monte Carlo integration of the model's likelihood, which uncovers the predictive distribution, as follows:

$$p(\boldsymbol{y}|\boldsymbol{x}) \approx \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) q(\boldsymbol{w}|\mathcal{D}) d\boldsymbol{w} \approx \frac{1}{T} \sum_{t=1}^{T} p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}_t) \tag{2.32}$$

where T is the number of forward passes. If we assume the likelihood to be Gaussian, the Monte Carlo simulation provides an estimation of the variance and mean of the distribution.

Lasso, Ridge and Elastic Net regularizations

Lasso, Ridge and Elastic Net regressions directly affect the loss function shape $C(\boldsymbol{w})$. They all modify the cost function to constrain the model and help the overall optimization process.

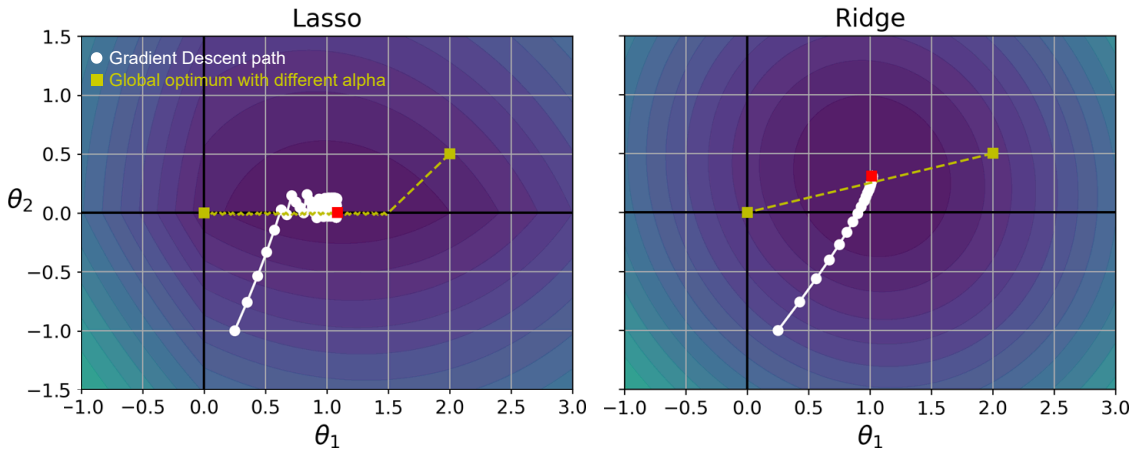


Figure 2.12: Comparison between Lasso and Ridge regularizations techniques. The small white circles show the path that Gradient Descent takes to optimize the two model parameters θ_1 and θ_2. On the other hand, if we increase the α parameter, the global optimum would move left along the dashed yellow line for either regularization.

The least Absolute Shrinkage and Selection Operator Regression (Lasso Regression) adds the l_1 norm of the weight vector

$$C(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \alpha \sum_{i=1}^{n} |w_i| \tag{2.33}$$

where $\mathcal{L}(\boldsymbol{w})$ is generic loss function with input weights $\boldsymbol{w}$ and *alpha* is a parameter that controls the level of regularization. High α values result in a very regularized model with all weights close to zero. Indeed, this type of regularization aims to optimize the network while keeping weights as close to zero as possible. That helps at reducing the capacity of the model and avoids large weights that could produce unstable output behaviors. Moreover, Lasso regression, giving the same importance to all weights, eliminates the weights of the least important features. Therefore, it automatically performs feature selection and produces a sparse model. Sparse models have a lower memory footprint and can achieve better inference performance on dedicated hardware accelerators. It is important to remember that Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

On the other hand, Ridge regression adds the l_2 norm of the weight vector as shown in equation (2.34).

$$C(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + \alpha \sum_{i=1}^{n} w_i^2 \tag{2.34}$$

As Lasso regression, the l_2 contribution forces the learning algorithm to fit the data and keep the model weights as small as possible. Moreover, as before, the α parameter acts as regularization parameter. However, as shown in Figure 2.12, Ridge

regression does not tend to eliminate small weights, but maintains all components. Indeed, increasing α, the global optimum moves left along the dashed yellow line. It is clear how the smaller parameter is firstly pushed to zero for Lasso regression by the optimization procedure. On the other hand, Ridge regression for all values of α maintains also small parameter values.

Finally, it is possible to mix both regressions, weighting the two contributions. Indeed, Elastic Net simply adds the two regressions as shown by the following equation

$$C(\boldsymbol{w}) = \mathcal{L}(\boldsymbol{w}) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2}\alpha \sum_{i=1}^{n} \theta_i^2 \tag{2.35}$$

where the parameter r controls the mix ratio between the two contributions. Therefore, when $r = 0$, Elastic Net is equivalent to Ridge regression, and when $r = 1$, it is equivalent to Lasso regression.

Max-Norm regularization

Another popular technique that affects the shape of the loss function is max-norm regularization. In contrast to Lasso and Ridge regressions, directly constrains the magnitude of the weights under a certain threshold r. Therefore, it imposes to the weights $\boldsymbol{w}$ of the incoming connections, $\| \boldsymbol{w} \|_2 \leq r$. Reducing r increases the amount of regularization and helps reduce overfitting, constraining weights to values with smaller magnitudes.

Part I

Chapter 3

Remote Sensing for Service Robotics

Remote sensing is the process of detecting and monitoring the physical characteristics of an area/object or observing a phenomenon from a distance, using sensors and without making physical contact. A few decades back, it was initially known for acquiring data from spaceborne satellite and airborne platforms equipped with optical and radar sensors [34]. Recently, image and spatial data acquisitions from UAV, UGV, and even edge device sensors are considered as RS methods too. RS sensors are considered a payload even if mounted on edge vehicles, and no processing is performed on-board machines. Different platforms acquire data, and cloud infrastructures are heavily exploited to process and extract valuable information not only for the goal of the specific application but also for navigation and task execution of autonomous agents. Indeed, RS is a helpful tool to support autonomous agents' decisions and enhance their perception. Moreover, in the last decade, RS has been significantly growing in popularity due to the exceptional technological developments brought by the research community [237]. Imaging sensor improvements and the rise in information infrastructure, including processing data, data storage, and communication among the technological devices, are at the foundation of its current expansion. Furthermore, advancements in aerial platforms, UAV, and UAS (Unmanned Aircraft System) have increased the capability of RS methods and broadened up the application areas [189]. For these reasons, RS is an essential source of information for service robotics applications, enabling better productivity and decision-making.

This chapter introduces some fundamentals about RS that will be helpful for the first part of the presented dissertation. Firstly, different remote sensing platforms are introduced and categorized, highlighting positive and negative characteristics for each of them. Then, different types of RS data are analyzed and discussed from the point of view of a subsequent machine learning manipulation.

3.1 Remote sensing platforms

Remote sensing supporting platforms can be divided taking into account four key characteristics that describe their capabilities and performance:

- **Sensor Type**: sensors can generally be divided into two categories considering if they have their source of illumination (active) or measure reflected sunlight (passive). For instance, they belong to active sensors such as Synthetic Aperture Radar (SAR) and LiDAR. On the other hand, Hyperspectral Imaging (HSI), magnetic sensor (MS) belong to the passive group.
- **Spatial Resolution**: it refers to the linear spacing of a measurement or the physical dimension that represents a pixel of the image. It is also known as Ground Sampling Distance (GSD).
- **Temporal Resolution**: it specifies the revisit time of the platform to determine the data acquisition frequency.
- **Object Range**: it is the distance between the sensing platform and the target object.

RS platforms can be broadly divided into long-range (e.g., satellite and airborne vehicles) and short-range (e.g., UAV/UAS and UGV/ground sensors), considering the four main characteristics. Several further operational aspects are associated with the existing remote sensing platforms to be considered before selection. Table 3.1 reports all platforms with the respective operational aspects. Platform selection should consider all pros and cons to best accommodate the necessity of the specific application. For instance, images acquired by satellite and aerial platforms are often severely affected by cloud cover [179]. Instead, edge devices are not affected.

In the rest of the section, all leading platforms for RS are discussed and analyzed.

	RS Long-Range Platforms		**RS Close-Range Platforms**	
Operational Aspects	**Satellite**	**Airborne**	**UAV/UAS**	**UGV Ground sensors**
Observation space	Worldwide	Regional	Local	Local
Sensor diversity	MS/HSI/SAR	MS/HSI/LiDAR/SAR	MS/LiDAR/HIS	MS/LiDAR (HSI)
Maneuverability	No/limited	Moderate	High	Limited
Ground coverage	Large (10 km)	Medium (1 km)	Small (100 m)	Small (50 m)
FOV	Narrow	Wide	Wide/super wide	Wide/super wide
Revisit rate	Day	Hours	Minutes	Minutes
Spatial resolution	0.30–300 m	5–25 cm	1–5 cm	1–5 cm
Spatial accuracy	1–3 m	5–10 cm	1–25 cm	3–50 cm
Deployability	Difficult	Complex	Easy	Moderate
Operational risk	Moderate	High	Low	Moderate
Cost	High	Medium	Low	Low

Table 3.1: Operational aspects and characteristics of different Long and close-range RS Platforms [8].

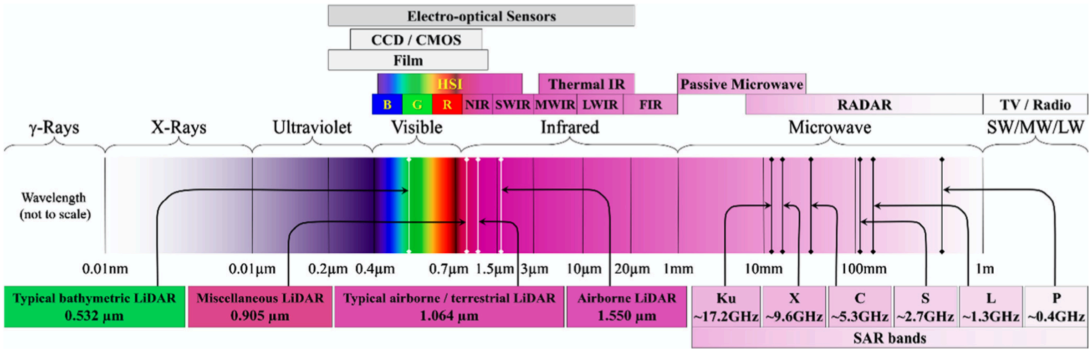


Figure 3.1: Electromagnetic spectrum and related sensors type. Altogether, they cover most of the low energy frequencies [8].

3.1.1 Satellites

Satellites have been the source of information for earth observation programs for the past four decades. The National Aeronautics and Space Administration (NASA) designed and launched the first Landsat-1 mission in 1972. Nowadays, nearly 600 satellites, operated by more than 50 countries, are actively making Earth observations. As shown in Figure 3.1, satellite sensors capture data that span the electromagnetic spectrum from the visible band to low energy microwave waves with SAR. On-board sensors technology continues to progress, improving performance and incorporating better spatial and spectral resolution [196]. Moreover, also multiple sensors on the same platform are gaining significant attention. The monitoring goal and data to be acquired influence the type of sensor and taken orbit. For instance, different orbital paths are followed to achieve continuous monitoring and mapping.

More recent missions, such as Sentinel-1 and Sentinel-1, have started providing free high temporal and spatial resolution images. Moreover, captured images cover a large area that is used to observe the desired phenomena at a global scale [8]. Consequently, several research works take advantage of the high revisit time for diverse applications: land-cover classification [39, 165, 168], vegetation and ocean monitoring [2] and post-earthquake analysis [117].

Costs, low revisit time flexibility, dependency on weather conditions, and atmospheric interference are the main drawbacks and limitations of the satellite platform for RS. Nevertheless, they are considered among the most robust and reliable platforms due to their nature.

3.1.2 Airborne vehicles

Since the introduction of publicly available satellite imagery with 30 cm spatial resolution, the actual difference between spaceborne and airborne is increasingly

Figure 3.2: Graphical representation of fixed-wing (left) and multi-rotor (right) UAVs operating in a crop. They are the most common contributors to the data ecosystem for open-air service robotics applications.

becoming negligible. However, especially with the employment of UAS, airborne platforms are more flexible and customizable for the specific application. Sensors type, flights schedule, and platform speed are all parameters that the user can easily control. The coverage is more limited than satellite platforms but much more significant than UAV ones. Moreover, the LiDAR sensors technology advancement allows to effectively acquire data at a lower speed, offering better point distributions.

3.1.3 Unmanned aerial vehicles

UAVs are among the most agile and dynamic machines ever created [157], and their characteristics can be exploited to perform short-range RS. They are primarily remotely operated by an operator, but researchers and aviation/airspace policies work toward large-scale autonomous flying. Due to the recent technological advancements, UAV systems have been gaining interest and popularity among the RS community as an interesting platform to carry RS payloads. Drones are flexible and budget-effective platforms and are fully customizable for the end application, constituting a valuable alternative to satellite and airborne vehicles.

Drones can be classified into two broad categories: fixed-wing and multi-rotor. Maximum flying speed, flying time, payload capability, and maximum altitude are characteristics that distinguish the two types of UAVs. Nevertheless, multi-rotors

are more agile and easy to control, but they show a lower flying time than fixed-wing aircraft. Figure 3.2 depicts both type of drones in an operational environment. Either RS systems can mount multiple flight control devices such as Global Navigation Satellite System (GNSS) and Inertial Measurement Units (IMU) to enable autonomous flight. However, in most of the countries, commonly used rules restrict flights only to the line of site operations. For instance, the Italian Civil Aviation Authority (ENAC) divides critical and not critical areas. For the latter, flights are limited to an altitude up to 70 m with a maximum radius of 150. Although, rules continue developing, and future regulations could start considering fully autonomous flight, further exploiting the capabilities of these machines.

3.1.4 Unmanned ground vehicles and ground sensors

Proximity sensors are mainly adopted for specific application necessities. The primary purpose is a real-time assessment of the area/object observed and informing the user on phenomenon variations. Thus, extracted data is not usually useful for autonomous navigation of agents and task execution. Nevertheless, their employment help overcome some limitations of other platforms [179]: weather dependence, calibration, atmospheric correction, and georeferencing are only some of the limitations of previously discussed platforms. Therefore, there is an increasing interest in their application. They are placed as payloads on UGVs or as statistics sensors. RGB optical sensors, thermal sensors, and LiDAR are among the most commonly utilized sensors.

3.2 Spectral indices

Nowadays, remote sensing is widely used for hearth monitoring and precision agriculture. Vegetation indices are captured by exploiting electromagnetic wave reflectance. Indeed, reflectance changes with plant type, tissue water content, and other correlated factors. The reflectance from vegetation or emission characteristics of vegetation is determined by chemical and morphological properties of the canopies or leaves [266]. The primary uses for remote sensing of vegetation are based on the following light spectra: (i) the ultraviolet region (UV), ranges from 10

Name	Index	Formula	Reference
Normal Difference Vegetation Index	NDVI	(NIR - R)/(NIR + R)	[123]
Ratio Vegetation Index	RVI	R/NIR	[119]
Difference Vegetation Index	DVI	NIR - R	[204]
Soil-Adjusted Vegetation Index	SAVI	(NIR - R)(1 + L)/(NIR + R + L)	[113]

Table 3.2: Common vegetation indices with related formulas and references.

to 380 nm; (ii) the visible spectra, which are comprised of the blue (450–495 nm), green (495570 nm), and red (R) (620–750 nm) wavelength ranges; and (iii) the near and mid-infrared bands (NIR) (850–1700 nm) [56]. Advancements in high-resolution spectral instrumentation have brought an expansion in the number of bands captured by the remote sensing platforms, which are employed to determine various indices to characterize vegetation status [8]. In Table 3.2 are reported main vegetation indices proposed in the literature. One of the first vegetation indices proposed, Ratio Vegetation Index (RVI) [119], identifies vegetation from artifacts and is based on the principle that leaves absorb relatively more red than the infrared wavelength. The RVI has been used extensively for vegetation monitoring and green biomass estimations. Conversely, Richardson et al. [204] proposed the Difference Vegetation Index (DVI) as the difference between NIR and R. It is used to monitor the vegetation ecological environment due to its sensitivity for change in the background soil. However, DVI does not present any form of normalization, making its interpretation ambiguous for certain applications. On the other hand, Normalized Difference Vegetation Index (NDVI) is by far the more utilized vegetation index; it is the normalized ratio between the red and near-infrared bands [123]. It is beneficial to quickly identify plants' vigor and vegetation status and is very sensitive to slight variations. Nevertheless, background soil, atmosphere, weather, and leaf canopy shadow can affect the vegetation index value. Consequently, to overcome some limitations of RVI and NDVI, Huete [113] introduced the Soil Adjusted Vegetation Index (SAVI). It includes the soil conditioning index, L, which further improves the sensitivity of NDVI and makes it more resilient. Finally, with the growing interest in RS, many other vegetation indices have been proposed for different types of applications [260].

Chapter 4

Data Driven Long-Range Remote Sensing for Robotics

Long-range RS data is valuable information for different service robotics applications. It can be directly used to guide the task execution of autonomous agents or support the short-range RS layer [168, 165]. However, processing algorithms give value to data, and usually, common goals cannot be achieved with traditional programming. Moreover, data hides complex patterns for most applications, and designing meaningful features is difficult and time-consuming. Consequently, DL solutions constitute a promising instrument for analyzing and extracting long-range RS data information. Computational load and latency are not a matter of much concern at this level, and the cloud can be employed to exploit the potential of DL algorithms fully. Data is acquired from satellite and airborne platforms and sent through a communication channel to post-process by a trained model. Subsequently, extracted information can support navigation and decision-making of edge vehicles or feed other long or close-range pipelines.

This chapter discusses two contributions that leverage DL algorithms to distill value from long-range data. In the first case study, multi-temporal satellite imageries are exploited to obtain a high-resolution version of the captured area. Indeed, free-of-cost satellite or airborne data usually have a high temporal resolution at the expense of a lower spatial resolution. Afterward, enhanced acquisitions can be processed by a downstream pipeline to extrapolate occupancy grid maps. On the other hand, the second research project takes advantage of pre-processed high-resolution long-range imagery to automatically compute a global path for autonomous navigation of edge vehicles in row crops fields. A DL model is trained with simulated images to extract a given occupancy grid map points of interest. Subsequently, a post-processing custom global-path planning algorithm generates a viable path for the target autonomous agent. Either contribution is an example of how RS can constitute valuable information for service robotics applications. Other examples of DL algorithms applied to long-range RS data proposed by the author of this dissertation can be found in [168, 165].

4.1 Residual attention deep neural networks for multi-image super resolution

Super-resolution (SR) algorithms serve the purpose of reconstructing higher resolution images from either single or multiple low-resolution (LR) images. Due to constraints such as sensor limitations and exceedingly high acquisition costs, it is often challenging to obtain high-resolution (HR) images. In this regard, SR algorithms provide a viable opportunity to enhance and reconstruct HR images from LR images recorded by the sensors. Over more than three decades, progress has steadily been observed in the development of Super-resolution, as both multi-frame and single-frame SR now have substantial applications that can use the image generation purposefully.

SR is very significant to RS because it provides an opportunity to enhance LR images despite the inherent problems often faced in RS scenarios. The hardware and material costs for smaller missions around data accumulation are very high. Additionally, on-board instruments on satellites continue to generate ever-increasing data as spatial and spectral resolutions also increase, and this has progressively become challenging for compression algorithms [242] as they try to meet the bandwidth restrictions [20].

There are two main methods used in SR: Single-image SR (SISR) and Multi-image SR (MISR). SISR employs a single image to reconstruct an HR version of it. However, a single image is quite limited in the amount of information that it provides, mainly post the LR image formation process. Contrastingly, MISR involves multiple LR images of the same scene acquired from the same or different sensors to construct an HR image. The significant advantage MISR holds over SISR is in how it can draw out otherwise unavailable information from the different image observations of the same scene. It consequently constructs a high spatial resolution image. However, to achieve the additional benefits of MISR, many problems need to be solved. Conventionally, multiple images are obtained by either a satellite during its multiple orbits or by different satellites at different times or different sensors acquiring images simultaneously. Many complications need to be considered with so many variables involved, such as cloud coverage, time variance in scene content, and invariance to absolute brightness variability.

There has been significant progress in Single-image SR as deep learning methods, and deep neural networks have been brought into use, allowing a better efficient generation of non-linear maps to deal with complex degradation models. However, there has not been any similar progress in MISR. In the past few years, many deep learning-based approaches have been exploited to address the MISR problems in the context of enhancing video sequences [121, 32, 118]. However, MISR is rarely exploited for remotely sensed satellite imagery. Kawulok et al. [126] demonstrated the

potential benefits of information fusion offered by multiple satellite images reconstruction and learning-based SISR approaches. In their work, EvoNet framework [127] based on several deep CNNs was adopted to exploit SISR in the preprocessing phase of the input data for MISR.

Recently, a challenge was set by the European Space Agency (ESA) to super-resolved multi-temporal PROBA-V satellite imagery[1]. In this context, a new CNN-based architecture *DeepSUM* was proposed by Molini et al. [177] to super resolve multi-temporal PROBA-V imagery. An end-to-end learning approach was established by exploiting both spatial and temporal correlations. Most recently, Deudon et al. presented *HighRes-net* based on deep learning to deal with the MISR of remotely sensed PROBA-V satellite imagery [62]. They proposed an end-to-end mechanism that learns the sub-tasks involved in MISR: co-registration, fusion, up-sampling, and registration-at-the-loss.

With the presented contribution [214], building over the latest breakthroughs in SISR [64, 264, 269, 58], we propose a deep learning MISR solution for remote-sensing applications that exploits both spatial and temporal correlations to combine multiple low-resolution acquisitions smartly. Indeed, our model provides a real end-to-end efficient solution to recover high-resolution images, overcoming limitations of previous similar methodologies, and providing enhanced reconstruction results. So, the presented research constitutes an exceptional opportunity, easily replicable, to access better quality and more helpful information for the remote-sensing community.

4.1.1 Methodology

MISR aims at recovering an HR image I^{HR} from a set of T LR images $I^{\mathrm{LR}}_{[1,T]}$ of the same scene acquired in a certain temporal window. In contrast to SISR, MISR can simultaneously benefit from spatial and temporal correlations, being able to achieve far better reconstruction results theoretically. Either way, SR is an inherently ill-posed problem since a multiplicity of solutions exist for any given set of low-resolution images. So, it is an underdetermined inverse problem, of which the solution is not unique. Our proposed methodology, based on a representation learning model, aims at generating a super-resolved image I^{SR} applying a function H_{RAMS} to the set of $I^{\mathrm{LR}}_{[1,T]}$ images:

$$I^{\mathrm{SR}} = H_{\mathrm{RAMS}}(I^{\mathrm{LR}}_{[1,T]}, \Theta) \tag{4.1}$$

where Θ are model parameters learned with an iterative optimization process.

In other words, we propose a fully convolutional Residual Attention Multi-image Super-resolution network (RAMS) that can efficiently extract high-level features

[1] https://kelvins.esa.int/proba-v-super-resolution/

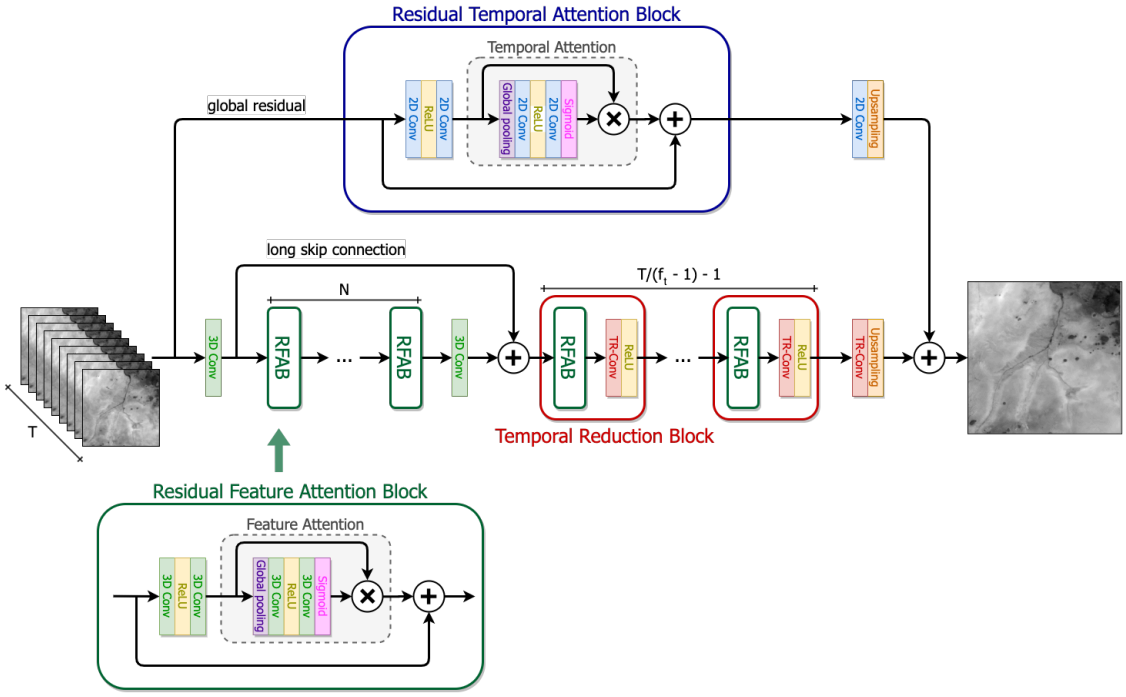


Figure 4.1: Overview of the Residual Attention Multi-image Super-resolution Network (RAMS), assuming to work with single-channel LR images ($C = 1$) to simplify the discussion. A tensor of T single-channel LR images constitutes the input of the proposed model. The main branch extracts features, with 3D convolutions, in a hierarchical fashion, while a feature attention mechanism allows the network to select and focus on most promising inner representations. Concurrently, a global residual path exploits a similar attention operation in order to make an aware fusion of the T distinct LR images. All computations are efficiently performed in the LR feature space and only at the last stage of the model an upsampling operation is performed in both branches.

concurrently from T LR images and fuse them, exploiting a built-in visual attention mechanism. Attention directs the focus of the model only on the most promising extracted features, reducing the importance of less relevant ones and mostly transcending limitations of the local region of convolutional operations. Moreover, extensive use of nested residual connections lets all the redundant low-frequency information, present in the set $I^{\text{LR}}_{[1,T]}$ of LR images, flow through the network, leaving the model focusing its computation only on high-frequency components. Indeed, high-frequency features are more informative for HR reconstruction, while LR images contain abundant low-frequency information that can directly be forwarded to the final output [269]. Finally, as the majority of the model for Single-image Super-resolution [151, 264, 58], all computations in our network are efficiently performed in the LR feature space requiring only an upsample operation at the final stage of the model.

In the following paragraphs, we present the overall architecture of the network with a detailed overview of the main blocks. Finally, we conclude the methodology section with precise details of the optimization process for training the network.

Network architecture

An overview of the RAMS network, with its main three blocks and two branches, is depicted in Figure 4.1. As a high-level description, the model takes as input a single set of T low-resolution images $I^{\text{LR}}_{[1,T]}$ that can be represented as a tensor $\mathbf{X}^{(i)}$ with shape $H \times W \times T \times C$ where H, W and C are the height, width, and channels of the single low-resolution images, respectively. The upper global residual path proposes a simple SR solution, making an aware fusion of the T input images. On the other hand, the central branch exploits 3D convolutions residual-based blocks in order to extract spatial and temporal correlations from the same set of T LR images and provide a refinement to the residual simple SR image.

More in detail, in the first part of the main path of the model, we use a simple 3D convolutional layer, with each filter of size $f_h \times f_w \times f_t$, to extract F shallow features from the input set $I^{\text{LR}}_{[1,T]}$ of LR images. Then, we apply a cascade of N residual feature attention blocks that increasingly extract higher-level features, exploiting local and non-local, temporal, and spatial correlations. Moreover, we make use of a long skip connection for the shallow features and several short skip connections inside each feature attention block to let flow all redundant low-frequency signals and let the network focus on more valuable high-frequency components. The three dimensions H, W and T are always preserved through reflecting padding. The first part of the main branch can be modeled as a single function H_I that maps each tensor $\mathbf{X}^{(i)}$ to a new higher dimensional one $\mathbf{X}^{(i)}_I$ with shape $H \times W \times T \times F$:

$$\mathbf{X}^{(i)}_I = H_I(\mathbf{X}^{(i)}) \tag{4.2}$$

In the second part of the main branch, we further process the output tensor $\mathbf{X}^{(i)}_I$ with $\lfloor T/(f_t - 1) \rfloor - 1$ temporal reduction blocks. In each block, we intersperse a residual feature attention block with 3D convolutional layer without padding on the temporal T dimension (TR-Conv). So, H, W and F remain invariant and only the temporal dimension is reduced. The output of this second block is a new tensor $\mathbf{X}^{(i)}_{II}$ with shape $H \times W \times f_t \times F$, where the temporal dimension T is reduced to f_t:

$$\mathbf{X}^{(i)}_{II} = H_{II}(\mathbf{X}^{(i)}_I) \tag{4.3}$$

Finally, the output tensor $\mathbf{X}^{(i)}_{II}$ is processed by a further TR-Conv layer that reduces T to one and an upscale function $H_{\text{UP}|_{3\text{D}}}$ that generates a tensor $\mathbf{X}^{(i)}_{\text{UP}|_{3\text{D}}}$ of shape $sH \times sW \times C$ where s is the scaling factor. The overall output $\mathbf{X}^{(i)}_{\text{UP}|_{3\text{D}}}$ of the main branch sums with the trivial solution provided by the global residual. Indeed, the global path simply weights the T LR images of the input tensor $\mathbf{X}^{(i)}$ with a

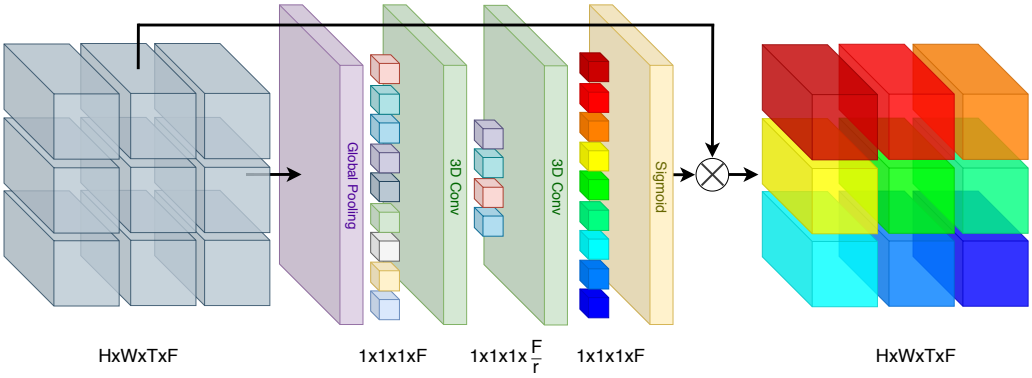


Figure 4.2: Reference architecture of a feature attention block. A series of convolutional operations and non-linear activations are applied to the input tensor with shape $H \times W \times T \times F$ in order to generate different attention statistics for each feature F that concurrently take advantage of local and non-local correlations. Consequently, each tensor's feature is properly re-scaled, enabling the network to focus on most promising components and letting residual connections heed of all redundant low-frequency signals.

residual temporal attention block with filters of size $f_h \times f_w$. Then it produces an output tensor $\mathbf{X}^{(i)}_{\text{UP}|_{2D}}$ of shape $sH \times sW \times C$ that is added to the one of the main branch. So, the final SR prediction of the network $\hat{\mathbf{Y}}^{(i)} = I^{\text{SR}}$ is the sum of the two contribution:

$$\hat{\mathbf{Y}}^{(i)} = H_{\text{RAMS}}(\mathbf{X}^{(i)}) = (\mathbf{X}^{(i)}_{\text{UP}|_{3D}} + \mathbf{X}^{(i)}_{\text{UP}|_{2D}}) \tag{4.4}$$

The upscaling procedure is identical for both branches; after several trials with different methodologies, such as transposed convolutions [62], bi-linear resizing and nearest-neighbor upsampling [184], we adopted a sub-pixel convolution layer as explained in detail in [220]. So, for either branch, the last 2D or 3D convolution generates $s^2 \cdot C$ features in order to produce the final tensors of shape $sH \times sW \times C$ for the residual sum.

In conclusion, the overall model takes as input a tensor $\mathbf{X}^{(i)}$ with shape $H \times W \times T \times C$, works always efficiently in the LR space and generates only at the final stage an output tensor $\hat{\mathbf{Y}}^{(i)}$ with shape $sH \times sW \times C$. In the following paragraphs, the three major functional blocks, residual feature attention, residual temporal attention, and temporal reduction blocks are further explained and analyzed.

Residual attention blocks are at the core of the RAMS model; their specific architecture allows it to focus on relevant high-frequency components and let redundant, low-frequency information flow through the residual connections of the network. Inter-dependencies among features, in the case of feature attention blocks, or temporal steps, in the case of temporal attention blocks, are taken into account computing for each of them, relevant statistics that take into account local and non-local, temporal and spatial correlations. Indeed, either 3D or 2D convolution

filters operate with local receptive fields loosing the possibility to exploit contextual information outside of their limited region of view.

Except for the global residual path, all residual attention blocks are residual feature attention blocks, as shown in Figure 4.1. Indeed, each block of features is weighted up in order to trace most promising high-frequency components, and a residual connection lets low-frequency information flow through the network. More formally, the output of a residual feature attention block with a generic tensor, $\mathbf{X}_n^{(i)}$, is equal to:

$$F_{RFA}(\mathbf{X}_n^{(i)}) = \mathbf{X}_n^{(i)} + H_{FA}(\mathbf{X}_*^{(i)}) \cdot \mathbf{X}_*^{(i)} \tag{4.5}$$

where H_{FA} is the feature attention function and $\mathbf{X}_*^{(i)}$ is the output of two stacked 3D convolutional layers.

$$\mathbf{X}_*^{(i)} = W_2 * max(0, W_1 * \mathbf{X}_n^{(i)} + B_1) + B_2 \tag{4.6}$$

where W_1,W_2 and B_1, B_2 represent the filters with size $f_h \times f_w \times f_t$ and biases respectively and, '$*$' denotes the 3D convolution operation. The number of filters is always equal to F as the ones extracted by the first 3D convolutional layer. Therefore, all low-frequency components in $\mathbf{X}_n^{(i)}$ can flow through the residual connection and H_{FA} can focus the attention of the network to more valuable high-frequency signals. To this end, the feature attention block takes the feature-wise global spatial and temporal information into a feature descriptor by using a global average pooling. Therefore, from the tensor $\mathbf{X}_*^{(i)}$ with shape $H \times W \times T \times F$ we extract $z_F \in \mathbb{R}^F$ feature statistics using the following equation:

$$z_F = \frac{1}{H \times W \times T} \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{k=1}^{T} \mathbf{X}_*^{(i)}(i, j, k) \tag{4.7}$$

Statistics of the feature z_F can be viewed as a collection of descriptors, whose values contribute to express the whole stack of temporal images [106]. In Figure 4.2, it is possible to observe the global pooling operation which output is a tensor $\mathbf{Z}_F^{(i)}$ with shape $1 \times 1 \times 1 \times F$ and last dimension equal to z_F. In addition, the output tensor $\mathbf{Z}_F^{(i)}$ is further processed by a stack of two 3D convolutional layers with a ReLU [181] and sigmoid activation function, respectively. Indeed, as discussed in [106], the stack of two convolutional layers with the filter of size $1 \times 1 \times 1$ allows creating a non-linear mapping function which is able to deeply capture feature-wise dependencies from the aggregated information extracted by the global pooling operation. The first 3D convolutional layer reduces the feature size by a factor of r, and then the second layer restores the original dimension and constraints its values from zero to one with a sigmoid function in a non-mutually exclusive relationship. Finally, the original tensor $\mathbf{X}_*^{(i)}$ is weighted up by the processed attention statistics as shown in Equation (4.5).

The primary purpose of the global residual path is to generate a starting trivial solution for the upsampling problem. More accurate is this starting prediction,

and more simplified is the role of the main branch of the network, leading to a lower reconstruction error. However, the input of the model $\mathbf{X}^{(i)}$ has T different LR images that have to be merged. Intuitively, for each input sample $I^{\text{LR}}_{[1,T]}$, there are some LR images more similar to each other. Therefore, giving them more relevance when merging the T LR images would most probably lead to higher quality predictions. In this context, the aim of the residual temporal attention block is to make an aware weighing of the different input temporal images, letting the network to make an upsample solution with primarily the most similar temporal steps. That is accomplished with an asymmetrical mechanism to the one employed in the residual feature attention blocks and can be summarized by the following formula:

$$F_{RTA}(\mathbf{X}^{(i)}) = \mathbf{X}^{(i)} + H_{TA}(\mathbf{X}^{(i)}_{*}) \cdot \mathbf{X}^{(i)}_{*} \tag{4.8}$$

where H_{TA} is the temporal attention function and $\mathbf{X}^{(i)}_{*}$ is the product of a stack of two 2D convolutional operations as depicted in Figure 4.3 with $f_h \times f_w$ and $T \cdot C$ as filter size and number of features, respectively. Then, as already introduced with the feature attention blocks, the temporal block takes the temporal-wise global spatial information into a feature descriptor by using a global average pooling operation. Finally, those statistical descriptors are processed by a stack of 2D convolutional layers with ReLU and sigmoid as activation function, respectively, scaling the $T \cdot C$ channels of the input tensor, as shown in Equation (4.8). As for feature attention blocks, the first convolutional layer reduces the number of the last dimension by a factor of r, giving the network the possibility to fully capture temporal-wise dependencies from the aggregated output information of the global average pooling operation.

The aim of the last block of the main branch is to slowly reduce the number of temporal steps so that the temporal depth eventually reduces to one. Indeed, the output tensor $X^{(i)}_{I}$ of the N residual feature attention blocks has T temporal dimensions that need to be merged. To this end, we further process the incoming tensors with $\lfloor T/(f_t - 1) \rfloor - 1$ temporal reduction blocks. Each one is composed

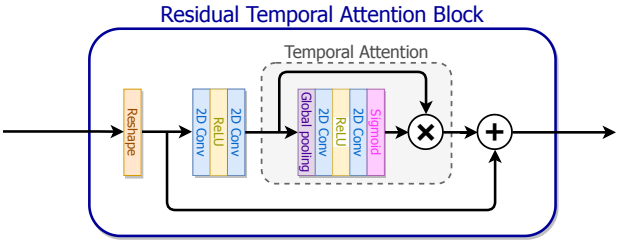


Figure 4.3: Reference architecture of a residual temporal attention block. If the number of channels $C \neq 1$ the input tensor $\mathbf{X}^{(i)}$ is reshaped in $H \times W \times (T \cdot C)$. Consequently, all temporal channels are weighted with some statistics computed by the layers of the temporal attention block.

of a residual feature attention block and a 3D convolutional layer without any reflecting padding in the temporal dimension, denoted TR-Conv. Therefore, at each TR-Conv layer we reduce T of $f_t - 1$. The attention blocks allow the network to learn the best space to decouple image features, "highlighting" more promising features to maintain when reducing the temporal dimension. The output of the last temporal reduction block is a tensor $X_{II}^{(i)}$ with shape $H \times W \times f_t \times F$ where the temporal dimension T is reduced to f_t. The last TR-Conv, before the upsampling function $H_{\mathrm{UP}|_{3D}}$, reduces to one the number of temporal steps and generates $s^2 \cdot C$ features for the sub-pixel convolutional layer.

Training process

Learning the end-to-end mapping function H_{RAMS} requires the estimation of model parameters Θ. That is achieved by minimizing a loss $\mathcal{L}$ between the reconstructed super-resolved images I^{SR} and the corresponding ground truth high-resolution images I^{HR}.

Several loss functions have been proposed and investigated for the SISR problem, such as L_1 [139, 264], L_2 [177, 64] and perceptual and adversarial losses [144]. However, in typical MISR remote-sensing problems, LR images are taken within a certain time window, and they could have an undefined spatial misalignment one to each other. So, we must take into account that the super-resolved output of the model I^{SR} will be inherently not registered with the target image I^{HR}. Moreover, since we can have very different conditions among the images part of the same scene, it is important to make the loss function independent from possible intensity biases between the super-resolved I^{SR} and the target I^{HR}. Indeed, if we get a super-resolved image $I^{\mathrm{SR}} = I^{\mathrm{HR}} + \epsilon$, with ϵ constant and low enough to avoid numerical saturation, we can consider its reconstruction perfect, since it represents the scene with the same level of detail of the ground truth.

With these premises, inspired by the metric proposed in [164], we defined $I^{\mathrm{SR}}_{\mathrm{crop}}$ as the super-resolved output cropped of d pixels on each border and we consider each possible patch $I^{\mathrm{HR}}_{u,v}$, $u, v \in [0{,}2d]$ of size $(sH - 2d) \times (sW - 2d)$ extracted from the ground truth I^{HR}. We compute the mean biases between the cropped $I^{\mathrm{SR}}_{\mathrm{crop}}$ and the patches $I^{\mathrm{HR}}_{u,v}$ as follows:

$$\mathrm{b}_{u,v} = \frac{\sum_{i=1}^{sH-2d} \sum_{j=1}^{sW-2d} \left[I^{\mathrm{HR}}_{u,v} - I^{\mathrm{SR}}_{u,v} \right](i,j)}{(sH - 2d)(sW - 2d)} \tag{4.9}$$

The loss is then defined as the minimum mean absolute error (L_1 loss) between $I^{\mathrm{SR}}_{\mathrm{crop}}$ and each possible alignment patch $I^{\mathrm{HR}}_{u,v}$. We use the MAE instead of the most used MSE since we experimentally find that provides better results for image restoration problems, as proved by the previous works[151, 269].

$$\mathcal{L} = \min_{u,v \in [0,2d]} \frac{\| I^{\mathrm{HR}}_{u,v} - (I^{\mathrm{SR}}_{u,v} + b_{u,v}) \|_1}{(sH - 2d)(sW - 2d)} \tag{4.10}$$

where $\|\cdot\|_1$ represents the L_1 norm of a matrix, i.e. the sum of its absolute values.

4.1.2 Experiments and results

In this section, we test the proposed methodology in an experimental context, training it on the Proba-V dataset [164] of real-world satellite images. It comprises 415 training scenes and 176 for validation for the RED band and 396 for training, and 170 for validation for NIR. Each scene is composed of several LR images (from 9 to 35, depending on the scene) with a dimension of 128x128 pixels and a single HR ground truth with a dimension of 384x384 pixels. The images are encoded as 16-bit png files, even if the actual signal bit-depth is 14 bits. Finally, each image features a binary mask that distinguishes reliable pixels from unreliable ones (e.g., due to cloud coverage).

We evaluate RAMS performance in comparison with other approaches, including a state-of-the-art SISR algorithm, to demonstrate the superiority of Multi-image models. To implement our network, we use the TensorFlow framework. The complete code with a pre-trained version of our model is available online [2].

Data pre-processing

Before training the model, we pre-process the dataset with the following steps:

- register each LR image using as reference the one with maximum clearance c
- select the clearest T images from each scene that are above a certain clearance threshold c_{min}
- pre-augment the training dataset with n_p temporal permutations of the LR input images
- normalize the images by subtracting the dataset mean intensity value and dividing by the standard deviation

For each scene of the dataset, we consider a reference image the one with the maximum clearance c. We consider translation as a transformation model during the registration process, which computes the necessary shifts to register each image for both axes. The registration is performed in the Fourier domain using normalized cross-correlation as in [186]. Furthermore, we set a threshold $c_{\text{min}} = 0.85$ on the clearance for an image to be accepted to avoid using awful images that can worsen the SR performance. The acceptable images are then sorted in order of clearance, and the best T are selected. Since each scene of the dataset contains at least 9

[2] https://github.com/EscVM/RAMS

LR images, we set $T = 9$ to fully exploit all the available information for most of the scenes. One of the Proba-V dataset characteristics is that the LR images of a particular scene have no clear temporal order. Therefore, there is no reason to prefer a specific order in the T input images to another. Therefore, the training dataset is pre-augmented by performing n_p random temporal permutations of the selected T input images to help generalization. In this way, we can train the algorithm to independently identify the best temporal image on the input tensor position. We set this permutation parameter to $n_p = 7$, reaching a total of 2905 training data points for RED and 2751 for NIR. Finally, each image is normalized by subtracting the mean pixel intensity value computed on the entire dataset and dividing by the standard deviation. After the forward pass in the network, all the tensors are then denormalized, and the subsequent evaluations are performed on the 16 bits unsigned integer arrays.

Experimental settings

The scaling factor of the Proba-V dataset is $s = 3$. Since we have different scenes for RED and NIR data, we treat the problem for the two bands separately. For this reason, we have $C = 1$, since we consider images with a single channel. We set $F = 32$ and $f_h = f_w = f_t = 3$ as number of filters and kernel size respectively for each convolutional layer. Therefore, the number of temporal reduction blocks is $\lfloor T/(f_t - 1) \rfloor - 1 = 3$, since each block reduces the temporal dimension of 2. In all the residual attention blocks, we set $r = 8$ as the reduction factor. After testing different values with a grid search, we set $N = 12$ as the number of residual feature attention blocks in the main branch of the network. We find that decreasing this number causes a loss of performance while increasing it gives a little improvement in the results at the cost of a high increase in the number of parameters. $N = 12$ is, therefore, the best compromise between network size and prediction results. In total, our model has slightly less than 1M parameters.

In most of the SR applications present in literature, LR images are obtained from the artificial degradation of the target HR images. In contrast, the real-world nature of the dataset, in which LR images are obtained independently from HR images, causes an unavoidable misalignment between the super-resolved output and the ground truth. To take into account this problem, the authors of the dataset consider a maximum shift of ± 3 pixels on each axis between I^{SR} and target I^{HR}, computed on the basis of the geolocation accuracy of the Proba-V satellite [164]. When computing the loss function, we can therefore set $d = 3$. Besides, since the Proba-V dataset also provides binary mask that marks with one reliable pixel and with 0 unreliable (e.g., concealed by clouds) ones, we adapt the loss function to use this information to refine the training process. During the loss computation, we want pixels marked as unreliable in the target binary mask M^{HR} not to influence the loss computation. Practically, we can simply multiply the cropped super-resolved

image $I^{\mathrm{SR}}_{\mathrm{crop}}$, and the HR patch $I^{\mathrm{HR}}_{u,v}$ by the correspondent cropped mask $M^{\mathrm{HR}}_{u,v}$ and average all the quantities over the number of clear pixels. The bias computation is therefore adapted from equation (4.9) as:

$$\mathrm{b}_{u,v} = \frac{\sum_{i,j}\left[I^{\mathrm{HR}}_{u,v} \cdot M^{\mathrm{HR}}_{u,v} - I^{\mathrm{SR}}_{u,v} \cdot M^{\mathrm{HR}}_{u,v}\right](i,j)}{\|M^{\mathrm{HR}}_{u,v}\|_1} \tag{4.11}$$

where $\|\cdot\|_1$ represents the L_1 norm of a matrix, i.e. the sum of its absolute values. In the same way, the loss is adapted from equation (4.10) as:

$$\mathcal{L} = \min_{u,v\in[0,6]} \frac{\|I^{\mathrm{HR}}_{u,v} \cdot M^{\mathrm{HR}}_{u,v} - (I^{\mathrm{SR}}_{u,v} \cdot M^{\mathrm{HR}}_{u,v} + b_{u,v})\|_1}{\|M^{\mathrm{HR}}_{u,v}\|_1} \tag{4.12}$$

To train the model, we extract from each LR image 16 patches with a size of 32×32 pixels and the corresponding HR and masks patches with a size of 96×96. We further check every single patch and remove those that have a target mask M^{HR} with less than 0.85 clearance. The total number of training data points obtained is 41678 for RED and 40173 for NIR. During the training process, we further perform data augmentation with random rotations of 90°, 180° and 270° and random horizontal flips.

We set the batch size to 32. Therefore, during training, we have an input tensor with shape $32 \times 32 \times 32 \times 9 \times 1$ and an output tensor with shape $32 \times 96 \times 96 \times 1$. We optimize the loss function with Adam algorithm [133] with default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. We set an initial learning rate $\eta_{\mathrm{i}} = 5 \times 10^{-4}$ and we reduce it with a linear decay down to $\eta_{\mathrm{f}} = 5 \times 10^{-7}$. We train two different networks for RED and NIR spectral bands on a workstation with an Nvidia RTX 2080Ti GPU with 11GB of memory and 64GB of DDR4 SDRAM. We use the TensorFlow 2.0 framework with CUDA 10. In total, we train the models for 100 training epochs for about 16 hours.

Temporal Self-Ensemble (RAMS+)

As explained in previous section, during the training process images are augmented with random permutation in the temporal axis. For this reason, it is possible to maximize the performance of the model, by adopting a self-ensemble mechanism during inference, similarly to what done in previous super-resolution works [151, 235, 269]. For each input scene, we consider a certain number P of random permutations on the temporal axis and we denote as $\left\{I^{\mathrm{LR}}_{[1,T],0}, \cdots, I^{\mathrm{LR}}_{[1,T],P}\right\}$ the resulting set of inputs. The output of the inference process is therefore the average of the predictions on the whole set. We call this methodology RAMS+$_P$, where P is the number of random permutations performed:

$$I^{\mathrm{SR}} = \frac{1}{P}\sum_{i=1}^{P} H_{\mathrm{RAMS}}\left(I^{\mathrm{LR}}_{[1,T],i}\right) \tag{4.13}$$

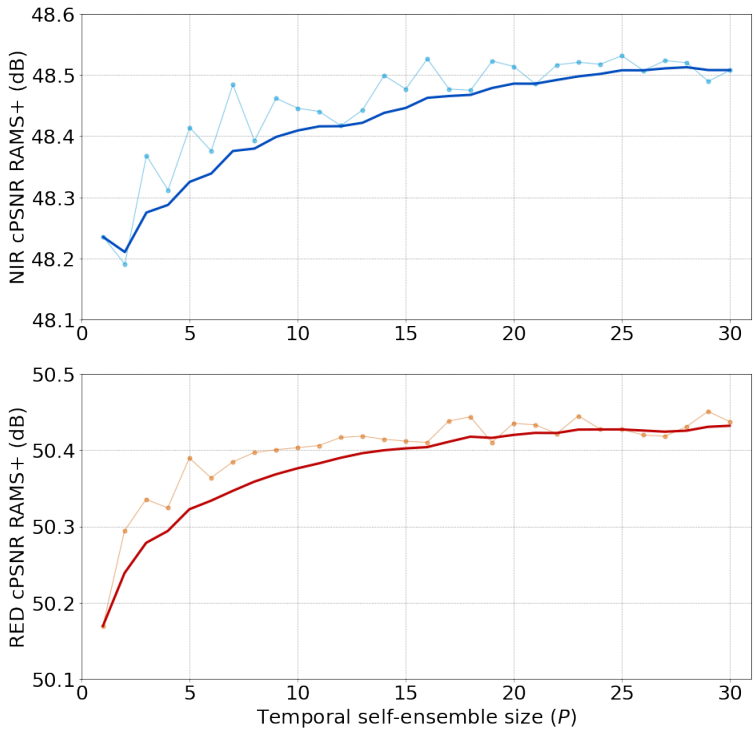


Figure 4.4: Results with a temporal self-ensemble of size P. The highlighted curves represent an exponential moving average of the results to clearly show the trend. The values for $P = 1$ are equivalent to RAMS.

Figure 4.4 shows cPSNR results on the testing dataset for a different number of permutated predictions. The trend clearly shows how increasing P results in better performance on both the spectral bands, with an effect that tends to saturate for $P \geq 25$. For the following evaluation, we select $P = 20$ to present the results for RAMS+. It is worth noting that even if this method allows increasing the performance of the network sharply, inference time grows linearly with P, with RAMS+$_{20}$ taking roughly 20 times as long as RAMS. Another aspect to highlight is that the permutations are performed randomly, so different results can be achieved even with the same value of P.

Quantitative results

To evaluate the obtained results, we need to use a slightly modified version of PSNR and SSIM [252] criteria to take into consideration all the aspects we considered in the previous section to obtain a proper loss function. Martens et al. [164] propose a corrected version of the PSNR, called cPSNS, that is obtained from a corrected mean squared error (cMSE). The computation of the cMSE is performed in the same way as we did for the loss in equation (4.12): it is the minimum MSE

between $I^{\text{SR}}_{\text{crop}} + b_{u,v}$ and the HR patches $I^{\text{HR}}_{u,v}$:

$$\text{cMSE} = \min_{u,v\in[0,6]} \underset{\text{clear}}{\text{MSE}}\Big(I^{\text{HR}}_{u,v}, I^{\text{SR}}_{\text{crop}} + b_{u,v}\Big) \tag{4.14}$$

where $\underset{\text{clear}}{\text{MSE}}$ represents the mean squared error computed only on pixels marked as clear in the binary mask $M^{\text{HR}}_{u,v}$. Again, we can simply multiply the matrices by the mask to make unreliable pixels irrelevant:

$$\underset{\text{clear}}{\text{MSE}} = \frac{\|I^{\text{HR}}_{u,v} \cdot M^{\text{HR}}_{u,v} - (I^{\text{SR}}_{u,v} \cdot M^{\text{HR}}_{u,v} + b_{u,v})\|_2^2}{\|M^{\text{HR}}_{u,v}\|_1} \tag{4.15}$$

where $\|\cdot\|_2$ represents the Frobenius (L_2) norm of a matrix, i.e. the square root of the sum of its squared values. We can then compute the cPSNR as:

$$\begin{aligned} \text{cPSNR} &= 10\log_{10}\frac{(2^{16}-1)^2}{\text{cMSE}} \\ &= \max_{u,v\in[0,6]} 10\log_{10}\frac{(2^{16}-1)^2}{\underset{\text{clear}}{\text{MSE}}(I^{\text{HR}}_{u,v}, I^{\text{SR}}_{\text{crop}} + b_{u,v})} \end{aligned} \tag{4.16}$$

where $2^{16}-1$ is the maximum pixel intensity for an image encoded on 16 bits.

In the same way, we can define a corrected version of the SSIM metric: cSSIM is the maximum SSIM between $I^{\text{SR}}_{\text{crop}} + b_{u,v}$ and the HR patches $I^{\text{HR}}_{u,v}$, again multiplied for the mask $M^{\text{HR}}_{u,v}$.

$$\text{cSSIM} = \max_{u,v\in[0,6]} \text{SSIM}\Big(I^{\text{HR}}_{u,v} \cdot M^{\text{HR}}_{u,v}, I^{\text{SR}}_{\text{crop}} \cdot M^{\text{HR}}_{u,v} + b_{u,v}\Big) \tag{4.17}$$

Table 4.1 shows the comparison of cPSNR and cSSIM metrics with several methods on the validation set. We consider as the baseline the bicubic interpolation

Band	**NIR**		**RED**	
Metric	**cPSNR**	**cSSIM**	**cPSNR**	**cSSIM**
Bicubic	45.12	0.9767	47.63	0.9846
IBP[115]	45.96	0.9796	48.21	0.9865
BTV[76]	45.93	0.9794	48.12	0.9861
RCAN[269]	45.66	0.9798	48.22	0.9870
VSR-DUF[118]	47.20	0.9850	49.59	0.9902
HighRes-net[62]	47.55	0.9855	49.75	0.9904
DeepSUM[177]	47.84	0.9858	50.00	0.9908
DeepSUM++[178]	47.93	0.9862	50.08	0.9912
RAMS (ours)	**48.23**	**0.9875**	**50.17**	**0.9913**
RAMS+$_{20}$ (ours)	**48.51**	**0.9880**	**50.44**	**0.9917**

Table 4.1: Average cPSNR (dB) and cSSIM over the validation dataset for different methods.

of the best image of the scene selected considering the clearance, i.e., the number of clear pixels as marked by the binary masks.

IBP[115] and BTV[76] methods are tested with the same methodology presented in Molini et al. [177]. They achieve slightly better results than the baseline with both the metrics.

RCAN [269] is currently one of the Single-image Super-resolution state-of-the-art networks. We trained from scratch two models, one for each spectral band, setting $G = 5$ and $B = 5$, as the number of residual groups and residual channel attention blocks, respectively, for a total of about 2 million parameters. We train the two models from scratch on the Proba-V dataset, selecting the best image per scene as input. RCAN shows better performance concerning classical methods but is far beyond the other MISR networks, showing how the additional information coming from both spatial and temporal correlations is vital to boost the super-resolution process.

VSR-DUF[118] has been developed to upsample video signals using a temporal window of several frames. We train two models from scratch, one for each spectral band, using our methodology's 9 LR images as input. The authors consider three different architectures depending on the number of convolutional layers and find better results, increasing the depth of the model. We select the baseline 16 layers deep architecture that already has more than double parameters with respect to RAMS, with the same number of input images.

HighRes-net[62] algorithm got second place in the Proba-V challenge and featured a single network for both spectral bands that recursively reduce the temporal dimension to fuse the input LR images. We train the model on our training dataset with default architectures. Since the authors designed the architecture to have an input temporal dimension multiple of 2, we set it to 16, as it is closest to 9.

DeepSUM [177] is the algorithm winner of the original Proba-V challenge, and the authors have further developed it with DeepSUM++[178]. We train our RAMS on the same training dataset as these two works.

Results clearly show how the proposed methodology can obtain the best results with the two metrics on both the spectral bands and thus represents the current state-of-the-art for Multi-image Super-resolution for RS applications. Using temporal self-ensemble, RAMS+ is able to achieve even higher performance. We show the value for RAMS+, setting $P = 20$ as the size of the ensemble, which is the value at which we experimentally find that the resulting gain starts to saturate. However, further increasing the ensemble size can result in even better performance, though at the cost of a significant inference speed drop.

It is worth mentioning that our methodology reaches a result of 0.93367908 on the test set of the Proba-V challenge as provided by the official site and places at

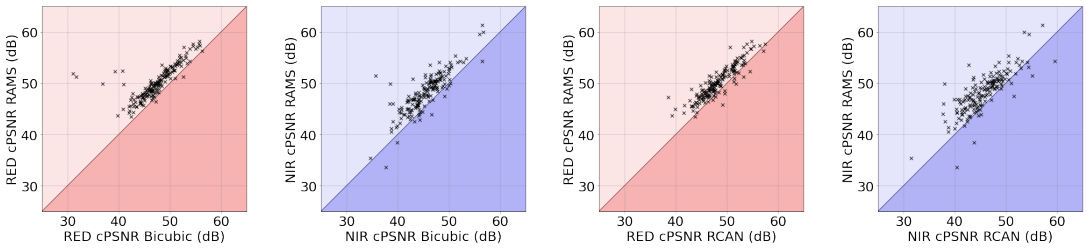


Figure 4.5: cPSNR comparison between RAMS and bicubic inteprolation and RAMS and RCAN(SISR) on the validation set. Each data point represents a scene of the dataset: when a cross is above the line, the correspondent scene is reconstructed better by RAMS.

the top of the leaderboard available after the end of the official challenge[3]. This score is computed as the mean ratio between the cPSNR values of the challenge baseline on each testing scene, and the correspondent submitted cPSNR for both the spectral bands. This result has been obtained by retraining the two networks with both training and validation datasets together.

Figure 4.5 shows a direct comparison between the cPSNR results of RAMS and the bicubic interpolation baseline and RCAN (SISR state-of-the-art). Each cross represents a scene of the validation dataset of the corresponding spectral band. The graphs on the left show how our method strongly beats the bicubic upsampling on almost all the scenes, 98% for RED and 91% for NIR. That is coherent with a general worse behavior of all the methods on the NIR images, probably due to an intrinsic worse information quality of the NIR dataset. The graphs on the right show, on the other hand, the enormous potential of MISR with respect to SISR methods. It can be observed how again RAMS outperforms RCAN an almost all the scenes, with results only slightly worse than to bicubic interpolation, 92% for RED, and 91% for NIR. That is reasonable since RCAN results are someway in the middle between bicubic and RAMS.

[3] https://kelvins.esa.int/proba-v-super-resolution-post-mortem/leaderboard/

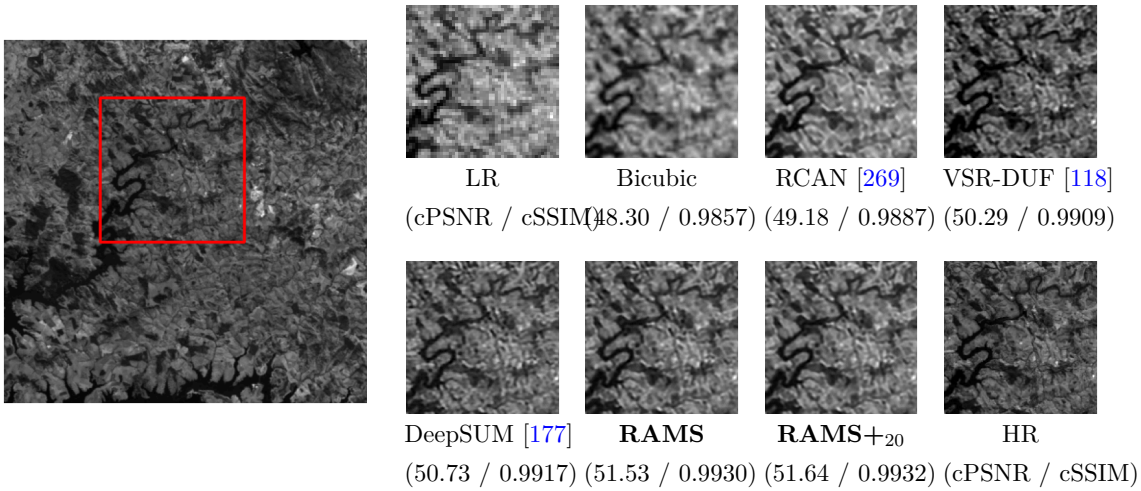


Figure 4.6: Qualitative comparison between different methods on RED imgset0302.

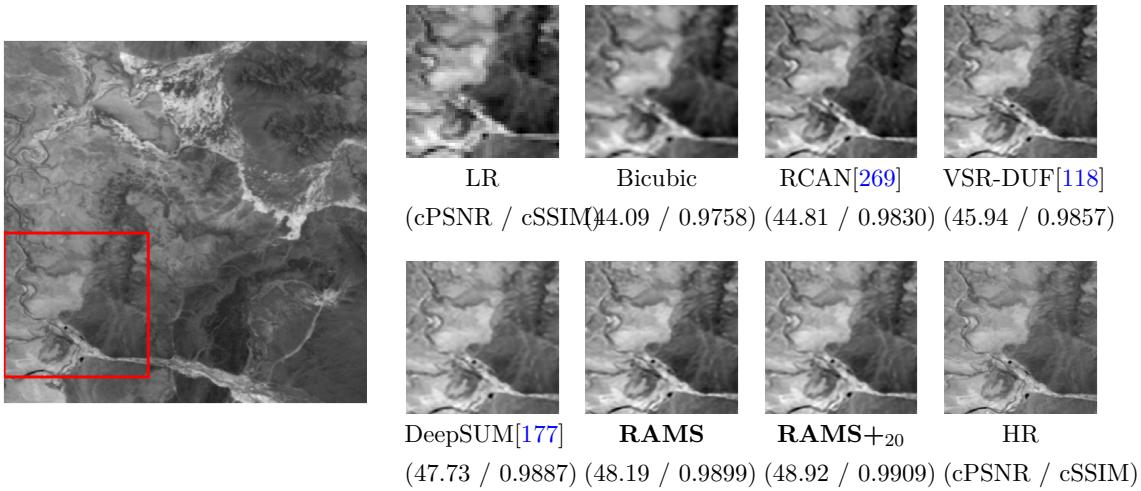


Figure 4.7: Qualitative comparison between different methods on NIR imgset0596.

A visual comparison between some of the methods taken in the exam is shown in Figures 4.6 and 4.7 for a RED and NIR image respectively. We provide a zoomed patch of the best LR input image of the scene, its bicubic interpolation, and the inference output of RCAN, VSR-DUF, DeepSUM, RAMS and RAMS+$_{20}$, together with the target HR ground truth. cPSNR and cSSIM scores for the image under analysis are also provided. From this comparison, MISR methods clearly show a better performance with respect to bicubic and SISR (RCAN). However, it is not trivial to understand which method is the better among MISR algorithms with a visual inspection of the results, only. As found by Ledig et al. [144], the task of achieving pleasant-looking results is a different optimization problem

from maximizing the fidelity of the reconstructed information. Therefore, results with high content-related metrics as PSNR and SSIM frequently appear less photo-realistic to a human eye. However, in the context of remote sensing, the fidelity of the pixels content is vital to ensure that the super-resolved image are meaningful, thus the quality of results should be inferred by using content-related metrics, rather than by visual inspection.

4.2 DeepWay: a deep learning waypoint estimator for global path generation

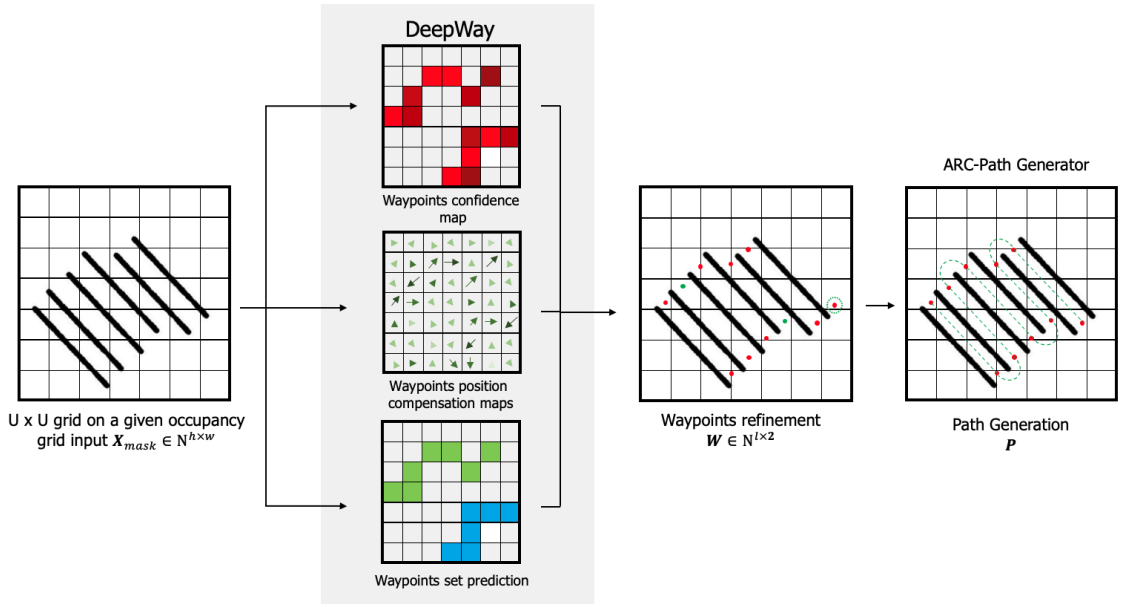


Figure 4.8: Given an occupancy grid of the analyzed crop $\boldsymbol{X}_{mask}$, DeepWay estimates the global waypoints, $\hat{\boldsymbol{Y}}^{(i)}$, and the set they belong to, directly from the full input image. Subsequently, a waypoint refinement algorithm post-processes the prediction of the network, taking care of possible missing and misplaced waypoints (green dots and dashed circles). Finally, a global path generator produces a global path plan $\boldsymbol{P}$ ensuring the full coverage of the crop and the centrality with respect to rows.

Agriculture 4.0 brought a new concept of agriculture based on the introduction of robotics, artificial intelligence, and automation into the agricultural processes to increase production efficiency and cut labor costs. In this regard, self-driving agricultural machinery plays a relevant role in production efficiency by providing a 24/7 weather-independent working production system and cost-cutting since there is no need for a paid driver when performing the required task anymore. A good path generator is crucial for obtaining high autonomous navigation performance. However, the global path generation automation problem has been a bit neglected by the research community. Nevertheless, the most common solutions for this task are based on clustering techniques applied on satellite images or aerial footage taken from the drones. For instance, in [273], authors use clustering in order to detect the rows of the vineyards from UAV images, and then the trajectory is computed by exploiting the information given by the clusters. As shown in [248], extrapolating information regarding the row crops from the images is complex and computational

heavy, and even though there are other solutions besides clustering such as [53], the complete pipeline for obtaining a global path is still tricky and time-consuming due to this necessity of information regarding the crops position and orientation.

In this regard, we introduce DeepWay [171], a novel deep learning approach for global path generation of row-based crop environments. As input, it requires just an occupancy grid of the analyzed parcel and provides, as output, a trajectory able to cover each row of the considered crop, avoiding unwanted collisions with fixed obstacles. The deep neural network is trained on a carefully devised synthetic dataset and is designed to predict global path waypoints directly from the binary mask of crops. The network achieves zero-shot generalization. We train on randomly generated occupancy grid synthetic maps and then directly deploy the policy with real data without any adaptation or fine-tuning. Successively, output waypoints are processed with a refinement pipeline to remove spare waypoints and add missing ones. Finally, the global path is computed through the A*, RRT* search algorithms or with ARC-PG (Adaptive Row Crops Path Generator) [41], a custom global path planner designed explicitly for the analyzed problem that produces more application coherent paths in a smaller amount of time. Extensive experimentation with the synthetic dataset and real satellite-derived images of different scenarios are used to validate the proposed methodology. All of our training and testing code and data are open source and publicly available [4].

4.2.1 Methodology

Given an occupancy grid map of the row crop taken into account, we frame the end rows waypoint detection as a regression problem, estimating positions of the different points with a fully convolutional deep neural network. So, a single neural network, DeepWay, predicts the global path waypoints directly from the full input image, straight from image pixels to points in one evaluation. Since the whole detection pipeline is a single model, it can be optimized end-to-end directly on waypoints estimation. Prior works on global path generation for row-based crops heavily rely on local geometric rules and hardcoded processes that struggle to scale and generalize to the variability of possible real scenarios. On the other hand, DeepWay learns to simultaneously predict all waypoints of the input crop's grid map with their corrections and membership set using features coming from the entire image. It trains on full occupancy grid maps optimizing directly waypoints estimation performance and reasoning globally about the input data. Finally, a post-processing waypoint refinement and ordering algorithm is used to correct missing points misplaced detections and order them before the last global path generation.

[4] https://github.com/fsalv/DeepWay

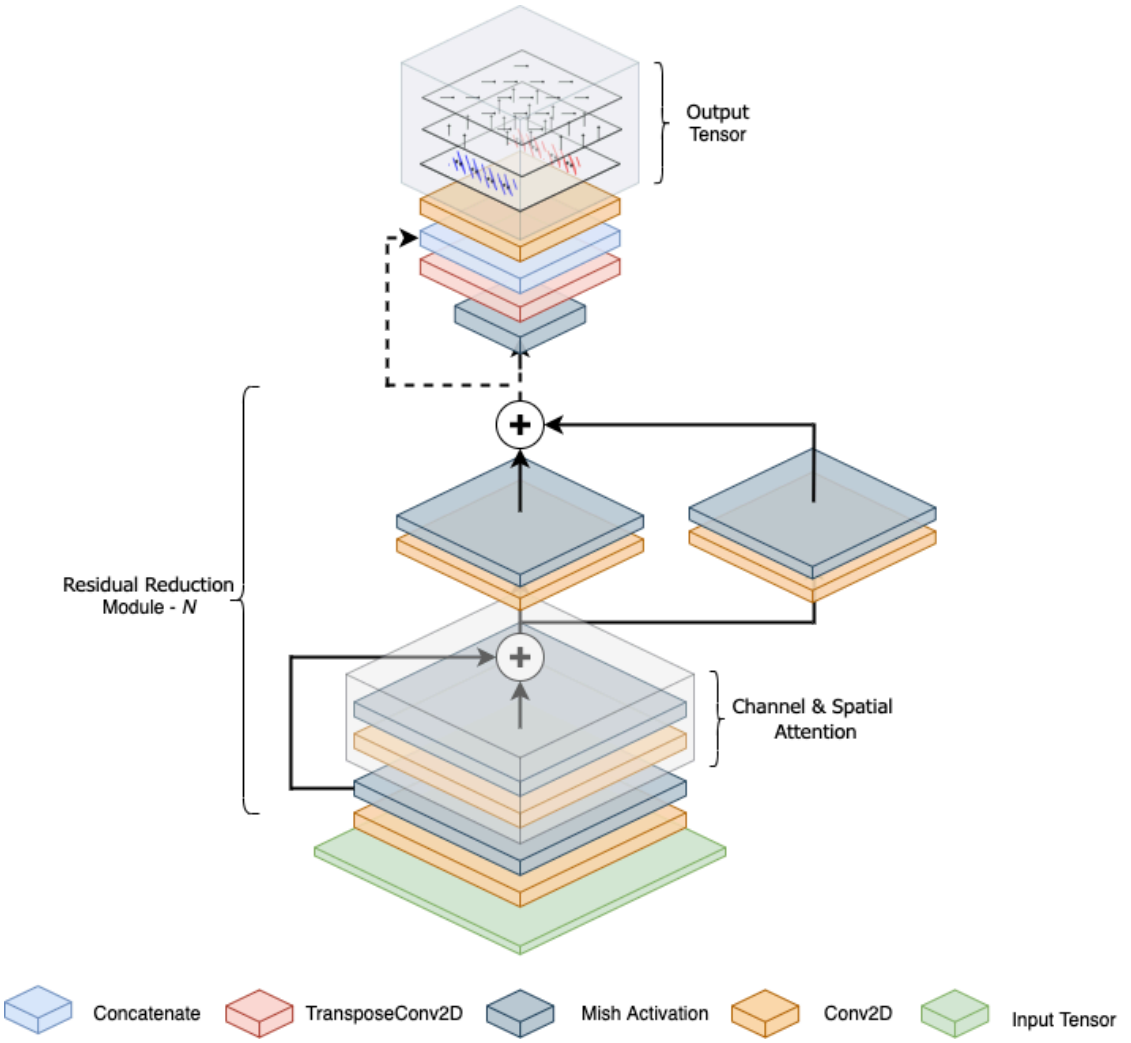


Figure 4.9: Overview of the DeepWay architecture. The model takes a tensor $\boldsymbol{X}^{(i)}$ as input and reduces its spatial dimension with a stack of N residual reduction modules. The synergy of the channel and spatial attention layers lets the network focus on more promising and relevant features. Finally, the neural network outputs a tensor $\boldsymbol{Y}^{(i)}$ of dimension $U \times U \times d$ that for each cell u encodes, waypoint confidence probability $P(u)$, position compensation couple (Δ_x, Δ_y) and membership set.

Our methodology divides the input image of dimension $H \times W$ into a $U_h \times U_w$ grid, and if the center of an end row waypoint falls into a grid cell, that cell is responsible for detecting that point. Each cell, u, predicts a vector with d dimensions, $\boldsymbol{s}_u \in \mathbb{R}^d$, where the L2 norm of the post activated token, $\| \boldsymbol{a}_u \|_2$,encodes the confidence probability $P(u)$. It reflects how confident is the network that a waypoint is placed inside its boundaries. If no waypoint is present, the confidence score should tend to zero. Otherwise, we want to be more close to one as possible. Moreover,

each grid cell, with two dimensions of $\boldsymbol{s}_u$, predicts a position compensation couple (Δ_x, Δ_y) that, if necessary, moves the predicted points from the centre of the cell. Finally, some dimensions are devoted to form an embedding $\boldsymbol{e}_u$ that encodes the membership set, where $\boldsymbol{e}_u \in \mathbb{R}^{(d-2)}$. Indeed, directly making the network estimate to which side a waypoint belongs largely simplifies the post-processing phase.

In Figure 4.8 is presented a high-level overview of the operation principle of the methodology. DeepWay, given an occupancy grid input, predicts a tensor $\hat{\mathbf{Y}}^{(i)}$ of dimension $U \times U \times d$. For each cell u encodes, waypoint confidence probability $P(u)$ as the L2 norm of the post activated vector $\boldsymbol{a}_u$, membership set, and position compensation couple (Δ_x, Δ_y). Indeed, the inference grid $U_h \times U_w$ is k times smaller than the original dimensions, H and W, of the input. So, each u cell contains $k \times k$ original pixels. Without an explicit position compensation mechanism, the network would not be able to adjust the position of a waypoint detection, unable to place it in the correct position of the original input space dimension. As depicted in the right side of Figure 4.10, where the $U_h \times U_w$ is superimposed to the occupancy grid input, most of the row terminal parts do not have a centered u cell that can perfectly fit a prediction. Indeed, as in the case of the highlighted area, two u cells cover the specific end row, and none of the two perfectly fits the position of the ground truth placed in the middle point that connects the two side rows. Nevertheless, each cell contains $k \times k$ positions that can be used to refine the placement of an eventual waypoint detection. More specifically, each u grid cell can predict two values, Δ_x and Δ_y, that let displace possible prediction respect to a reference R_u placed in in the center of the cell. So, the coordinates of a certainly detected waypoint in the

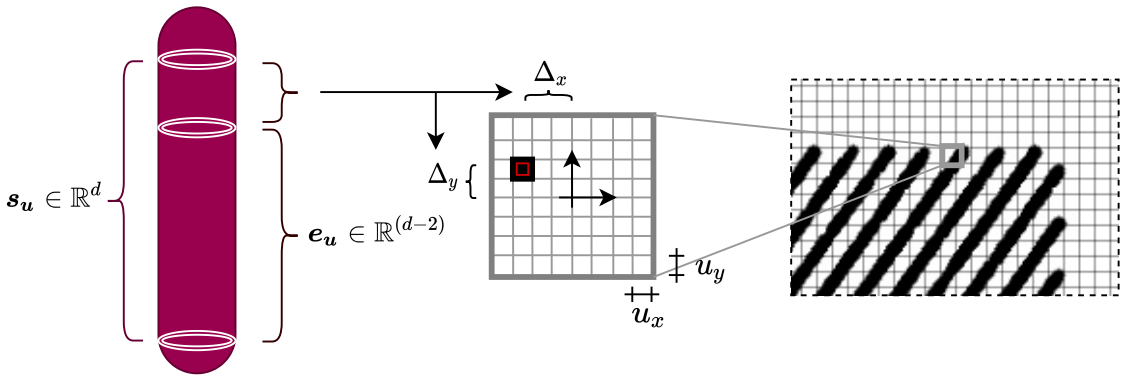


Figure 4.10: DeepWay estimates for each cell u a probability $P(u)$ as the L2 norm of the post activated vector $\boldsymbol{s}_u$. Moreover, two dimensions are used to predict position compensation couple (Δ_x, Δ_y). They adjust detected waypoints on the original occupancy map dimension, $H \times W$. For instance, the highlighted area shows with a red square the actual position of the specific ground truth and the need to displace the prediction from the center of the cell.

original input dimension $H \times W$ can be found using the following equation

$$\hat{\mathbf{y}}_O = k(\hat{\mathbf{y}}_U + \frac{\boldsymbol{\Delta} + \mathbf{1}}{2}) \tag{4.18}$$

where $\hat{\mathbf{y}}_O$ and $\hat{\mathbf{y}}_U$ are the two vectors containing the coordinates x and y in the $\mathcal{R}_O$ and $\mathcal{R}_U$ reference frames, respectively. Position compensations are normalized, and the reference frame, $\mathcal{R}_u$, of the cell u is centered with respect to the cell itself.

Therefore, in order to obtain the final waypoints estimation in the original input space, a confidence threshold t_c is applied to the waypoints confidence in order to select all detected waypoints with a probability $P(u) > t_c$. Subsequently, equation (4.18) is used on all selected waypoints in conjunction with the position compensation maps in order to obtain the respective coordinates on the original reference frame of the input. Moreover, a waypoint suppression algorithm is applied to remove all couple points with a reciprocal Euclidian distance inferior to a certain threshold d_c. The predicted waypoint with the highest $P(u)$ is maintained, and the remaining ones are discarded. Finally, K-Means++ clustering, [13], with 2 as number of cluster is fitted on all embeddings $\boldsymbol{e}_u$ to find the membership group of each predicted waypoint.

Network architecture

DeepWay is a fully convolutional neural network that is directly fed with an occupancy grid map of a row-based crop and predicts waypoints for a successive global path generation. In particular, an input tensor $\mathbf{X}^{(i)}$ is progressively convoluted by a stack of N residual reduction modules. Each module is composed of a series of convolutional 2D layers with Mish, [175], as activation function and channel and spatial attention layers to let the network highlight more relevant features, [255]. Moreover, each module terminates with a convolutional layer with stride two in order to reduce the spatial dimension of the input tensor. After N residual reduction modules, the two first dimensions are reduced by a factor $k + 1$. Therefore, a transpose convolutional layer with stride two is interposed to obtain an output tensor with the two first dimensions equal to $U \times U$. Moreover, as firstly introduced by segmentation networks, [208], a residual connection with the output tensor coming from the $N - 1$ block is added in order to include important spatial information to the tensor before the last layer. Subsequently, similarly to single-stage object detection network [201, 154], the output tensor $\hat{\mathbf{Y}}^{(i)}$ with shape $U_h \times U_w \times d$ is computed with a 1x1 convolution operation. Finally, two non-linear activation functions are employed to compute position compensation couple and confidence probability $P(u)$ of a waypoint. Tanh activation function is applied to the first two $\boldsymbol{s}_u$ dimensions to obtain position compensation couple (Δ_x, Δ_y) and a

squash non-linear activation is applied before the L2 norm.

$$\boldsymbol{a}_u = \text{squash}(\boldsymbol{s}_u) = \left(1 - \frac{1}{e^{||\boldsymbol{s}_u||}}\right) \frac{\boldsymbol{s}_u}{||\boldsymbol{s}_u||} \tag{4.19}$$

The squash function of equation (4.19) ensures that short vectors get shrunk to almost zero length and long vectors get shrunk to a length slightly below one while always maintaining the orientation of the multidimensional array $\boldsymbol{s}_u$. $P(u)$ is obtained as $\| \boldsymbol{a}_u \|_2$. On the other hand, no activation is adopted to obtain the embedding $\boldsymbol{e}_u$ for each waypoint. Afterward, the post-processing pipeline is used to process the output tensor further $\hat{\mathbf{Y}}^{(i)}$ and obtain the final waypoints estimation in the original input space. An overview of the overall architecture of the network is shown in Figure 4.9.

Waypoint refinement

In order to generate a suitable path from the waypoints, we further process the network predictions to refine them and identify the correct order for connecting the waypoints. We cluster the predicted points using K-Means++ clustering on all embeddings $\boldsymbol{e}_u$. This approach automatically clusters together points that are close to each other and can give a first subdivision of the waypoints into main groups. To get the order of the waypoints inside each group, we project them along the perpendicular to the rows and sort them in this new reference system.

The row angle is estimated with the progressive probabilistic Hough transform technique [166]. This algorithm is a classic computer vision feature extraction method, able to detect lines in an image and return an estimate of starting and ending points. Even though this algorithm may seem enough to solve the whole problem of finding the target waypoints in the mask without the need of a neural network, this approach is too dependent on several hyper-parameters that cannot be well-defined a-priori and generally is not able to cope with holes and irregularities which are inevitably present in real-world field occupancy grids. We experimentally find that the application of this method leads to a high number of false-positive and false-negative detections of lines on both the synthetic and the satellite datasets. However, we still use it to estimate the row angle by averaging the orientations of each couple of detected points. In the case of a complete failure of this approach that can happen with the most complex masks, we estimate the angle using a probabilistic iterative process that minimizes the number of intersections with the rows starting from points close to the image center.

After ordering the points inside each cluster, we adopt a refinement approach to insert possible missing waypoints or delete duplicated ones by counting the number of rising and falling edges in the mask along the line connecting two consecutive points. Finally, we compute the order by considering a pattern A-B-B-A. Every intra-groups connection is performed by checking possible intersections with the

rows and correcting the order consequently. If there is a missing point in one of the two groups even after the waypoints refinement process, we remain within the same group, avoiding any possible intersection with the rows. In this way, we put the focus on building feasible paths in the field. So, it is possible to summarize the whole waypoint refinement process as:

1. cluster waypoints using K-Means++ with 2 as number of clusters on all embeddings $\boldsymbol{e}_u$ estimated by the network.

2. estimate orientation of the occupancy grid $\boldsymbol{X}_{occ}$ with the progressive probabilistic Hough transform [166].

3. order points in each cluster projecting them along the normal to the direction estimated in step 2).

4. obtain the final ordered list of waypoints $\boldsymbol{W}_{ord} \in \mathbb{N}^{l \times 2}$ selecting the points from the two main clusters following an A-B-B-A scheme.

The global path is generated from the ordered list $\boldsymbol{W}_{ord}$ with the A*, RRT* or with ARC-PG, [41], algorithms. The latter is specifically designed for the analysed problem and consists of two main steps: the intra-row paths are obtained with a gradient-based planner between the starting and the ending waypoints of each row. On the other hand, the inter-row paths are generated with a circular pattern in order to keep a safe margin from the end of the rows to avoid collision during the turns. Considering an end-row waypoint $\boldsymbol{p}_i$ and the successive point that starts the following row $\boldsymbol{p}_{i+1}$, the waypoints are firstly moved along the row direction to get an end-row margin d_{er}:

$$\begin{aligned} \boldsymbol{p}_i^{shifted} &= \boldsymbol{p}_i + d_{er} \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \\ \boldsymbol{p}_{i+1}^{shifted} &= \boldsymbol{p}_{i+1} + (d_{er} + \Delta\boldsymbol{d}) \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \end{aligned} \tag{4.20}$$

where α is the angle estimated during the post-processing steps and $\Delta\boldsymbol{d}$ is the distance between the two points along the row direction:

$$\Delta\boldsymbol{d} = (\boldsymbol{p}_i - \boldsymbol{p}_{i+1}) \cdot \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \tag{4.21}$$

The end-row margin d_{er} can be selected depending on the meters/pixel resolution of the occupancy grid in order to have a target margin in meters in the real environment. A circular interpolation is adopted to connect the shifted points by linearly interpolating the angles considering the mean point as the center of the circumference. The whole sequence of points obtained by the intra-row and inter-rows

planning creates the global path $\boldsymbol{P} = \{(x, y)|x, y \in \mathbb{R}\}$, defined in the reference system of the occupancy grid $\boldsymbol{X}_{occ}$. If the field map is georeferenced, it is possible to convert the global path into a list of geographic coordinates that can be directly used in the local planning phase to control an autonomous agent motion.

Training process

Learning the end-to-end DeepWay mapping function requires the estimation of model parameters Θ. That is achieved by minimizing a loss $\mathcal{L}$ between the occupancy grid and all the values to be estimate: waypoint probability, positions compensation and set membership. For the first three values, $P(u), \Delta_x$ and Δ_y a modified version of the L_2 sum-squared error is adopted:

$$J(\Theta)_r = \sum_{i,j=0}^{U} [\mathbb{1}_{i,j}^{wp} \lambda^{wp} (y_{i,j} - \hat{y}_{i,j})^2 + \mathbb{1}_{i,j}^{nowp} \lambda^{nowp} (y_{i,j} - \hat{y}_{i,j})^2] \tag{4.22}$$

where, $\mathbb{1}_{i,j}^{wp}$ and $\mathbb{1}_{i,j}^{nowp}$ denote if a waypoint is present or absent from the i, j cell. Therefore, it is possible to give more relevance to cells with a waypoint that are considerably less than true negatives. On the other hand, the embeddings generation $\boldsymbol{e}_u$ is tackled maximizing agreement between representation of the same set via a contrastive loss $J(\Theta)_c$ in the latent space. Defining sim as $\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^T \boldsymbol{v} / \parallel \boldsymbol{u} \parallel \parallel \boldsymbol{v} \parallel$ the contrastive loss has two different contributions:

$$J(\Theta)_p = -\sum_{i,j=0}^{U_p} \mathbb{1}_{[i \neq j]} \log(\text{sigmoid}(\text{sim}(e_i, e_j))) \tag{4.23}$$

and,

$$J(\Theta)_n = -\sum_{i,j=0}^{U_n} \mathbb{1}_{[i \neq j]} \log(1 - \text{sigmoid}(\text{sim}(e_i, e_j))) \tag{4.24}$$

where, U_p and U_n are the group of embeddings $\boldsymbol{e}_u$ of the same or different set, respectively. The first contribution pushes waypoints of the same set towards similar representations in the latent space. Conversely, $J(\Theta)_n$ progressively pushes the cosine similarity of waypoints of different groups to a negative value.

Finally, the final loss is composed by the equal contribution of the two components:

$$\mathcal{L} = J(\theta)_r + J(\theta)_c = J(\theta)_r + (J(\theta)_p + J(\theta)_n) \tag{4.25}$$

4.2.2 Experiments and results

In the following sections are presented experimental results with the proposed methodology. We introduce the generated synthetic dataset, training hyperparameters, and results with different metrics. We conclude with an evaluation with a custom dataset of real satellite images.

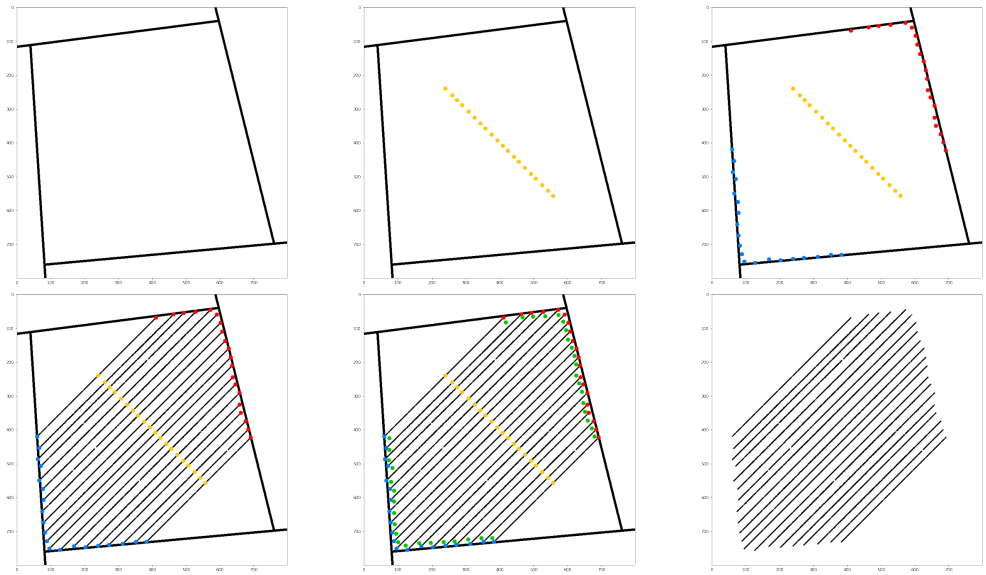


Figure 4.11: Occupancy grid generation process for a 800×800 mask with $N = 20$ and $\alpha = \pi/4$. Firstly random borders are generated, then, N row centers (yellow) are identified starting from the image center. Starting (blue) and ending (red) points are found at the intersection with the borders, with some random displacement to add variability. The actual row lines are then generated, adding holes with a certain probability.

Synthetic and real dataset creation

Due to the lack of online datasets of row crop occupancy grids and the complexity of building a real one in scale, we carefully devise a synthetic one. We mainly focus on linear crops, since the majority of the real-world case enters in this category. Thanks to the geometrical simplicity of a row crop field, we design an algorithm to generate any number of occupancy grids of shape $H \times W$ with a random number of rows N and angle α. We select $N = 20$ as the minimum number of rows per image, and 50 as maximum. α can be any angle between $-\pi/2$ and $\pi/2$. The images are generated as single-channel masks with 1-bit values: 0 for the background and 1 for the occupied pixels.

N points are identified as centres of the rows along a line perpendicular to α starting from the image centre. To take in consideration any possible orientation of the field with respect to the point of view and any possible angle between the rows and the field edges, we generate borders with randomly orientations and we define the first and the last point of each row such that the line that connects them passes trough the row centre and has an orientation equal to α. To further increase the variability of generated images, a random displacement is added to the coordinates of each central point and the length and angle of each row. In this way, the inter-row distance is varying for some pixels, and the field edges are not exactly straight. Finally, holes are randomly placed in order to simulate errors in the occupancy grid

generation, and each image is randomly rescaled to get masks of different sizes. The actual row points are generated as filled circles with a random radius of 1 or 2 pixels, to address the possible variations the width of the rows.

To generate the ground truth waypoints, we start considering the mean between each pair of first and last points of the rows. Then, we move those points towards the inside of the field, ensuring that waypoints are in between the two rows. That is a relevant aspect to ease the final path generation. Indeed, external waypoints could easily lead to wrong trajectories skipping some rows or going through some already covered. Figure 4.11 illustrates all the steps for the masks generation.

In addition to the synthetically generated dataset, we also manually collect and annotate 100 satellite images of different row-based crop scenarios from the Google Maps database. Those images are manually processed to extract both the occupancy grid and the target waypoints for the prediction evaluation. On average, the collected images have a ratio of 0.243 meters for pixel, with an inter-row average distance of 2.61 m. Real-world images are essential to demonstrate the ability of our approach to generalize to real-case scenarios and that training the network with a synthetic dataset is equally effective. Figure 4.12 shows two examples of manually annotated satellite images used as the test set. It is worth to mention that the satellite images also present different variations, with respect to the ones present in the synthetic dataset (e.g. slight row curvature), in order to test the robustness of the methodology.

Experimental settings

The first convolutional layer and the last one have 7 and 3 as kernel sizes. All other ones have 5 and 16 as kernel dimension and number of filters, respectively. On the other hand, we adopt the same default parameters for the channel and spatial attention layers of Woo et al. [255]. We use 3000 synthetic images for training with a resolution of 800x800 and $k = 8$. So, each prediction made by the network, $\hat{\mathbf{Y}}^{(i)}$, over a grid $U_h \times U_w$ with equal axis size, before the post-processing, has a spatial dimension of 100x100. Moreover, we train the network with 200 epochs using Adam

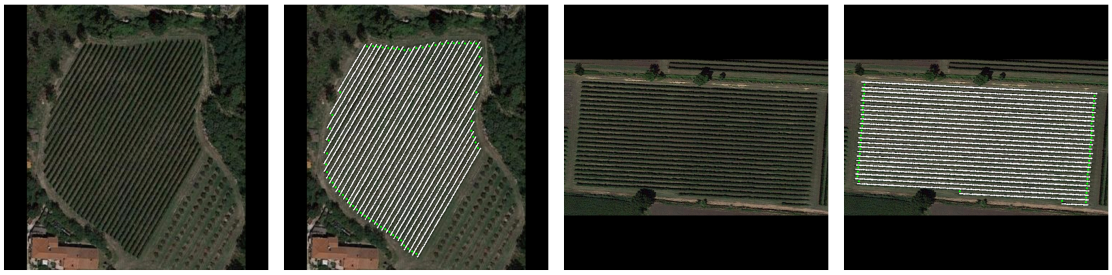
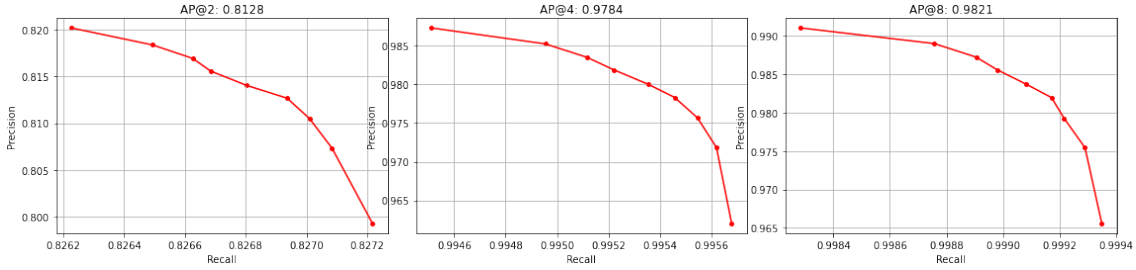


Figure 4.12: Two examples of real-world images taken from Google Maps satellite database and manually annotated. Green points are the ground truth waypoints.



(a) (b) (c)

Figure 4.13: Average Precision results on the synthetic test set for different values of r_c. More restrictive ranges obtain lower values of recall and precision. Conversely, decreasing the value of t_c the precision is mostly affected due to the growing number waypoints with low-score that are predicted by the network.

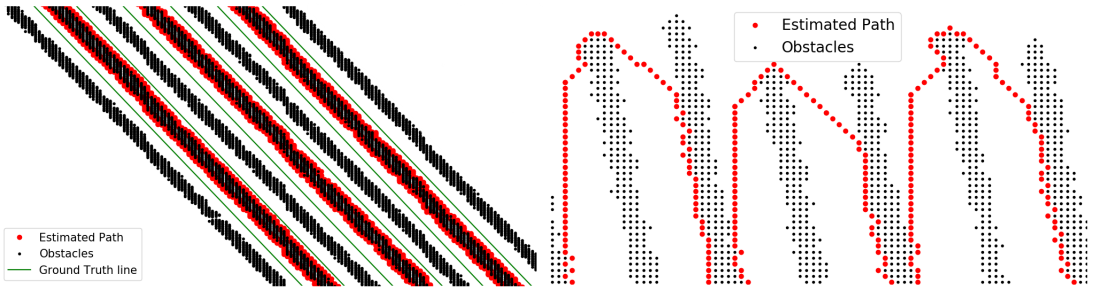
optimizer, [133], with a fixed learning rate equal to $\eta = 3e - 4$ and batch size of 16. The optimal learning rate, η, is experimentally derived using the methodology described in [223].

The resulting fully-convolutional network is a light-weight model with less than 60,000 parameters, a negligible inference latency and that can be easily trained with a commercial GPU in less than 20 minutes. We make use of a workstation with an NVIDIA 2080, 8 GB of RAM and the TensorFlow 2 machine learning platform [1].

Waypoint estimation evaluation

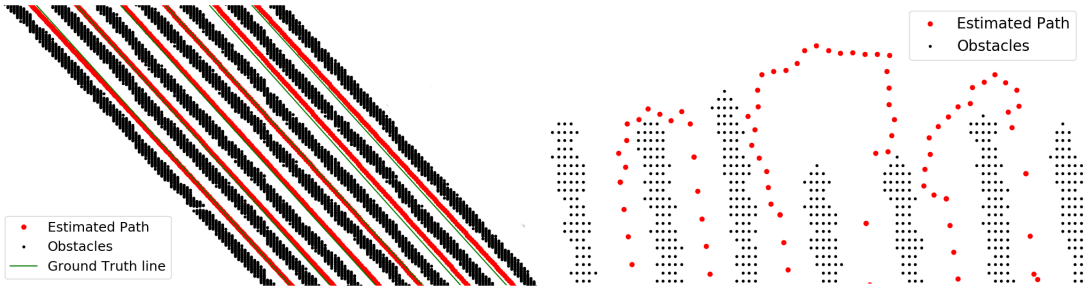
After training, the network is evaluated with 1000 synthetic images. The evaluation aims at assessing its precision and recalls in detecting points within a certain radius r_c from the ground truth. Moreover, as explained in 4.2.1, the waypoint estimation is found setting a certain value of a confidence threshold t_c. So, different recall and precision values can be obtained by fixing different thresholds. For that reason, we adopt an adaptation of the Average Precision (AP) metric that is commonly used across object detection challenges like PASCAL Visual Object Classes (VOC), [74], and MS Common Objects in Context (COCO), [153]. Therefore, if a waypoint prediction is within the selected radius r_c is counted as a true positive (TP). However, if more predictions fall within the selected range, only one is counted as TP and all others as false positive (FP). On the other hand, all ground-truths not covered by a prediction are false negatives (FN). So, the AP computation at a certain distance r_c is obtained with a common definition of recall and precision, varying the value of threshold t_c from zero to one. We set the distance threshold for the waypoints suppression to $d_c = 8$ pixels.

In Figure 4.13 are depicted three graphs obtained with different values of r_c and 0.1 as size step for t_c. At a distance range of 8 pixels, the Average Precision equals 0.9821, but with only a radius of 2, the AP value drops at 0.8128. Moreover, as



(a) (b)

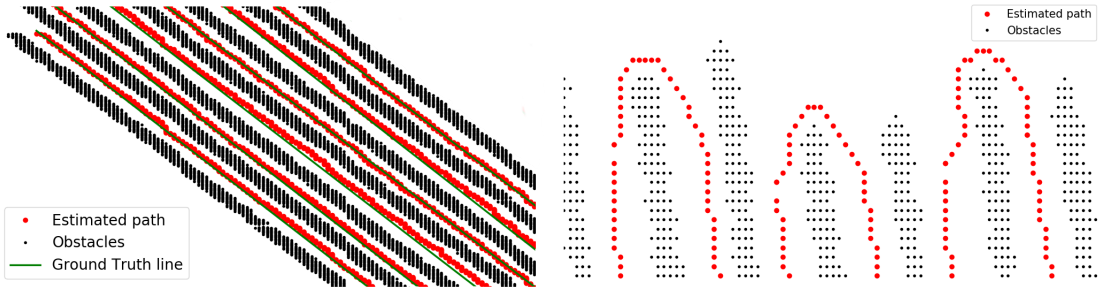
Figure 4.14: An example of a path solution found by the standard A* algorithm. (a) shows the partial path inside the row crops, while (b) represents the portion of the path between two different rows.



(a) (b)

Figure 4.15: An example of a path solution found by RRT* algorithm. (a) shows the partial path inside the row crops, while (b) represents the portion of the path between two different rows.

expected by this metric, recall and precision are inversely proportional by modifying the value of the threshold t_c. In addition to the synthetic test dataset, we also compute the AP metric on the manually annotated satellite images. We reach an AP of 0.9794 with a distance range r_c of 8 pixels (1.94 m on average), 0.9558 with 4 (0.97 m on average) and 0.7500 with 2 (0.49 m on average). As expected, these results are slightly worse concerning the synthetic images with low values of r_c, since the real-world masks are generally more complex, with irregular borders and sudden changes in the length of the rows. All these aspects are only marginally covered by our synthetic generation process, but this does not cause a high drop in the AP metric, meaning that our approach can generalize to real-world examples in a zero-shot manner. Finally, for either synthetic or real satellite test samples, DeepWay is always able to place estimated waypoints in the right set even in the presence of a row shortening as in the first example of Figure 4.12. Indeed, related embeddings of the two crop sides are always very spaced apart in the latent space, making K-Means very effective. In Fig. 4.17, we present a set of satellite images



(a) (b)

Figure 4.16: An example of path solution found by the proposed algorithm ARC-PG. (a) shows the partial path inside the row crops, while (b) represents the portion of the path between two different rows.

with the corresponding predicted and ordered waypoints. In the third image, it is possible to observe how DeepWay is also able to handle big holes in the rows. On the other hand, the fourth image shows how fields with variable orientations can cause sub-optimal predictions and incomplete field coverage.

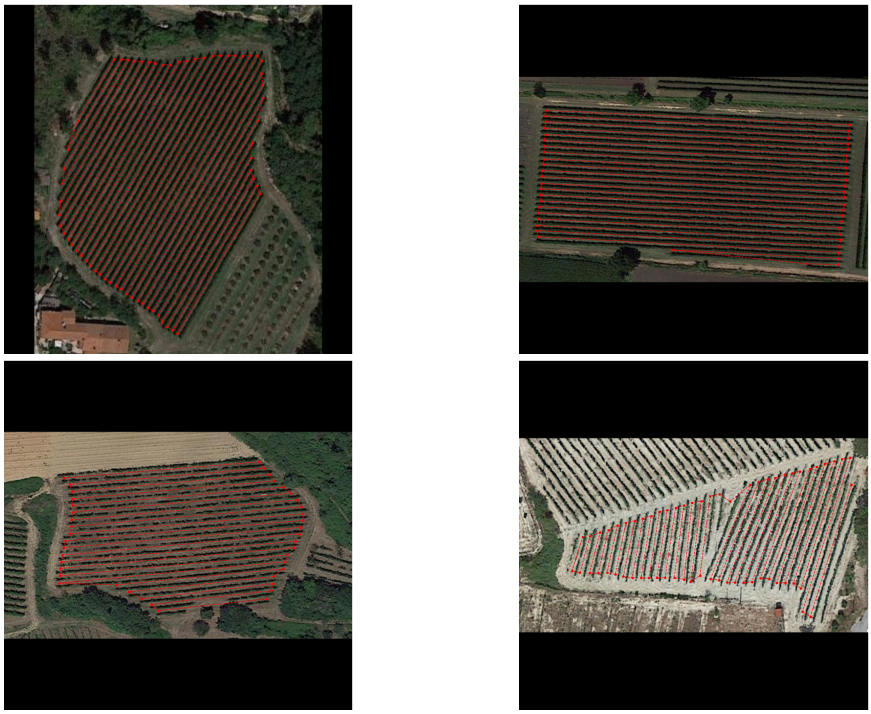


Figure 4.17: Some examples from the dataset of real-world satellite images taken from Google Maps with the ordered predicted waypoints.

Path generation evaluation

In this section, A*, RRT* and ARC-PG, [41], algorithms are evaluated using DeepWay post-processed generated waypoints. In the first place, time consumption and the number of visited cells are compared, as shown in Tables 4.2 and 4.3. All the algorithms have been tested in the same working condition on an Intel Core i7-9750H CPU @ 2.60 GHz and 16 Gb of RAM, using all 1000 synthetic and 100 real-world binary images to check the accuracy of the proposed approach on both artificial and real images.

One of the main advantages of ARC-PG is the ability to detect anomalies during the path computation task that may be caused by missing or wrong ordered waypoints and very close rows crop. This additional feature comes from the intrinsic structure of the algorithm. Indeed, it exploits the gradient descent principle inside rows crop and A* algorithm to switch between different rows. Consequently, if the gradient planner cannot find a valid path between two consecutive waypoints, it means either there are issues with the waypoints (ordering or absence) or the row crops are too close, and the planner fails. Such intrinsic feature requires the attention of a hypothetical user to check what is going wrong in the path computation.

Furthermore, ARC-PG shows promising results in row crop centrality, the number of visited cells, and processing time using both the artificial and the real images dataset. Indeed, it visits a lower number of cells concerning A* and is faster of both A* and RRT* algorithms, as shown in Table 4.2 and Table 4.3. Moreover, it outperforms the A* regarding the row crop centrality as shown in Table 4.4, where the MAE has been computed inside the rows crop with respect to the ground truth reference obtained by connecting the ground truth waypoints with a line. The row crop centrality can be visually checked observing Figure 4.16 and Figure 4.14. Finally, the RRT* can maintain a better central path for ARC-PG algorithm, as can be visually observed comparing Figure 4.16 and Figure 4.15 and numerically in Table 4.4, however, it has very high processing times relative to both dataset type as shown in Table 4.2 and Table 4.3.

For what concerns the Fault Rate (FR) on the synthetic dataset, both the A* and the RRT* algorithms have higher values than the proposed solution, as shown in Table 4.4 because the main fault conditions are free spaces on the same row

Algorithm	$t_{min}[s]$	$t_{max}[s]$	$t_{\mu}[s]$	$\#c_{min}$	$\#c_{max}$	$\#c_{\mu}$
A*	4.89	45.50	10.61	53536	173557	92522
RRT*	56.52	659.08	216.21	39	97	67
ARC-PG	1.44	8.67	4.60	49054	122906	83194

Table 4.2: Comparison between the different path planning algorithms on the artificial dataset. t and $\#c$ stand for time and number of visited cells, respectively.

Algorithm	$t_{min}[s]$	$t_{max}[s]$	$t_{\mu}[s]$	$\#c_{min}$	$\#c_{max}$	$\#c_{\mu}$
A*	1.19	92.78	4.37	26736	436913	70604
RRT*	18.63	475.65	129.38	17	119	55
ARC-PG	0.64	62.96	3.69	23135	355093	63817

Table 4.3: Comparison between A*, RRT* and ARC-PG algorithms on the dataset composed of satellite images. t and $\#c$ stand for time and number of visited cells, respectively.

Dataset Type	**Algorithm**	**MAE**	**FR**[%]
Synthetic	A*	7.88	34
	RRT*	1.08	34
	ARC-PG	1.60	5
Satellite	A*	8.10	29
	RRT*	1.07	43
	ARC-PG	1.57	31

Table 4.4: Comparison of different path planning algorithm in terms of Mean Absolute Error(MAE) and Fault Rate (FR).

crop. It is worth highlighting that the ARC-PG algorithm does not fail in such a condition thanks to using a custom computed cost map. On the other hand, with the real-world dataset all the considered algorithms have similar values of FR since the main issue is a lack or wrong ordering of estimated waypoints. In such conditions, any of the used planners cannot find a valid path. However, the proposed ARC-PG succeeds in notifying the unreliability of the current waypoints array.

Chapter 5

Data Driven Short-Range Remote Sensing for Robotics

Short-Range RS data is increasingly growing in popularity; imaging sensor improvements and the rise in information infrastructure, including processing data, data storage, and communication among the technological devices, are at the foundation of its current expansion. Furthermore, advancements in aerial platforms, UAV/UAS, UGV, and edge devices have increased RS methods' capability and broadened the application areas [189]. It is essential to specify that RS sensors at this level are considered a payload even if mounted on edge vehicles, and no processing is performed on-board machines. Different platforms acquire data, and cloud infrastructures are exploited to process and extract valuable information not only for the goal of the specific application but also for navigation and task execution of autonomous agents. Thus, also for this level, computational load and latency are not a matter of much concern, and the cloud can be employed to exploit the potential of DL algorithms fully. Moreover, a growing amount of literature works propose to make short and long-range RS work in synergy. Indeed, a data-driven approach that works in conjunction with the two sources of information can leverage the advantages of one system to compensate for the limitations of the other.

Within such a context, the presented chapter proposes a deep learning methodology to refine long-range information with UAV for a precision agriculture data pipeline [173, 130]. Indeed, satellite-based imagery proved to be a valuable tool for monitoring, assessing diseases, and driving tasks of autonomous agents. However, freely available satellite imagery with low or moderate resolutions showed some limits in specific agricultural applications, e.g., where crops are grown by rows. Thus, the presented methodology introduces a novel satellite imagery refinement framework based on a deep learning technique that exploits information derived adequately from high-resolution images acquired by UAV/airborne multispectral sensors. A central system can later exploit refined information to drive the behavior of edge vehicles.

5.1 UAV and machine learning based refinement of satellite-driven vegetation index

Precision agriculture is considered to be a fundamental approach to pursue a low-input, high-efficiency, and sustainable agriculture [194] which implements new technological solutions [225]. For precision agriculture to be effective, however, a reliable description of the local status of the crops is essential to perform site-specific management practices when using automatic machinery and robotics [51, 41, 40]. To this extent, the relevance of RS has widely been demonstrated for the extension of in-field surveys to entire plots or even regions [188, 193]. This is particularly true for satellite imagery, which has profitably been exploited for in-field mapping [249], crops status monitoring [163] and diseases assessment [23], both spatially and temporally [38].

However, freely available satellite imagery with low or moderate resolution showed some limits in specific applications, resulting in being not directly suitable for field monitoring purposes in some agricultural contexts [131, 187], such as orchards and vineyards. Indeed, detailed crop information is usually required in these contexts, provided by computing crop status indexes, such as the NDVI [123], even at plant scale. The presence of different elements in these scenarios, such as crops and terrain (inter-row space, in the case of crops grown in rows), causes pixels with diverse nature in low-resolution satellite imagery, which can lead to biased crop indices [131].

This section presents a novel approach to refine long-range satellite imagery by exploiting information derived from short-range UAV-driven high-resolution multispectral images. The proposed method, based on DL techniques, is able to provide enhanced decametric NDVI maps of vineyards from frequent and freely available moderate resolution satellite imagery. In order to train the CNN-based model, only a single UAV-driven dataset is required, making the proposed approach cost-effective and straightforward. In addition, by using a K-Means++ [13] classifier, 3-class vineyard vigour maps are profitably derived from the NDVI maps, which are a valuable tool for growers.

5.1.1 Methodology

The refinement framework developed in this study aims to increase the reliability of the decametric NDVI maps of vineyards derived from freely available satellite imagery. It is based on a CNN-based architecture, hereafter called RarefyNet, which is capable of learning feature representations with a supervised approach after a training phase. The RarefyNet, taking advantage of compositionality, is able to extract in a hierarchical manner features from its input data and exploit its internal knowledge to obtain a refined value of its input samples. In order to train the RarefyNet, a single UAV-driven dataset is used as a reference. Indeed,

NDVI maps from UAV airborne sensors are shown to be more reliable than raw moderate resolution satellites in describing actual crop status [131]. Once trained, the RarefyNet can refine the satellite-driven decametric NDVI maps of the vineyard acquired in any time period during the vine growing season. In addition, using a K-Means-based classifier, vineyard maps with three vigour classes (low, medium, and high vigour) are profitably derived from the NDVI maps, which are a valuable tool for growers.

Considering a decametric NDVI map $\boldsymbol{X}_{raw}$ from a raw satellite dataset, constituted by pixels $x_i \in \boldsymbol{X}_{raw}$, the pixels $\hat{y}_i$ of an enhanced NDVI map $\hat{\boldsymbol{X}}$ can be generated by the RarefyNet non-linear mapping function with parameters Θ as

$$\hat{y}_i = F(\mathbf{X}^{(i)}, \Theta) \tag{5.1}$$

where $\mathbf{X}^{(i)}$ is an input tensor derived from $\boldsymbol{X}_{raw}$. Input tensor $\mathbf{X}^{(i)}$ is defined to collect information, in terms of the NDVI digital value and position on the map, on pixel x_i and on a subset of its neighbourhood. Indeed, the contribution of a map pixel is strictly related to its relative position with respect to its surrounding pixels. More in detail, input tensor $\mathbf{X}^{(i)}$ is thus defined as a three-dimensional tensor $\mathbf{X} \in \mathbb{R}^{(3\times3\times2)}$, where the first layer is a 3×3 map patch (formally $\mathbf{X}^{(i)}_{:,:,0}$), centered in x_i (formally element $X^{(i)}_{1,1,0} = x_i$) and the second layer ($\mathbf{X}^{(i)}_{:,:,1}$) is made of the set of unique location values of map pixels $\mathbf{X}^{(i)}_{:,:,0}$ in the first layer, defined as the linear indexing of the raster matrix. Of course, in order to also consider boundary pixels, a zero-padding operation is performed on the overall maps to allow tensor extraction in boundary pixels. That does not influence the behaviour of later feature maps of the network.

Network architecture

A graphical representation of the overall RarefyNet architecture is illustrated in Figure 5.1. Inspired by [130, 231], input tensor $\mathbf{X}^{(i)}$ feeds a stack of two inception blocks that gradually extract the spatial correlation between the 8 neighbourhood pixels and central target pixel $X^{(i)}_{1,1,0}$. The features of NDVI map $\mathbf{X}_{:,:,0}$ are concurrently processed by an ensemble of parallel convolutional layers with the same number of filters n, but different filter sizes f and dilatation rates k. Indeed, distinct kernel sizes extract different correlations from the data, and, on the other hand, Atrous convolutions take advantage of non-local spatial correlations. Finally, batch normalization [114], as a regularization technique, is applied to each branch before an Exponential Linear Unit (ELU) [50] activation and final concatenation along the feature dimension. Zero padding is applied before each module in order to preserve the first two dimensions of the input tensor. So, starting with the first inception block, an input patch $\mathbf{X}^{(i)}$ with shape (3,3,2) is concurrently processed by the ensemble of parallel convolutions producing an output tensor of shape (3,3,n_{I}) where

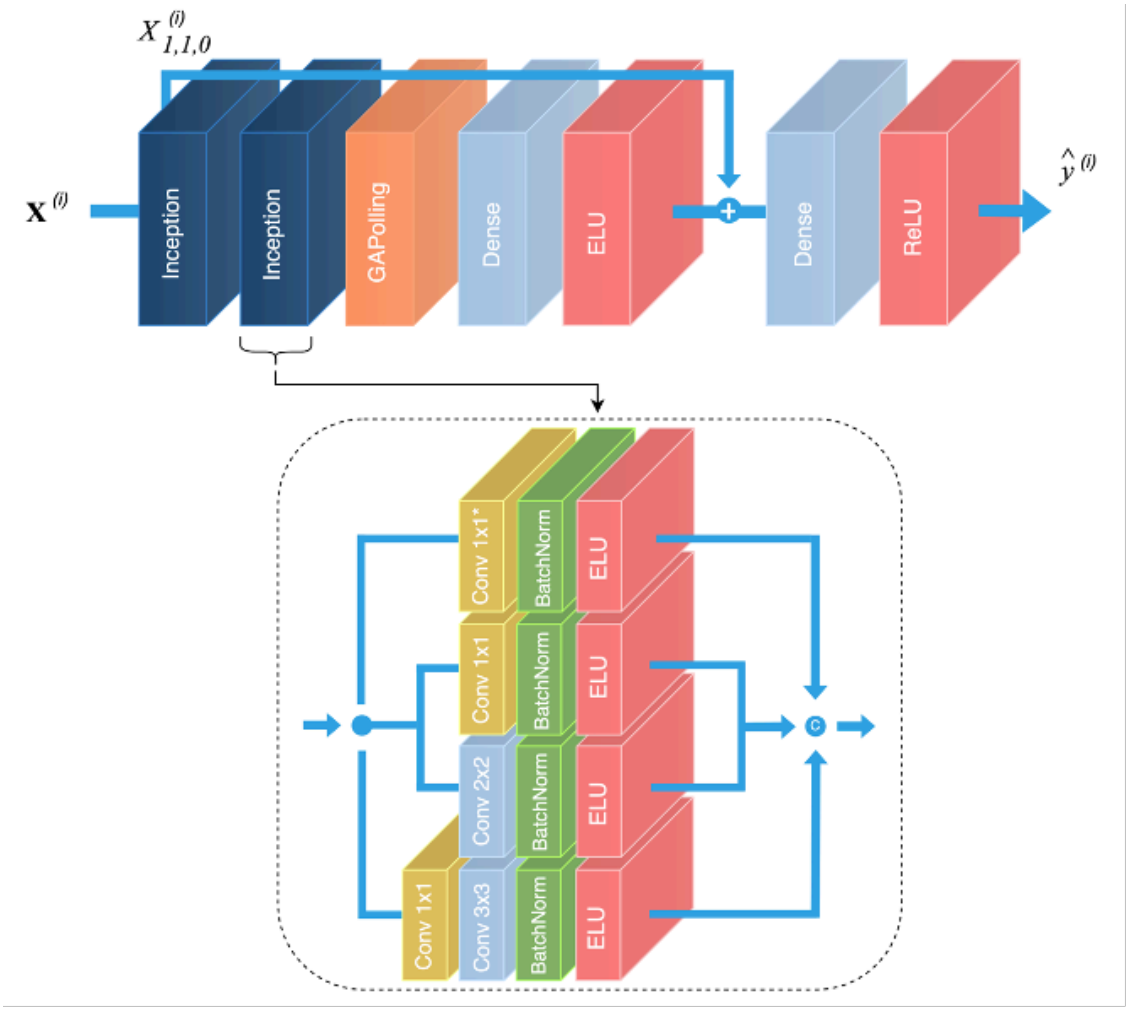


Figure 5.1: Graphical representation of the proposed RarefyNet model. The overall residual architecture is depicted in the top part of the figure with a detailed overview of its inception modules. Input tensors are processed by two inception modules that build their representations on top of each other, concatenating outputs of their different branches.

n_{I} is the result of the feature maps concatenation of the different convolutional branches. The second inception module builds on top of this features tensor by constructing further high-level representations and generating a multi-dimensional array with n_{II} feature maps.

The output tensor produced by the cascade of inception blocks feeds a Global Average Pooling (GAP) layer, which reduces the rank of the input tensor producing a 1-D output array. The GAP operation reduces the spatial dimension of its input tensors, reinforcing the feature maps to be confidence maps of concepts. The GAP 1-D output array feeds a fully connected layer that terminates with a single unit

with ELU as an activation function. The ELU brings non-linearity to the model but still produces both positive and negative values. At this stage, a residual connection sums the output of the dense layer with the original NDVI pixel $X^{(i)}_{1,1,0}$ to be refined. The residual connection, inspired by super-resolution neural network architectures, covers a primary role inside the overall model; it essentially simplifies the role of the first part of the network by moving its objective towards a mere refining operation of the satellite's input pixel. Indeed, the model does not have to recreate the input pixel value after processing the convolutional filters but progressively learns from ground truths how to use the starting satellite input value with its eight neighbors to estimate the inter-row radiometric contributions and refine the raw decametric NDVI value x_i. Finally, a second fully connected layer with Rectified Linear Units (ReLU) with activation functions produces output prediction $\hat{y}^{(i)}$ by removing any offset between the satellite and the UAV NDVI spaces.

The final architecture, shown in Figure 5.1, is thus the result of a careful design aimed at obtaining the best performance in terms of reliability and computational costs. RarefyNet is a lightweight neural network architecture with 16,296 trainable parameters. Every inception block has four parallel branches with different filter sizes f and dilatation rates k. In the first branch (bottom of Figure 5.1), the 1×1 convolution halves the number of feature maps in order to reduce the number of parameters and the computational requirements by the following convolutional layer. The first inception module produces eight feature maps for each branch, which are linked in a unique output tensor with n_{I} channels after being separately preprocessed by a batch normalization layer and an ELU activation function. Equally, the second inception block produces $n_{\mathrm{II}} = 32$ feature maps for each branch, which are linked in a final tensor that feeds the GAP layer. Subsequently, a fully connected layer reduces the 1-D output tensor first to 32 and then to 1 before feeding the residual connection. Moreover, a dropout layer, with $p = 0.2$, is inserted between the two fully connected layers in order to regularize the network and produce a very robust and reliable model [226]. Finally, an output neuron with a ReLU activation function closes the head of the network to compensate and mitigate possible biases.

Training process

To identify an effective set of parameters Θ, the RarefyNet model equation (5.1)) has to be trained. The training phase is an iterative process during which parameters Θ are adjusted to reduce the error defined as the difference between the desired refined NDVI values $\hat{y}$ and reference value y. The enhanced NDVI map derived from the UAV flights is adopted as the reference dataset for the training phase in this application. In particular, the UAV-driven dataset is derived by detecting vineyard canopies within the high-resolution imagery and by a proper down-sampling procedure, described in detail in [130]. The defined training samples

are thus made by the properly paired tensors $\mathbf{X}^{(i)}$, from raw satellite-driven NDVI pixel, and a reference NDVI from the accurate UAV-driven dataset. Moreover, in order to enlarge the number of available training examples and consequently reduce possible overfitting problems, a simple data augmentation technique is applied: considering the i-th sample and maintaining the central satellite pixel $\mathbf{X}^{(i))}_{1,1,:}$ fixed, it is possible to produce $(K-2)$ new samples from each original training data point by rotating the other 8 pixels around the central one.

During the training phase, a loss function $\mathcal{L}$ based on the norm-2 measure

$$\mathcal{L} = \left(\frac{1}{m} \left(\sum_{i=1}^{m} \mid \hat{y}^{(i)} - y^{(i)} \mid^2 \right) \right)^{\frac{1}{2}} \tag{5.2}$$

of the difference between model output predictions $\hat{y}^{(i)}$ and reference $y^{(i)}$ is used together with a mini-batch gradient descent method and m training instances to optimally identify the parameters Θ of the network. The loss function $\mathcal{L}$ is a standard performance measure for regression problems, and it estimates how much error the model typically makes in its predictions, with a higher weight for large errors. Model training is therefore performed iteratively by feeding the network with a batch of a certain size and updating the parameters with small steps determined by learning rate η, using the gradient of the selected loss function.

We adopt the technique proposed by Smith et al. in [223] to identify the maximum value of learning rate $\eta = 5 \cdot 10^{-4}$ to start with. Finally, besides batch normalization and dropout, the AdamW [158] updating rule is used, which is a modified version of the well-known Adam optimizer [133] with fixed L2 regularization. It proposes a simple fix to the classic updating rule, but it has repeatedly shown far better results than standard L2 regularization for all experimentations.

5.1.2 Experiments and results

The effectiveness of the proposed approach to refine moderate-resolution imagery by using UAV-driven imagery is tested in the vineyard selected as the case study. The RarefyNet is implemented in the TensorFlow framework [1] and trained with satellite and UAV-based datasets acquired in May 2017 (time I). For validation purposes, the trained RarefyNet is used to enhance the NDVI map from the satellite platform acquired in three different time periods (June, July, and September: time II, III, and IV), and the results are compared with the more accurate UAV-driven NDVI maps. More in detail, the study is conducted in a vineyard located in Serralunga d'Alba, Piedmont, in the northwest of Italy, shown in Figure 5.2 The selected area includes three parcels, with a total surface of about 2.5 hectares. The area is located at approximately 44°62'4" latitude and 7°99'9" longitude in the World Geodetic System 1984. The test site elevation is within the range of 330 to 420 m above sea level, with steep slope areas (about 20%). Parcels are cultivated

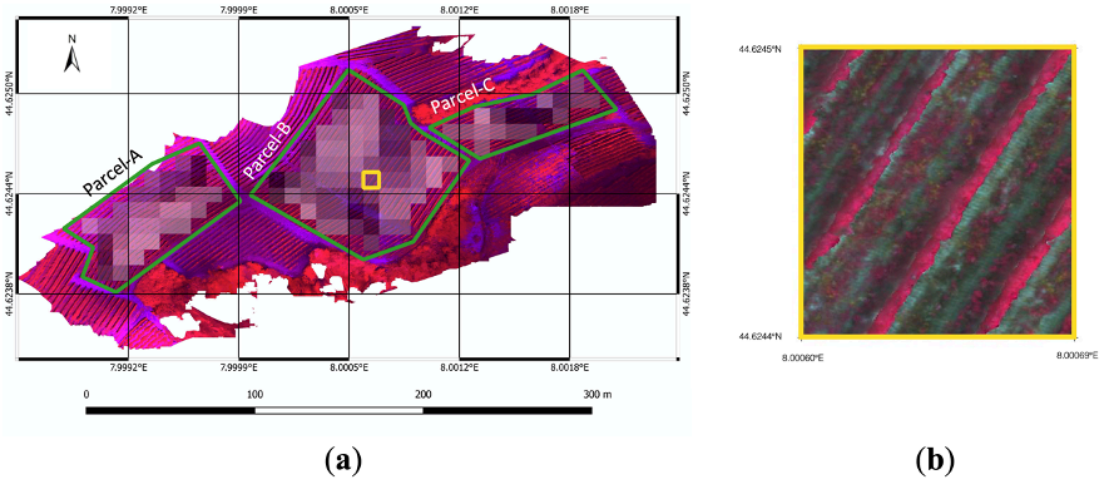


Figure 5.2: **(a)** Selected test field located in Serralunga d'Alba (Piedmont, northwest of Italy). The boundaries of the three considered parcels, named "Parcel-A", "Parcel-B" and "Parcel-C", are marked with solid green polygons. The concurrent illustration of low resolution and high-resolution maps derived from satellite and UAV respectively is represented in false colours (NIR, Red and Green channels). **(b)** Enlargement of UAV imagery highlighted by the yellow square in Figure 1.a.

with the cultivar Nebbiolo grapevine. The vineyard soil is predominantly loamy. The irregularity of the terrain's morphology, in terms of altitude, slope, and soil exposure to the sun, affects microclimatic conditions and water availability within and between parcels [52].

As a side note, the notation used in the following sections is lightened to interpret results better. All symbols in the new notation are again introduced.

Study area and dataset

In this study, cloud-free level-2A Sentinel-2 Bottom of Atmosphere (BOA) reflectance images are used as moderate resolution satellite imagery. Sentinel-2 data products are downloaded from the Copernicus open access hub and imported into a processing platform Science Toolbox Exploitation Platform (SNAP) toolbox (6.0) provided by ESA. Using the subset command in SNAP, pixels of the sentinel-2 images were extracted according to the study area. Geometric, atmospheric, and Bidirectional Reflectance Distribution Function (BDRF) corrections were performed by using a Sen2cor processor, which is a plugin for SNAP [205, 124]. The selected satellite tiles were acquired on four dates during the 2017 growing season (Table 2) to consider different vegetative vine statuses. Only R and NIR bands (bands 4 and 8, respectively), that match with the spectral channels of UAV airborne sensors are used in this study (with ranges 650–680 nm and 785–900 nm, respectively) to

produce the NDVI maps [123], widely used for vegetation monitoring and health assessment of crops. The pixels that were included entirely within the boundaries of the three considered "Parcel A," "Parcel B," and "Parcel C" were selected, as shown in **(a)** of Figure 5.2.

The decametric UAV-based NDVI maps, used as accurate references, are derived from Red and Near-Infrared bands (with ranges 640–680 nm and 770–810 nm, respectively) of high-resolution multispectral imagery acquired by a UAV airborne Parrot Sequoia® multispectral camera. The UAV path was planned to maintain flight height close to 35 m concerning the terrain by properly defining waypoint sets for each mission block on the drone guidance platform based on the GIS cropland map. With this specification, the aerial images ground sample distance (GSD) turned out to be 5 cm (**(b)** of Figure 5.2). The UAV flights were performed on four different dates over the 2017 crop season (Table 5.1), according to the satellite's visiting dates. The high-resolution multispectral imagery was then processed to select only the pixels representing vines canopies and was down-sampled to be in accordance with the satellite's spatial resolution (as described in [131]), obtaining UAV-driven decametric NDVI map Y_{UAV}.

Experimental settings

The RarefyNet used in this experimentation is trained with training tensors derived from raw dataset X^{I}_{raw} and decametric NDVI map Y^{I}_{UAV}, which were acquired in May (time I). More in detail, after the sample's extraction procedure and the data augmentation process applied to the training samples, a set of 1379 and 591 tensors are obtained for the training and test procedures, respectively. The proposed architecture is trained for 300 epochs with a batch size of 64. No learning rate strategies are applied, but the learning rate value is kept constant for all the training epochs of the optimization procedure. All tests are carried out with the

Time period	**Dataset name**	**Acquisition date**	**Source**
I	X^{I}_{raw}	30 April 2017	Sentinel-2
	Y^{I}_{UAV}	5 May 2017	UAV
II	X^{II}_{raw}	6 July 2017	Sentinel-2
	Y^{II}_{UAV}	29 June 2017	UAV
III	X^{III}_{raw}	5 August 2017	Sentinel-2
	Y^{III}_{UAV}	1 August 2017	UAV
IV	X^{IV}_{raw}	17 September 2017	Sentinel-2
	Y^{IV}_{UAV}	13 September 2017	UAV

Table 5.1: Dataset acquisition details from the Sentinel-2 (X_{raw}) and UAV (Y_{UAV}) platforms.

TensorFlow framework [1] on a workstation with 64 GB of RAM, an Intel Core i7-9700K CPU, and an Nvidia 2080 Ti GPU.

Since, at the agronomical scale, maps of classes with different vigour levels can be derived by an expert in-field survey, the validation of the NDVI map refinement is performed by assessing their conformity to a 3-levels vigour map. Thus, a preliminary validation is performed by feeding the trained RarefyNet model with satellite-driven raw map X_{raw}^{II} (time II) and the obtained output, in the form of refined map $\hat{X}^{II}$, is compared with reference map V_{field}^{II} produced by the in-field survey [131]. For completeness, the effectiveness of satellite-driven raw map X_{raw}^{II} and UAV-driven NDVI map Y_{UAV}^{I} in discriminating vigour levels described in V_{field}^{II} is also investigated. To extend validation to other time periods (time I, III and IV), 3-levels vigour maps Y_{UAV} are derived by applying the K-Means++ algorithm to the UAV-driven dataset Y_{UAV}, to be used as the ground truth reference. Indeed, the soundness of this approach is confirmed by validating the selected classifier with the dataset of time II, clustering Y_{UAV}^{II} and comparing it with ground truth vigour map V_{field}^{II} (Figure 5.3). With this approach, the validation of the temporal effectiveness of the proposed satellite-driven dataset refinement framework is performed by refining datasets X_{raw}^{I}, X_{raw}^{III} and X_{raw}^{IV} and assessing the accordance between the obtained refined NDVI maps ($\hat{X}^{I}$, $\hat{X}^{III}$I and $\hat{X}^{IV}$) and the UAV-driven reference ones (Y_{UAV}^{I}, Y_{UAV}^{III}, Y_{UAV}^{IV}).

RarefyNet evaluation

The effectiveness of the refined NDVI map $\hat{X}^{II}$, generated by the trained RarefyNet model, in describing the actual vigour status of the vineyard selected as the case study is investigated by performing Analysis of Variance (ANOVA) between map pixels grouped based on the vigour classes expressed in V_{field}^{II}, selected as the ground truth (**(c)**, Figure5.4). In order to demonstrate the obtained improvement, the coherence of raw satellite-driven map X_{raw}^{II} and of UAV-driven NDVI map Y_{UAV}^{I} with the ground truth is performed. The ANOVA results, organized in Table 5.2, showed how NDVI raw map X_{raw}^{II}, derived from the satellite imagery, has no accordance with the map generated from in-field measurement V_{field}^{II}. The difference between the means of the pixel groups (Figure 5.4), obtained by clustering NDVI map X_{raw}^{II} by using the spatial information provided by in-field survey V_{field}^{II}, is found not to be significant, with obtained p-values ranging from 0.04 to 0.26 for all three considered parcels A, B, and C (Table 5.2). That confirms the limitations of X_{raw}^{II} indirectly providing reliable information regarding the status of the vineyards in this scenario, where the radiometric information reflected from the crop field could be affected by other sources (e.g., inter-row paths) that, in the case of crops grown by rows, could be predominant and could negatively affect the overall NDVI assessment. On the contrary, by using the same assessment approach, the

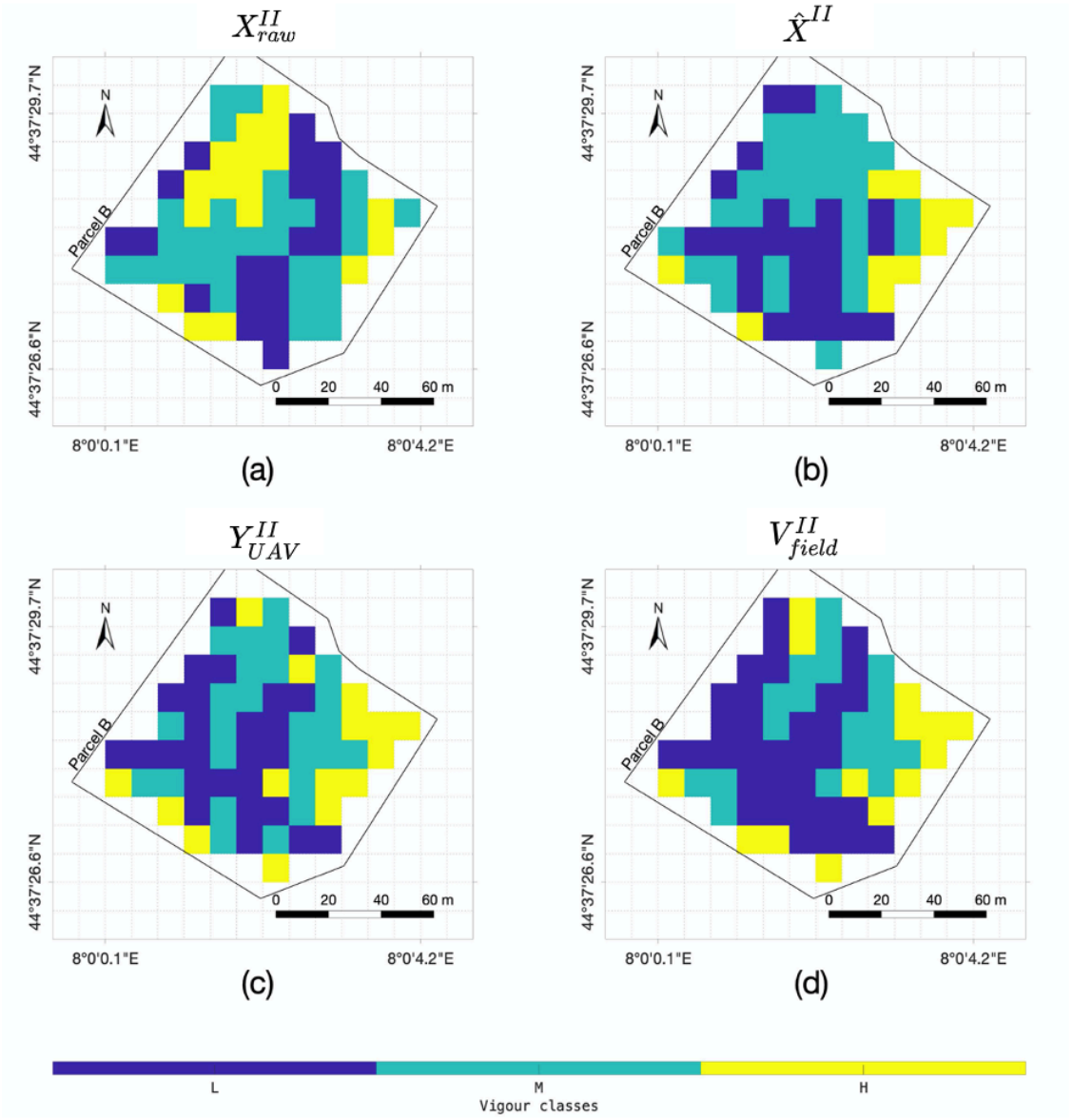


Figure 5.3: 3-level vigour maps **(a)** X^{II}_{raw}, **(b)** $\hat{X}^{II}$ and **(c)** Y^{II}_{UAV} of parcel B, derived from raw Sentinel-2 NDVI map X^{II}_{raw}, refined satellite NDVI map $\hat{X}^{II}$ and UAV-driven NDVI map Y^{II}_{UAV}, respectively. Vigour map **(d)** of parcel B from the expert's in-field survey V^{II}_{field}. Maps X^{II}_{raw}, $\hat{X}^{II}$ and Y^{II}_{UAV} are obtained by the selected K-Means based classifier.

effectiveness of the NDVI map derived from UAV imagery Y^{II}_{UAV} proved to be statistically significant, with different group means in all the considered parcels and showing a favorable coherence with in-field ground truth V^{II}_{field}. This preliminary

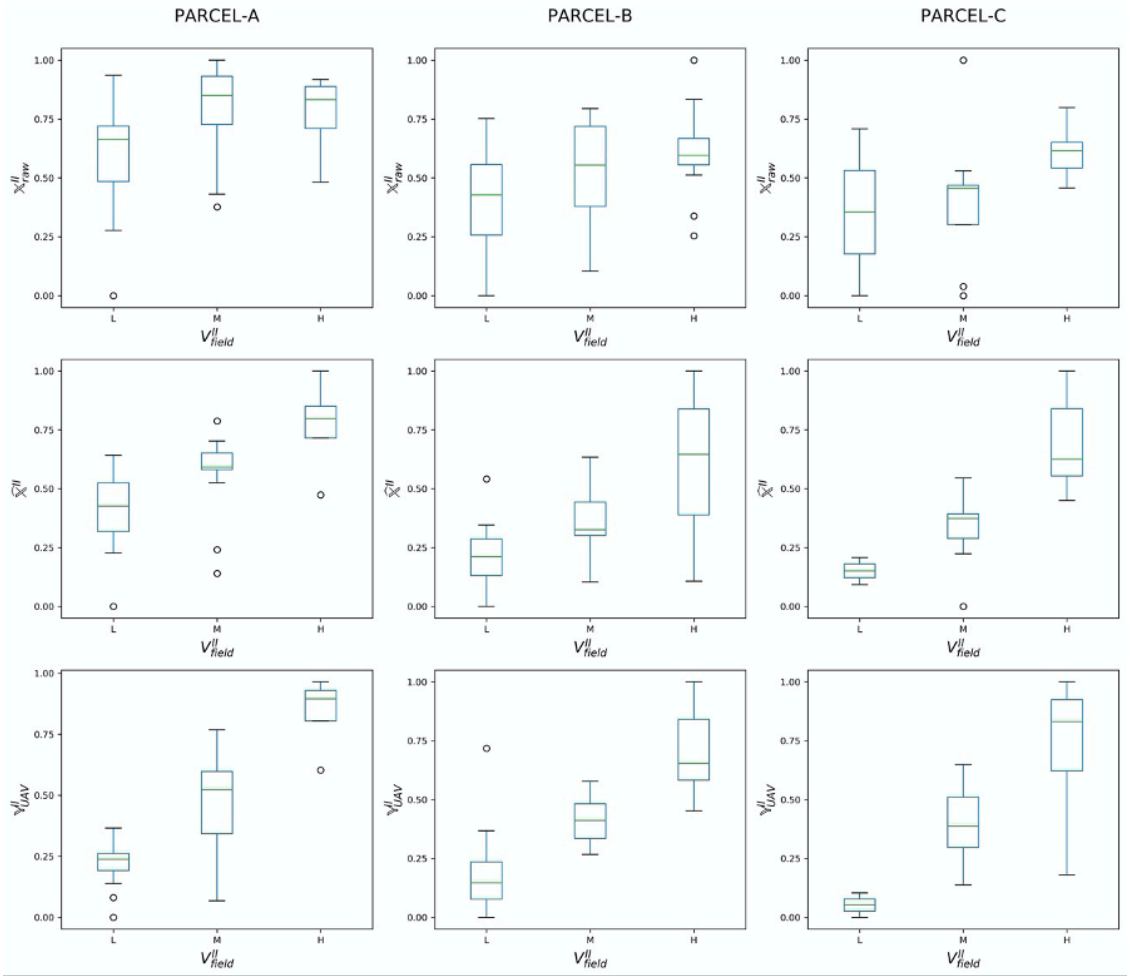


Figure 5.4: Pixel groups boxplots from raw satellite-driven map X^{II}_{raw}, refined satellite-driven map $\hat{X}^{II}$ and UAV-driven map Y^{II}_{UAV}, clustered according to the three vigour classes "L", "M" and "H" defined in map V^{II}_{field}. The boxplots are individually computed for each parcel (A, B and C).

analysis is propaedeutic to the quality assessment of the proposed new framework to refine the satellite-driven NDVI map with the RarefyNet model. The ANOVA results demonstrated how refined NDVI map $\hat{X}^{II}$ correlates with reference V^{II}_{field}, with small p-values ranging from 0.0015 to $3.17 \cdot 10^{-8}$ (Table 5.2), drastically improving the performance of raw satellite-driven dataset X^{II}_{raw}. The results presented so far prove that the proposed RarefyNet is capable of refining the raw Sentinel-2 driven map $\hat{X}^{II}$ of time period II by extracting the features from UAV-driven map Y^{I}_{UAV}.

To extend the performed analysis to other time datasets, all the maps produced from the UAV imagery (Y^{I}_{UAV}, Y^{II}_{UAV}, Y^{III}_{UAV}, Y^{IV}_{UAV}) are clustered into three vigour

Datasets	Parcel	Source	DF	SS	MS	F-Value	P-Value
$X^{II}_{raw}(V^{II}_{field})$	Parcel-A	Classes	2	0.3084	0.1541	3.4582	0.044081
		Error	31	1.3821	0.0445		
		Total	33	1.6905			
	Parcel-B	Classes	2	0.3938	0.1969	4.8928	0.010587
		Error	63	2.5353	0.0402		
		Total	65	2.9291			
	Parcel-C	Classes	2	0.1985	0.0992	1.4555	0.264401
		Error	15	1.0228	0.0681		
		Total	17	1.2213			
$\hat{X}^{II}(V^{II}_{field})$	Parcel-A	Classes	2	0.4749	0.2374	8.0112	0.001568
		Error	31	0.9189	0.0296		
		Total	33	1.3938			
	Parcel-B	Classes	2	1.3735	0.6867	22.9984	3.17E-08
		Error	63	1.8812	0.0298		
		Total	65	3.2547			
	Parcel-C	Classes	2	0.7071	0.3535	11.7444	0.000852
		Error	15	0.4515	0.0301		
		Total	17	1.1586			
$Y^{II}_{UAV}(V^{II}_{field})$	Parcel-A	Classes	2	1.3608	0.6804	30.0925	5.46E-08
		Error	31	0.7009	0.0226		
		Total	33	2.0617			
	Parcel-B	Classes	2	2.7135	1.3567	71.1664	6.87E-17
		Error	63	1.2010	0.0190		
		Total	65	3.9145			
	Parcel-C	Classes	2	0.9447	0.4723	8.7803	0.002988
		Error	15	0.8069	0.0537		
		Total	17	1.7516			

Table 5.2: ANOVA results for the June (time II) datasets X^{II}_{raw}, $\hat{X}^{II}$ and Y^{II}_{UAV} grouped according to ground truth vigour map V^{II}_{field}: raw Sentinel-2 X^{II}_{raw} does not show significant differences among the vigour group means defined by the field expert with in-field measurement V^{II}_{field}, whilst enhanced UAV map Y^{II}_{UAV} and the refined version of Sentinel-2 map $\hat{X}^{II}$ show significant differences among the group means. Degree of freedom (DF), sum of squares (SS) and mean square (MS) are reported with corresponding F-Value and P-Value.

classes by using a K-Means++ algorithm, obtaining a set of clustered maps Y^{I}_{UAV}, Y^{II}_{UAV}, Y^{III}_{UAV} and Y^{IV}_{UAV}. The soundness of the proposed clustering approach is demonstrated by comparing, parcel by parcel, map Y^{II}_{UAV} to in-field vigour map V^{II}_{field} by evaluating the Pearson correlation coefficients (Figure 3). The obtained

positive values, ranging from 0.68 to 0.84, showed that the produced clustered map Y^{II}_{UAV} is well correlated with V^{II}_{field}. This result, together with the highly favorable ANOVA results of Y^{II}_{UAV} in Table 5.2, makes it possible to consider the UAV-driven dataset as a robust and reliable reference in the following analysis.

Moreover, it can be demonstrated that it is possible to exploit the network to refine subsequent time periods, even if trained only with one single UAV-driven dataset. Refinement accuracy decreases with time, but correlations with V_{field} are always higher than X_{raw}. Altogether, it provides a clear demonstration of the possibility to exploit freely, frequently available, low-resolution long-range imagery to describe the variability of vineyards by refining the Satellite driven vegetation index. The synergy between the two RS levels and DL methodologies makes overcoming limitations of either source of data. Refer to Appendix A for further details with other time-series imagery.

Part II

Chapter 6

Neural Efficiency: Optimization and Compression to Enable Edge AI

Neural networks optimization is not only essential for service robotics, but it is at the foundation of Edge AI. Although computational cost is negligible for specific applications and models can be heavily supported by cloud infrastructures, that is not true for devices with limited computational capabilities. That also creates a problem for realizing pervasive deep learning, which requires real-time inference, with low energy consumption and high accuracy, in resource-constrained environments. Indeed, power consumption, latency, memory footprint, and computation load are crucial factors determining the applicability of models at the edge. Furthermore, neural efficiency should not be exclusively pushed by application constraints but should drive all artificial intelligence research. Indeed, deep learning networks today are progressively hitting bottlenecks as they scale to more complex tasks. Researchers attempt to break through the bottleneck by adding more computing power and training data [6]. However, these huge models consume vast amounts of power with limited added value. For instance, a study from the University of Massachusetts, showed that a single large Transformer model consumed 656,000 kWh at the cost of $1M-$3M to train the network [228]. On the contrary, our brain is amazingly efficient, requiring a mere 10 W to learn and act in our world.

Deep learning is usually referred to be inspired by the biological brain. Nevertheless, similarities are already broken by the model of the adopted neuron. Moreover, one of the most remarkable observations about our brain is that activities of neurons and connections are sparse: they might vary from less than one percent to several percent, but they are always highly sparse. Indeed, sparsity is one key ingredient of neural efficiency; dense representations, in which the neurons are both highly interconnected and active, are computationally intensive, limiting scalability and robustness. Therefore, sparsity is the first optimization technique introduced in this chapter.

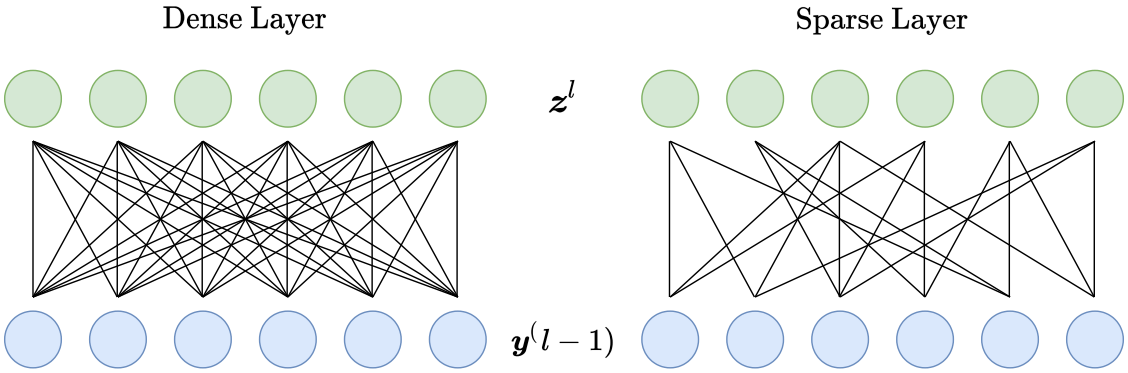


Figure 6.1: Comparison between a layer with dense and sparse weights. Moreover, the non-linear activation function applied to z can push sparsity even further.

The following sections introduce some important topics about neural efficiency that will be helpful for the second part of the presented dissertation. Firstly, it is introduced weights and activation sparsity, highlighting how it can improve memory, and computational efficiency without sacrificing accuracy [6]. Then, it is discussed how it is possible to lower networks bit precision, reduce memory footprint, and improve latency on devices with specific hardware accelerators. A technique known as "quantization" [116]. Subsequently, an example list of current hardware accelerators is presented to leverage sparsity and quantization with limited power consumption, introducing the concept of co-designing NN architecture and hardware. Finally, an optimization technique is introduced that aims to compress knowledge of a cumbersome model into a lightweight version of it. It has some similarities with transfer learning [156], but it has the advantage of producing a smaller and more efficient model [238, 99]. It is worth mentioning that a whole line of work has focused on optimizing the NN model architecture in terms of its micro-architecture. Automated Machine Learning (AutoML) and Neural Architecture Search (NAS) methods play an essential role in this regard. However, the search for NN architectures is strictly bound with generalization, which is connected with efficiency. Therefore, the architecture priors search should be prioritized because a model with enhanced generalization power is more energy/computational efficient and scalable.

6.1 Sparsity in neural networks

One of the reasons why our brain is so efficient is that most of the information stored and processed is extremely sparse: sparse activity and sparse connectivity. On the contrary, deep learning models are utterly dense with highly interconnected and active neurons. In order to obtain the output for each and every neuron, the contribution of each connected neuron must be taken into consideration. However,

we can take inspiration from the biological brain to bring sparsity to artificial neural networks. Firstly, limiting the interconnections between neurons is referred to as weight sparsity (Figure 6.1) and can reduce the number of connections by 9× to 13× [91]. That implies smaller storage size, download size, and memory usage because only non-zero weights need to be stored. Moreover, if an implementation can skip the computation of zero products, significant power, and latency improvements can also be derived from the network sparsity. Common hardware architectures (GP-GPU) are struggling with sparsity because the predictability of memory access patterns allows data to be prefetched in a timely manner, and the dense data packing enables processor vector units to be leveraged to full effect. However, apart from flexible devices such as Field Programmable Gate Array (FPGA), some companies are developing computer chips with sparsity at their core. For instance, the Wafer-Scale Engine (WSE) of Cerebras[1] is one of those.

On the other hand, limiting the number of activations is also possible. That is slightly less common than weights sparsity because activations such as ReLU already clip all logits with a negative sign. Nevertheless, in the literature are presented activations that give a much tighter control to the number of active neurons. For instance, the non-linear activation "k-winners" presented by the research group Numenta[2] allows to retain only the top-k active units of each layer. The rest as ReLU are set to zero. The same research group has also demonstrated how to concurrently use weight and activation sparsity to achieve 100× performance acceleration on FPGA without sacrificing accuracy and improving robustness.

If activation sparsity is easy to control, weights "pruning" raises important considerations. For instance, whether to remove weights gradually or all at once. Moreover, it is also fundamental to decide the criteria for selecting which weights to prune and if to control weight sparsity per layer or globally. Approaches in this field roughly fall into two categories: post-training pruning and progressively pruning during training. In the following two subsections, both methodologies are discussed.

It is also worth mentioning that pruning can be divided as structured or unstructured, depending on the chosen pruning strategy. Indeed, removing weights assessing only their magnitude leads to sparse matrix operations, which are known to be hard to accelerate and typically memory-bound. Nevertheless, it is a hardware-centric categorization that focuses less on the strategy itself.

[1] https://cerebras.net/

[2] https://numenta.com/

6.1.1 Post-training pruning

Network pruning has been widely studied to compress CNN models. In early works, network pruning proved to be a good way to reduce the network complexity, and over-fitting [142, 227]. More recently, Han et al. [91] successfully applied post-training pruning, sparsifying state-of-the-art CNN models with no loss of accuracy. The methodology is straightforward: firstly, networks weights are learned with a standard training procedure. Then, all connections with weights below a certain threshold are removed from the network. Finally, the resulting network is re-trained in order to learn the final weights for the remaining sparse connections. The presented technique can quickly reduce the number of connections by $9\times$ to $13\times$. However, post-training pruning can sometimes compromise accuracy irreparably. Thus, a slower and more controlled pruning precision during training can achieve better performance.

6.1.2 Pruning during training

Zhu et al. [272] accurately evaluated a large-sparse model with comparably-sized small-dense models demonstrating that the first consistently outperform the latter with enhanced efficiency and performance. They also introduced a simple gradual pruning algorithm that requires minimal tuning and can be seamlessly incorporated within the training process that is nowadays widely adopted by the deep learning community. The pruning method is done by having a binary mask with the same size and shape as each weight matrix. At the beginning of the training, the binary matrix is filled with ones. Successively, the binary weight masks are updated every Δt step as the network is trained to increase the sparsity of the network gradually. The following equation guides pruning

$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for } t \in \{t_0, t_0 + \delta t, \ldots, t_0 + n\Delta t\} \tag{6.1}$$

where s_i and s_f are the initial and final sparsity, respectively, and n are the number of pruning steps. The intuition behind equation (6.1) is to prune the network rapidly in the initial phase when the redundant connections are abundant and gradually reduce the number of weights being pruned each time as there are fewer and fewer weights remaining in the network [272]. In the same way as post-training pruning, the resulting pruned networks during training are much more efficient with an order of magnitude fewer weights to be loaded in memory.

6.2 Quantization: from float precision to integer-arithmetic-only inference

Quantization is another optimization technique that usually is placed in the pipeline after pruning. It is loosely related to work in neuroscience that suggests that the human brain stores information in a discrete/quantized form, rather than in a continuous form [174, 244]. For artificial neural networks, the fundamental idea behind quantization is that reducing weights precision brings less memory and obtains a considerable faster inference with dedicated hardware accelerators. Therefore, bit-depth representations can be reduced from float32 to float16 up to int8. Actually, there are also examples of Binary Neural Networks (BNN [112]) or Ternary Weight Networks (TWN [147]), but accuracy is traded off for latency. Even if this section focuses on quantization for inference, it is worth emphasizing that an important success of quantization has also been in NN training [16]. In particular, the breakthroughs of half-precision and mixed-precision training has been the main driver that has enabled an order of magnitude higher throughput in AI accelerators. Nevertheless, it has not proven easy to go below half-precision without significant tuning, and most of the recent quantization research has focused on inference.

Jacob et al. [116] introduced one of the most common quantization strategies that largely improves the trade-off between accuracy and on-device latency. Indeed, they proposed a quantization scheme that makes use of integer-only arithmetic during inference and floating-point arithmetic during training to primarily reduce any accuracy gap (quantization-aware training). Fundamentally, quantization-aware training simulates low precision behavior in the forward pass, while the backward pass remains the same. That induces some quantization error which is accumulated in the total loss of the model, and hence the optimizer tries to reduce it by adjusting the parameters accordingly. However, applying their quantization scheme without affecting the training procedure and with only a small accuracy loss is also possible.

As shown in the right schematic of Figure 6.2, it is an asymmetric quantization scheme because even if the real value $r = 0$ is represented exactly by a quantized value q, $q \neq 0$ if $r = 0$. The affine mapping between quantized values q and real number r is defined by the following equation:

$$r = S(q - Z) \tag{6.2}$$

where S and Z are the scale and zero-point quantization parameters, respectively. The constant S is an arbitrary positive real number and Z is of the same type as q, and is in fact the quantized value q corresponding to the real value 0. The main role of scale is to map the lowest and highest value in the floating range to the highest and lowest value in the quantized range. For instance, in the case of 8-bit quantization, the quantized range would be [-128,127] and dictated by the following

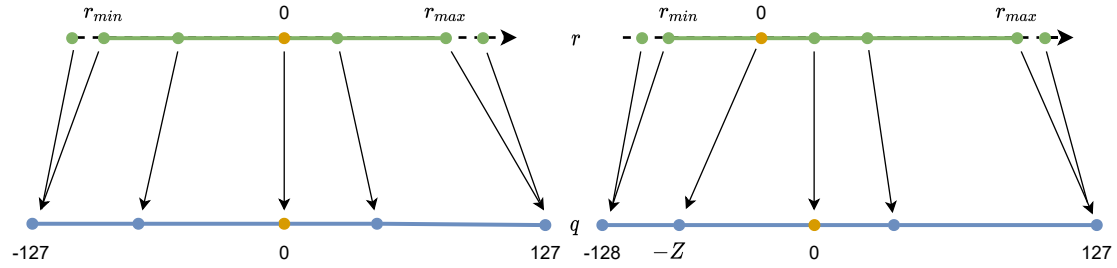


Figure 6.2: Comparison of symmetric quantization and asymmetric quantization in the case of 8-bit quantization.

equation

$$S = \frac{r_{max} - r_{min}}{q_{max} - q_{min}} \tag{6.3}$$

where r_{max} and r_{min} define the quantization range. Similarly, we can find the zero-point by establishing a linear relationship between the extreme floating-point values and the quantized values (equation (6.4)).

$$Z = q_{min} - \frac{r_{min}}{S} \tag{6.4}$$

Finally, quantization ranges are treated differently for weights and activations: for weights, r_{min} and r_{max} are simply the min and max of the different weight matrices. On the other hand, activations ranges depend on the inputs to the network. So, r_{min} and r_{max} are computed with some example samples and Exponential Moving Average (EMA) to aggregate measured ranges.

Once weights are quantized and quantization parameters are computed, it is possible to perform integer-arithmetic-only using equation (6.2). For instance, consider the multiplication of two square $N \times N$ matrices of real numbers, r_1 and r_2, with their product represented by $r_3 = r_1 r_2$, it is possible to write

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1) S_2(q_2^{(i,k)} - Z_2) \tag{6.5}$$

where the entries of each matrix r_α ($\alpha = 1, 2, \text{or } 3$) are written as $r_\alpha^{(i,j)}$ for $1 \leq i, j \leq N$. Equation (6.5) can be rewritten as

$$q_3^{(i,k)} = Z_3 + \frac{S_1 S_2}{S_3} \sum_{j=1}^{N} (q_1^{(i,j)} - Z_1)(q_2^{(i,k)} - Z_2) \tag{6.6}$$

In equation (6.6) the only non-integer term is the ratio between all scales, $M = (S_1 S_2)/S_3$. However, empirically can be found that the ratio M is always in the interval (0,1). Therefore, to further reduce it to Integer-only arithmetic, it is possible to further reduce M as

$$M = 2^{-n} M_0 \tag{6.7}$$

where M_0 is in the interval [0.5,1) and n is a non-negative integer. Using equation (6.7) we can obtain the integer values of M_0 and n which will act as a multiplier and a bit-wise shifter values, respectively. Finally, the accumulating products of integer values requires a 32-bit accumulator. Therefore, in order to have the quantized bias-addition, the bias is of the same type of the accumulator with $Z_{bias} = 0$ as zero-point and $S_{bias} = S_1 S_2$. The down-scaling is carried out by the M multiplier.

The presented quantization scheme brings a size reduction of more than 75% with more than 3× speed-up with specialized hardware accelerators. However, for some networks/applications, the accuracy loss cannot be neglected. For all these cases, it could be helpful to modify the training procedure to create quantization-aware training.

Quantization-aware training

Generally, reducing information of a network can bring a not insignificant loss of accuracy. That loss can be reduced with a process named quantization-aware training proposed by Jacob et al. [116]. Basically, quantization-aware training simulates low precision behavior in the forward pass while maintaining the backward pass unaffected. That induces some quantization error which is accumulated in the total loss of the model, and hence the optimizer tries to reduce it by adjusting the parameters accordingly. Overall, the procedure produces parameters more robust to quantization, considerably reducing the accuracy gap.

The methodology involves the injection of FakeQuant nodes into the training graph after every computation. A FakeQuant node is basically a combination of Quantize and Dequantize operations stacked together (equation (6.2)). Therefore, weights and activations are processed by these FakeQuant nodes as shown in Figure 6.3. The two presented sections are only a broader view of quantization optimization. We refer the interested reader to [83] for a recent and comprehensive survey

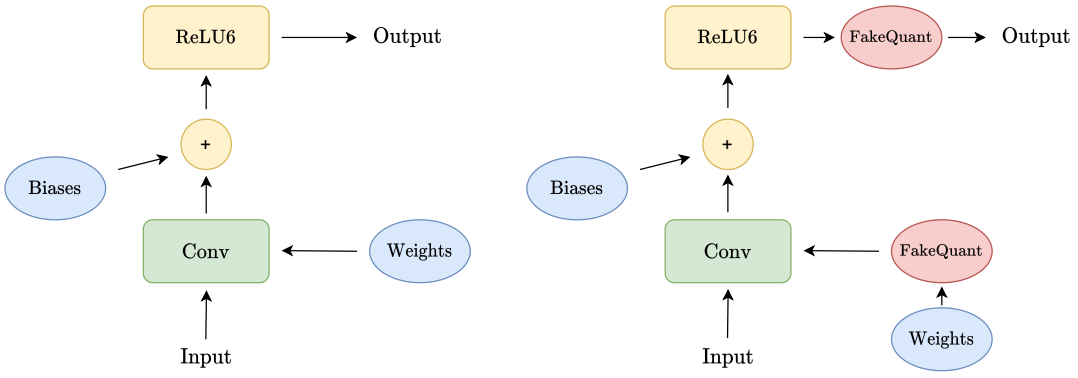


Figure 6.3: Quantization-aware training with FakeQuant nodes injection. ReLU6 activation introduces an upper cutoff to six.

	Intel NCS	**Intel Movidius NCS2**	**Coral USB Accelerator**	**NVIDIA Jetson AGX Xavier Developer Kit**	**NVIDIA Jetson Nano**
AI performance	100 GFLOPs (FP32)	150 GFLOPs (FP32)	4 TOPs (INT8)	32 TOPs (FP32)	472 GFLOPs (FP32)
HW accelerator	Myriad 2 VPU	Myriad X VPU	Google Edge TPU coprocessor	512-core NVIDIA Volta GPU with 64 Tensor Cores and 2x NVDLA Engines	128-core NVIDIA Maxwell GPU
CPU	N.A.	N.A.	N.A.	8-core NVIDIA Carmel Arm v8.2 64-bit CPU 8MB L2 + 4MB L3	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB LPDDR3	4 GB LPDDR3	N.A.	32 GB 256-bit LPDDR4x 136.5 GB/s	4 GB 64-bit LPDDR4 25.6 GB/s
Storage	N.A.	N.A.	N.A.	32GB eMMC 5.1	Micro SD card slot or 16 GB eMMC 5.1 flash
Power	1 W	1.5 W	1 W	10 / 15 / 30 W	5 / 10 W
Size	73 x 26 mm	73 x 26 mm	65 x 30 mm	100 x 87 mm	70 x 45 mm
Weight	18 g	19 g	20 g	280 g	140 g
Price	$ 70	$ 74	$ 60	$ 700	$ 99

Table 6.1: Main specifications of some edge AI devices. Prices indicated for each of them are referred to as the commercial value at the time of writing.

on quantization. We conclude with quantization, briefly mentioning a correlated optimization technique known as weight clustering.

6.2.1 Weight sharing

Clustering, or weight sharing, is an optimization technique strictly correlated with quantization. Indeed, the main idea is to reduce the number of unique weight values in a model, reducing memory footprint and storage size. It first groups the weights of each layer into a certain number of clusters then shares the cluster centroid value for all the weights belonging to the cluster. Weight compression can achieve up to 5× improvements in model compression with minimal loss of accuracy.

6.3 Co-designing NN architecture and hardware together

All optimization techniques discussed need support by the necessary hardware architecture. Indeed, modifications such as quantization, if not adequately backed up by the suitable computational units, can even worse the performance. Thus,

optimization techniques should always consider the particular target hardware platform.

Especially in the last few years, several edge AI devices have been presented to meet power and computational necessity. Edge AI devices are usually low-power embedded systems (TinyML [253]) or single-board computers with hardware accelerators to meet the computational requirements. Table 6.1 offers an example list of some of them with their main specifications. Devices that consume less than 1 mW are not discussed because they refer to a whole new branch of ML known as TinyML. It has several shared optimization concepts presented so far, but it also opens totally different challenges not discussed in this dissertation.

The two versions of Neural Computing Sticks (NCS) are USB dedicated hardware accelerators specifically used to perform deep neural network inferences. The first has a Myriad 2 Vision Processing Unit (VPU), while the second has Myriad X VPU and reaches eight times the performance of the previous generation. These two components request a USB 3.0 or 2.0 interface so that they can be easily used with cheap single-board computers such as a Raspberry board.

The Coral USB Accelerator, released in 2019, is an onboard edgeTPU coprocessor able to reach high-performance machine learning inference, with a limited power cost, for models with full-integer quantization. The board can work at different clock frequencies: maximum or reduced. These frequency types are one twice the other, so using the maximum frequency, there is an increase in inference speed with a consequent increase in power consumption. The edge TPU chip can also be integrated into a single board computer powered by ARM Cortex-M processors. These processor families are designed to be power-efficient for embedded devices and can also be used for NN inference. Because some of the ARM Cortex-M cores do not include dedicated floating-point units, the models should first be quantized before deployment.

Also, the NVIDIA Jetson edge devices are powered by ARM processors. For instance, the NVIDIA Jetson AGX Xavier, released in 2018, is a System-On-Module that guarantees high performance and power efficiency. The board contains DRAM, CPU, PMIC, flash memory storage, and a dedicated GPU for hardware acceleration, so it has been specifically realized to perform rapidly different neural network operations. It is possible to set different power mode configurations also selecting the number of CPU cores utilized: 10 W (2 cores), 15 W (4 cores), 30 W (2, 4, 6, or 8 cores). On the other hand, the NVIDIA Jetson Nano is a lightweight, powerful computer explicitly designed for AI in order to run multiple neural networks in parallel for image elaboration. The board mounts a 128-core NVIDIA Maxwell GPU, a Quad-Core ARM Cortex-A57 MPCore CPU and a 4 GB LPDDR4 memory and reaches the peak performance of 472 GFLOPs. It can work in two power modes: at 5 W or 10 W without the support of Tensor cores during the inference acceleration.

6.4 Distilling the knowledge of a neural network

We conclude this chapter with one last optimization technique that is not connected with the specific hardware implementation in contrast to all other methodologies. Nevertheless, knowledge distillation methods tend to have non-negligible accuracy degradation with aggressive compression, so they are rarely adopted alone. Indeed, the combination of knowledge distillation with prior methods, quantization, and pruning has shown excellent success [197].

The main idea is to train a large model or an ensemble of models and then use them as a teacher to train a more compact and efficient model [99]. In a supervised framework, labels carry only true answers but no other information regarding the problem. On the other end, cumbersome model predictions bring with them all the knowledge encapsulated by the model itself. Indeed, it is conceptually wrong to identify the knowledge in a trained model with only the learned parameter values. A network learns to map input vectors to output vectors, assigning probabilities to all of its outputs. Consequently, cumbersome model predictions, assigning small probabilities to wrong answers, reveal how the model itself generalizes. Thus, using knowledge distillation, we can train a small model to generalize the same way as the large model.

Caruana et al. [30] firstly proved the possibility to compress the knowledge of an ensemble into a single model. Lately, Hinton et al. [99] extended further the idea, using the class probabilities produced by the cumbersome model as targets for training the small model. In the following subsections two distillation methodologies are discussed: soft distillation [99] and hard-label distillation [238].

6.4.1 Knowledge distillation with soft targets

Neural networks typically produce class probabilities by using a softmax activation function, ϕ, placed in the last layer and applied to each logit z_i

$$a_i = \phi(z_i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}} \tag{6.8}$$

where T is a temperature parameter that is normally set to 1. However, setting a higher value for T produces a softer probability distribution. So, increasing T is possible to circumvent the small probability value of negative answers. Therefore, it is possible to distill the knowledge of a cumbersome model using a weighted average of two different objective functions for a classification problem. The first loss is the standard cross-entropy $\mathcal{L}_{\text{CE}}$ with the correct labels $\boldsymbol{y}$. On the other hand, the second component can be a second cross-entropy loss or the Kullback-Leibler KL divergence between the softmax of the teacher and the softmax of the student with a high value of T. Therefore, the distillation objective is

$$\mathcal{L}_{soft} = (1-\lambda)\mathcal{L}_{\text{CE}}(\phi(\boldsymbol{z}_s), \boldsymbol{y}) + \lambda T^2 \text{KL}(\phi(\boldsymbol{z}_s/T), \phi(\boldsymbol{z}_t/T)) \tag{6.9}$$

where $\boldsymbol{z}_t$ and $\boldsymbol{z}_s$ are the teacher and student logits vectors, respectively, and λ is the balancing coefficient. Since the magnitudes of the gradients produced by the soft targets scale as $1/T^2$ it is important to multiply them by T^2 when using both hard and soft targets.

6.4.2 Knowledge distillation with hard targets

Another distillation variant introduced by Touvron et al. [238] makes use of the hard decision of the teacher as a true label. Therefore, if $\boldsymbol{y}_t = \text{argmax}(\boldsymbol{z}_t)$ is the hard decision of the teacher, the distillation objective is

$$\mathcal{L}_{hard} = \frac{1}{2}\mathcal{L}_{\text{CE}}(\phi(\boldsymbol{z}_s), \boldsymbol{y}) + \frac{1}{2}\mathcal{L}_{\text{CE}}(\phi(\boldsymbol{z}_s), \boldsymbol{y}_t) \tag{6.10}$$

The presented alternative is parameter-free because the teacher predictions $\boldsymbol{y}_t$ plays the same role as the true label $\boldsymbol{y}$. It has been proven a better choice with Transformer based networks using label smoothing [232] to convert hard labels into soft labels.

Chapter 7

Autonomous Agents: Navigation and Perception

Edge vehicles are at the heart of a service robotics application, and they are at the end of the data pipeline analyzed so far. They are guided and instructed by information extracted by previous RS layers and leverage onboard computation to navigate and perceive their surrounding world. Unstructured environments, limited computational resources, and complex robot dynamics are only some of the challenges autonomous agents face. Efficiency and robustness to continuous variations are the two imperatives for autonomous agents. They are required to perceive and reason about their environment to accomplish their tasks and navigate our world. Traditionally, various approaches to enable vision, tactile, sound perception, and planning have been proposed. Nevertheless, researchers are starting to realize that the division in separated tasks, even if attractive from an engineering perspective, leads to pipelines that largely neglect interactions between the different stages [157, 270]. Indeed, the divide-and-conquer principle imposes fundamental limits on current autonomous machines guided by classical algorithms.

In the last decade, DL has progressively provided methodologies to overcome fundamental limitations and tackle most of the automatic human activities under system one division proposed by Daniel Kahneman [59]. Indeed, perception is one ability that received a considerable boost with DL algorithms. Moreover, DRL [230], and enhanced perception systems have recently opened the possibility of learning robust and efficient end-to-end policies directly from data without dividing the problem into traditional tasks (e.g., mapping and planning). However, DL still struggled with tasks that required reasoning and long-range planning. For this reason, RS and the related data ecosystem have a significant role in guiding edge vehicles in their environments, expanding their senses, and complementing their limitations. On the other hand, DL networks today can easily consume vast amounts of power, limiting scalability and applicability. Sparsity, reasoned architectural priors, distillation, quantization, and hardware accelerators are some of the

key ingredients to build more reliable and efficient machines.

In this context, two contributions to autonomous agents are discussed in this chapter. The first one completes the precision agriculture pipeline discussed so far; it presents an edge AI solution that leverages long, close-range extracted information to navigate between row-based crops autonomously [5]. Differently, the second work presents a novel lightweight perception solution to provide edge vehicles with the capability to recognize actions performed by humans [170]. Indeed, the majority of service robotics applications that require cooperation with humans necessitate a local understanding of body communication to interact and accomplish tasks. Both examples extensively use diverse edge AI and generalization techniques to provide power-efficient and robust solutions. Other examples of navigation and perception algorithms at the edge proposed by the author of this dissertation can be found in [230, 172, 9, 28].

7.1 Deep semantic segmentation for autonomous navigation in row-based crops

Agriculture 3.0 and 4.0 have gradually introduced autonomous machines and interconnected sensors into several agricultural processes [171], trying to introduce robust and cost-effective novel solutions into the overall production chain. For

Figure 7.1: Field tests with the Jackal platform in different seasonal periods of the same crop. Lush vegetation and thick canopies significantly reduce the GPS accuracy, affecting its reliability and consequently the overall navigation pipeline. Nevertheless, our proposed segmentation-based algorithm exploits semantic segmentation properties to provide a proportional controller that drives the robotic platform along the whole row.

instance, precision agriculture has progressively innovated tools for automatic harvesting [207], vegetative assessment [267], crops yield estimation [77], smart and sustainable pesticide spraying robotic system [61] and many others [130, 200]. Indeed, the pervasiveness of precision agriculture techniques has such a huge impact on certain fields of application that the adoption of them has become increasingly essential to achieve high product quality standards [54]. Moreover, the introduction of robots in agriculture will increasingly have a stronger impact on economic, political, social, cultural, security, [224], and will be the only tool to satisfy the future food demand of our society [70].

Several solutions have been proposed in the literature concerning the autonomous navigation in row-based crops to effectively tackle the well-known GNSS unreliability and degradation inside rows due to vegetation growing. Some of the proposed solutions employ GPS devices along with three-dimensional LIDARs and Inertial Measurement Units (IMU) [33], while others exploit only 2D LiDARs [26]. Nonetheless, they face difficulties reaching a large-scale implementation due to the elevated sensors costs and the lack of robustness due to different factors of variation. These issues made the research community move its focus on different kinds of technologies. Indeed, the state-of-the-art approaches for row crop scenarios with thick and high canopies are based on machine vision [199], and DL [3]. However, in [199] they use a multispectral camera for image acquisition which significantly raises hardware costs, while in [3] the controller provided is not proportional, and it is limited to 3 basics commands.

Building on the latest deep learning research for computer perception and exclusively using low-range sensors, the presented contribution proposes a novel low-cost and at the edge motion control system for autonomous navigation in row-based crops. It exploits semantic segmentation properties to provide a proportional controller that drives the robotic platform along the whole row without colliding with the vine plants. Moreover, it fuses RGB images and depth information of the observed scene to overcome illumination issues and provide a robust control in challenging situations without relying on expensive sensors and GPS signals. Finally, as previously mentioned, the presented work can be inserted in the data pipeline analyzed so far and exploited extracted information of previous levels. For instance, it can make use of long and short-range RS information to navigate from row to row [171] and select a zone of interest for the addressed task such as irrigating or treating specific plants highlighted by predicted vegetative indices [173]. The proposed solution is explicitly designed to adapt to seasonal variation, as shown in Figure 7.1, ensuring robustness in the different situations. Moreover, the low financial impact of our methodology dramatically reduces maintenance and production costs and enables a large-scale adoption of fully autonomous machines for precision agriculture

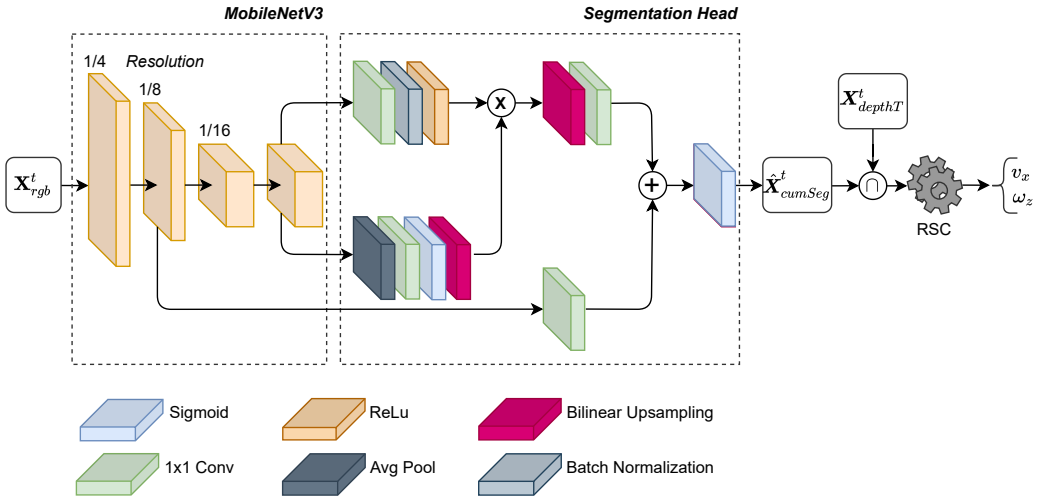


Figure 7.2: The network is fed with the RGB frame acquired by the camera to extract meaningful features. Then a customized segmentation head provides the segmented frame combining features from different resolutions. Successively, we fuse S segmentation maps and intersect the resulting matrix with the thresholded depth map. Finally, the RSC algorithm takes the obtained binary map as input and computes the control values for the autonomous vehicle.

7.1.1 Methodology

We propose a control algorithm to perform real-time autonomous navigation in row-based crops that leverages the latest advancement on edge AI to obtain a continuous and reliable control with a low-range hardware setup. The devised system exploits the joint information of RGB and depth inputs, cutting the costs of expensive sensors and overcoming issues experienced by solutions based on GPS.

The workflow of our proposal, as illustrated in Figure 7.2, is very straightforward; first, the RGB-D camera placed over the mobile platform acquires the depth map $\boldsymbol{X}^t_{depth}$ and the corresponding RGB frame $\mathbf{X}^t_{rgb}$ at a certain time instant t, where $\boldsymbol{X}^t_{depth} \in \mathbb{R}^{h/r \times w/r}$ and $\mathbf{X}^t_{rgb} \in \mathbb{R}^{h \times w \times c}$ with h, w, c and r as high, width, channels and reduction factor, respectively. Then, we feed a custom deep neural network for semantic segmentation that produces a binary mask as shown in the following equation,

$$\hat{\boldsymbol{X}}^t_{seg} = H_{seg}(\mathbf{X}^t_{rgb}, \Theta) \tag{7.1}$$

where $\hat{\boldsymbol{X}}^t_{seg} \in \mathbb{R}^{h/r \times w/r}$ is the segmentation map containing the semantic information of the input image $\mathbf{X}^t_{rgb}$ and Θ are the model variables leaved to discriminative learning.

$$\hat{\boldsymbol{X}}^t_{cumSeg} = \sum_{n=0}^{S} \hat{\boldsymbol{X}}^{t-n}_{seg} \tag{7.2}$$

Successively, in order to obtain more stable and coherent information, we select S consecutive segmentation maps at times $\{t-S, ..., t\}$ and we fuse them as shown in (7.2). Then, we join information coming from the depth channel $\boldsymbol{X}^t_{depth}$ to dynamically reduce the line of sight of the processed scene. Indeed, that is a fundamental point for effective and reliable navigation inside curved crop rows. Moreover, the line of sight reduction is dynamically computed to adjust the depth in different situations and discard as little information as possible from the segmented scene. That is achieved finding the maximum value in the depth matrix $max(\boldsymbol{X}^t_{depth})$ and generating a binary map $\boldsymbol{X}^t_{depthT}$, as follows:

$$X^t_{depthT_{\substack{i=0,...,h \\ j=0,...,w}}}(i,j) = \begin{cases} 0, & \text{if } (X^t_{depth})_{i,j} \geq d_{depth} \\ 1, & \text{if } (X^t_{depth})_{i,j} < d_{depth} \end{cases} \tag{7.3}$$

where $d_{depth} = l_{depth} * max(\boldsymbol{X}^t_{depth})$ and $l_{depth} \in \mathbb{R}$ is a scalar between 0 and 1. So, it is possible to limit the line of sight with an interception operation between the cumulative output of the network and the binary map generated,

$$\boldsymbol{X}^t_{ctrl} = \sum_{n=0}^{S} \hat{\boldsymbol{X}}^{t-n}_{seg} \cap \boldsymbol{X}^t_{depthT} \tag{7.4}$$

where $\boldsymbol{X}^t_{ctrl} \in \mathbb{R}^{h/r \times w/r}$ represents the pre-processed input for the control algorithm. In the $\boldsymbol{X}^t_{ctrl}$ binary map 1 means obstacles and 0 free-space.

Finally, the actuation values for linear and angular velocity, v_x and ω_z, are calculated with a simple algorithm that extracts from the segmentation map a proportional control for the two variables.

$$v_x, \omega_z = \text{RSC}(\boldsymbol{X}^t_{ctrl}) \tag{7.5}$$

In (7.5), RSC is the Row Segmentation Control algorithm that extracts from the pre-processed segmentation map, $\boldsymbol{X}^t_{ctrl}$, a continuous control for the platform. In addition, v_x is the linear velocity along x axis with respect to the robot's frame and ω_z is the angular velocity around z axis with respect to the UGV's frame following the right-hand rule.

The overall solution can be easily integrated with a global path planner based on information extracted from long and close-range RS layers.

Segmentation network architecture

The overall segmentation network acts as a function H_{seg}, parameterized by Θ, that at each temporal instant t takes as input the RGB frame from the onboard camera $\boldsymbol{X}_{rgb} \in \mathbb{R}^{h \times w \times c}$ and produces a binary map, $\hat{\boldsymbol{X}}_{seg} \in \mathbb{R}^{h/8 \times w/8}$. The output positive class segments the crops and the foliage in the camera view. Ideally, it should be equally split into the sides of the frame for a perfectly centered path.

Successively, the semantic information of the row, $\hat{\boldsymbol{X}}_{seg}$, is used in conjunction with its corresponding depth map to control all movements of the platform inside the crops rows.

Among all recent real-time semantic segmentation models, we carefully select an architecture that guarantees high accuracy levels by also containing hardware costs, optimization simplicity, and computational load. Indeed, the segmentation-based control does not considerably benefit from fine-grained predictions and elaborated encoder-decoder networks, [107], or two-pathway backbones, [263], does not bring any considerable improvement. Therefore, we adopt a very lightweight backbone, MobileNetV3 [105], followed by a reduced version of the Atrous Spatial Pyramid Pooling (ASPP) module, [42], to capture richer contextual information with minimal computational impact. Indeed, the output of the last layer of the backbone can not be used directly to predict the segmentation mask due to the lack of spatial details.

The overall architecture is depicted in Figure 7.2. The backbone extracts contextual information with repeated spatial reductions, and two of its branches at different resolutions feed the segmentation head. One layer applies atrous convolution to the 1/16 resolution to extract denser features, and the other one is used to add a skip connection from the 1/4 resolution to work with more detailed information. Finally, in order to maintain real-time performance even without hardware accelerators, we employ a 224×224 low-resolution input. So, we rescale the global average pooling layer setting the kernel size to 12×12 with strides (4,5). Finally, the 1/8 reconstructed map is processed by a sigmoid activation function that produces the output segmentation map $\hat{\boldsymbol{X}}_{seg}$. The reduced spatial resolution decreases latency and makes RSC more robust.

Row segmentation control algorithm

The segmentation-based control algorithm RSC is developed building over the algorithm presented in [5]. Indeed, we propose a simplified version of that control function to avoid useless conditional blocks and obtain a more real-time control algorithm.

Algorithm 1 Segmentation-based algorithm

Input: $\boldsymbol{X}^t_{ctrl}$: Pre-processed segmented image
Output: v_x,ω_z: Continuous control commands

```
noise_reduction_function()
for i=0,··· , w do
    c ←sum_colums(X^t_ctrl)
end for
zeros ← list_zero_clusters(c)
max_cluster ← find_max_cluster(zeros)
if cluster_lenght(max_cluster) ≥ anomaly_th then
    v_x,ω_z ← 0,0
else
    compute_cluster_center()
    v_x,ω_z ← control_function()
end if
```

Algorithm 1 contains the pseudo-code of the proposed custom control, that starting from a pre-processed segmented image $\boldsymbol{X}^t_{ctrl}$, is responsible for computing the driving velocity commands. First, a noise reduction function gets rid of undesired noise in the bottom part of the cumulative segmentation mask $\boldsymbol{X}^t_{ctrl}$ due to grass on the terrain, that may be wrongly segmented by the neural network. To perform such operation, we compute the sum over rows of $\boldsymbol{X}^t_{ctrl}$ obtaining an array $\boldsymbol{g}_{noise} \in \mathbb{R}^{h/r}$, then we set $\boldsymbol{X}^t_{ctrl(indices,:)} = 0$, where *indices* contains the matrix-row indices such that $\boldsymbol{g}_{noise} < th_{noise}$, with $th_{noise} = 0.03 \cdot max(\boldsymbol{g}_{noise})$ as threshold. We perform such operation because, in an ideal segmentation mask there are no 1s at the top of the image and on the bottom, whilst the majority of them are supposed to be in central rows. After the noise reduction phase, we store the sum over columns of the obtained matrix $\boldsymbol{X}^t_{ctrl}$ in the array $\boldsymbol{c} \in \mathbb{R}^{w/r}$, that contains the amount of segmented trees for each column. Therefore, every zero in $\boldsymbol{c}$ is a potential empty space where to route the mobile platform. Then, we select the clusters of zeros in $\boldsymbol{c}$, which are the groups of consecutive zeros, in order to store them in the list ***zeros***. Next, we look for the largest cluster of zeros ***max_cluster*** and in case of the length of such cluster is over an empirically chosen threshold, $anomaly\_th = 0.8 \cdot w/r$, the driving commands are set to zero value, because it means the provided cumulative segmentation mask $\boldsymbol{X}^t_{ctrl}$ has more zeros than ones, that is an anomaly, so for safety reason, the mobile platform is stopped. While, in case of no anomalies, we compute the cluster center that is given as input to the control function. The identified cluster contains the obstacle-free space information that can be exploited to drive the mobile platform safely; consequently, the linear and angular velocities are computed using the center of the selected cluster, which ideally corresponds to the center position of the row in front of the UGV. The desired velocities are

obtained using two custom functions:

$$\omega_z = -\omega_{z,gain} \cdot d \tag{7.6}$$

$$v_x = v_{x,max} \cdot \left[1 - \left[\frac{d^2}{(\frac{w}{2})^2}\right]\right] \tag{7.7}$$

where $\omega_{z,gain} = 0.01$ and $v_{x,max} = 1.0$ are two constants which define the angular gain and maximum linear velocity of the mobile platform respectively, w is the width of $\boldsymbol{X}^t_{ctrl}$ and d is defined as:

$$d = x_c - \frac{w}{2} \tag{7.8}$$

with x_c center coordinate of the selected cluster. Equation (7.6) represents the angular velocity control function, while equation (7.7) has been used to compute the linear velocity, as in [4] and [5]. Eventually, the control velocity commands sent to the actuators are smoothed using the EMA, formalized in equation (7.9), in order to prevent the mobile platform from sharp motion.

$$\boldsymbol{EMA}_t = \boldsymbol{EMA}_{t-1} \cdot (1 - \alpha_{EMA}) + \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} \cdot \alpha_{EMA} \tag{7.9}$$

where t is the time step and $\alpha_{EMA} = 0.18$ the multiplier for weighting the EMA, which value is found experimentally.

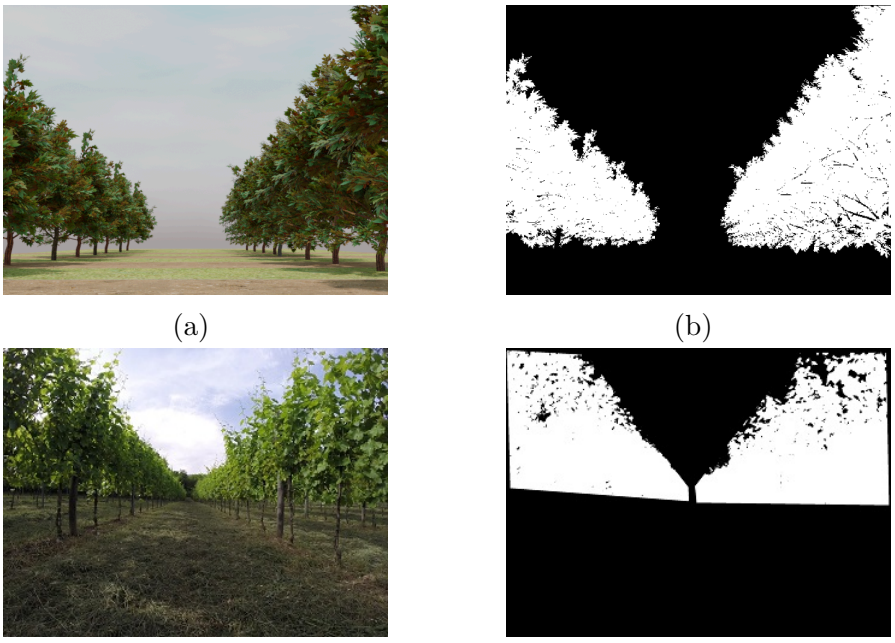
7.1.2 Experiments and results

In this section, we describe the main experiments conducted in simulation and real environments in order to validate the algorithm. First, we discuss the synthetic datasets creation for the training of the network. Then, we illustrate the training process and the optimization techniques adopted to minimize inference costs. Finally, we conclude with the simulation and real environments evaluations.

Dataset creation

We make exclusively use of synthetic data, and domain randomization [236] to enable affordable supervised learning and simultaneously bridge the domain gap between simulation and reality. Moreover, adopting a low-resolution input for the network degrades metrics on the synthetic dataset but at the same time improves generalization on the real-world data. All together allow achieving zero-shot generalization, deploying the trained network in the physical world without any adaptation or fine-tuning.

The segmentation neural network requires a set of RGB images and segmentation masks as inputs to be trained in a supervised manner. As a consequence,



(a) (b)

(c) (d)

Figure 7.3: An example of synthetic and real RGB images **(a)**, **(b)** and the corresponding segmentation masks **(b)**, **(d)**.

inspired by the work of Tejaswi Digumarti et al. [234], we generate synthetic RGB images coupled with the corresponding segmentation masks. For such purpose, we exploit Blender[1] 2.8, which is an open-source 3D computer graphics software compatible with Python language, and the Modular Tree[2] addon to speed up the tree generation. We design four main scenarios: two single different trees, one group of heterogeneous trees, and one row-based scenario with various backgrounds and soils. Then, exploiting Python language compatibility of Blender, we write a script able to automatically capture RGB images and the corresponding segmentation masks of the scene from different positions with respect to the central reference frame and with different illuminations conditions in order to obtain as much as possible a random and complete synthetic dataset. An example of a synthetic RGB image, along with the corresponding segmentation mask, is shown in Figure 7.3. Every single rendering takes about 30 seconds, multiplied by a total of 2776 rendering, which is about 23 hours of continuous work on an RTX 2080 GPU. That total number of images in conjunction with transfer learning allows training a segmentation network with high generalization capabilities while minimizing generation data costs.

[1] https://www.blender.org/

[2] https://github.com/MaximeHerpin/modular_tree/tree/blender_28

The overall segmentation training dataset comprises 2776 RGB synthetic images for training.

Finally, in order to create a dataset for testing the deep neural network, we carry out field surveys in two distinct agricultural areas in the North of Italy: Grugliasco near Turin in the Piedmont region and Valle San Giorgio di Baone in the Province of Padua in the Veneto region. The data is collected at different times of the day, with diverse weather conditions, and they present a variety of terrain types. To have different perspectives inside the rows, we acquire several videos with only three fixed orientations: one pointing the camera at the center of the vineyard row and the other two pointing to the left and right sides, respectively. For training and testing, we select one frame every 25 in order to work with quite different scenarios. In order to generate the masks for the supervised training, we exploit the collected images by first manually annotating them using an open-source software[71], and successively refining the obtained binary masks using a Gaussian mixture model[202] in order to train and evaluate the network with more accurate masks. Overall, our testing dataset consists of 1000 RGB images with their corresponding ground truth (Figure 7.3).

Network training, evaluation and optimization

For training the segmentation network, we employ the TensorFlow[3] 2 framework on a PC with 32-GB RAM, an Intel i7-9700K CPU, and an Nvidia 2080 Super GP-GPU We train our model applying transfer learning to the selected backbone. Indeed, rather than using randomly initialized weights, we exploit MobileNetV3 variables derived from an initial training phase on the 1k classes and 1.3M images of the ImageNet dataset [60]. Moreover, we pre-trained the overall segmentation network with Cityscapes [55], a publicly available dataset with 30 different classes and 5000 fine annotated images. Finally, we adopt a substantial data augmentation with random crops, brightness, saturation, contrast, rotation, and flips. Altogether, those techniques largely improve the model's final robustness and its generalization capability with a reduced number of training samples. We train the network with stochastic gradient descent with a learning rate of 0.03, batch size of $N = 32$, and Intersection over Unit (IoU) as loss function as shown in equation (7.10).

$$\mathcal{L}(\Theta) = \frac{1}{N}\sum_{i=0}^{N}\left(1 - \frac{\hat{\boldsymbol{X}}_{seg}^{(i)} \cap \boldsymbol{X}_{seg}^{(i)}}{\hat{\boldsymbol{X}}_{seg}^{(i)} \cup \boldsymbol{X}_{seg}^{(i)}}\right) \tag{7.10}$$

The accuracy over the test set is 0.8, with an IoU of 0.46. In comparison, the accuracy of the validation set with 0.1 of the synthetic dataset is 0.86 with a domain

[3] https://www.tensorflow.org

gap of 0.08. Moreover, our experimentation shows that larger input sizes improve segmentation over the synthetic dataset but significantly reduce accuracy over real images.

The trained network is optimized to reduce latency, inference cost, memory, and storage footprint. That is obtained with two distinct techniques: model pruning and quantization. We first simplify the topological structure, remove unnecessary parts of the architecture, and favor a more sparse model introducing zeros to the parameter tensors. Subsequently, with quantization, we reduce the precision of the numbers used to represent model parameters from float32 to float16. That can be accomplished with a post-training quantization procedure.

In Table 7.1 experimentation results with some reference architectures are summarized. It is possible to notice how weight precision and graph optimization have a high impact on the inference energy consumed by the network and its impact on the main memory of the tested device.

Motion controller evaluation

As already introduced in Section 7.1.2, all collected images are acquired with three fixed orientations of the crop rows. That allows us to have more flexibility when extrapolating statistical value to assess the controller performances. More specifically, for each row and for each type of video in the test set, we compute the mean value and standard deviation for the three most meaningful variables of the controller: the abscissa, the linear, and angular velocities. Grouping the calculated values by video class, we obtain results summarized in Table 7.2. The outcomes are referred to a frame dimension of 224×224 with maximum linear and angular velocities equal to 1 m/s and 1 rad/s, respectively. Furthermore, we set $l_{depth} = 0.5$, $th_{noise} = 0.03 \cdot max(\boldsymbol{g}_{noise})$, $\alpha_{EMA} = 0.1$, and we fuse $S = 3$ segmentation maps before feeding the RSC algorithm. In addition, for each video orientation of the test set, we compute the fault rate percentage (FR) of iterations where the control algorithm could not provide any command.

Device	**GO**	**WP**	**Latency** [ms]	$\mathbf{E}_{net}$ [mJ]	**Size** [MB]
RTX 2080	N	FP32	28 ± 109	819	9.3
	Y	FP32	0.1 ± 0.3	52	7.4
	Y	FP16	0.1 ± 0.2	39	4.9
Cortex-A57	Y	FP32	111 ± 0.9	166	4.2
	Y	FP16	111 ± 2.3	165	2.2
Cortex-A76	Y	FP32	55.4 ± 10.6	210	4.2
	Y	FP16	65.3 ± 9.5	248	2.2

Table 7.1: Comparison between different devices energy consumption and inference performances with graph optimization (GO) and weight precision (WP).

Class	**Metrics**	**Abscissa**	v_x[m/s]	ω_z [rad/s]	**FR** [%]
Center	μ_{center}	111.15	0.9926	0.0052	0.0
	σ_{center}	5.05	0.0005	0.0005	
Left	μ_{left}	44.42	0.6434	0.3566	0.04
	σ_{left}	7.67	0.0084	0.0083	
Right	μ_{right}	184.93	0.5971	-0.4026	0.26
	σ_{right}	12.98	0.0114	0.0129	

Table 7.2: Overall motion controller evaluation.

Simulation environment evaluation

In addition to previous results, the controller algorithm is tested in two custom simulated environments. As a simulator engine, we use Gazebo[4], which is ROS-compatible and allows us to customize the simulation environments and exploit several plugins to simulate sensors (e.g., cameras). Clearpath Robotics provides the simulation model of the Jackal UGV through the URDF file, which contains specifications about mechanical structure and links related to the robot platform. On the other hand, the simulation environments are designed from scratch as shown in Figure 7.4a and 7.4b. The inter-row distance ranges from 1.70 meters to 2.00 meters, while plants' distance in the same row ranges from 0.70 meters to 1.0 meters from each other. The simulated plane reproduces the bumpy and uneven terrain of a real crop exploiting the heightmap option of the SDF format for model building in Gazebo. The involved simulated camera is an Intel Realsense d435i, which means a Gazebo plugin can provide RGB frames and a depth map of what the UGV is seeing. It is placed 10 centimeters up and with 0 degrees of tilt concerning Jackal's plate.

The real trajectory followed by the Jackal has been obtained using a Gazebo plugin (*libgazebo_ros_p3d*), which provides the real pose of the robot inside the simulation environment so that it is possible to perform additional analysis on the followed path. The procedure to measure the ability of the proposed controller to maintain rows centrality is very straightforward. First, the midline between two crop rows is obtained, averaging the Euclidean distances of the points belonging to a row with respect to the nearest point related to the other rows, and approximating such distances with a cubic line. Subsequently, taking the midline as ground truth and the Jackal's trajectory, the Mean Absolute Error (MAE) is computed to measure the quality of the path.

The first simulation environment consists of straight crop rows, where the proposed controller has achieved excellent results obtaining a MAE= 0.0767 m over

[4] http://gazebosim.org/

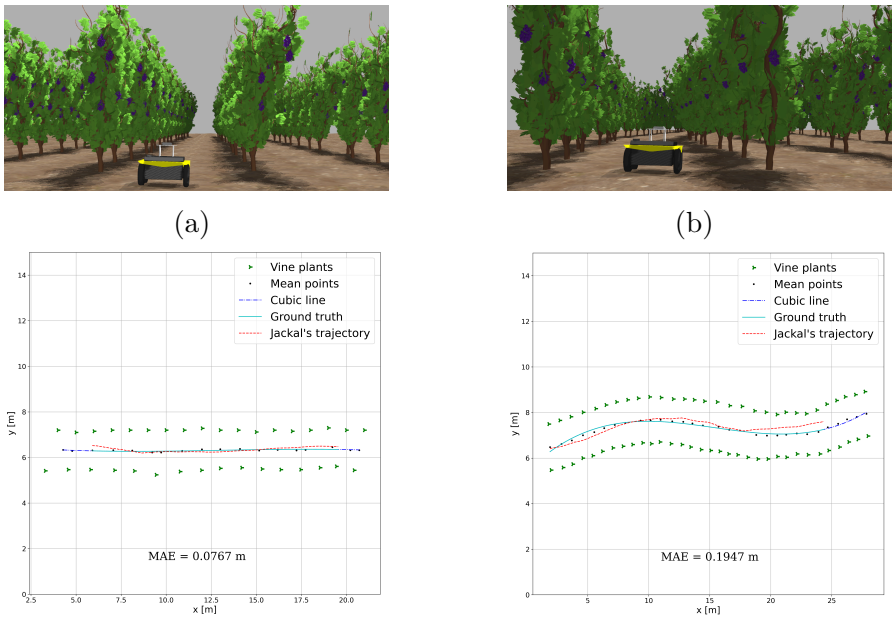


Figure 7.4: **(a)** and **(b)** two examples of straight and curved vine rows, respectively. On the other hand, **(c)** and **(d)** summarize the results coming from the two corresponding simulations. The black points represent the mid-distance between rows, while the blue line is an approximation of such points. Finally, the cyan line highlights the ideal path, and the red one is the trajectory followed by the agent.

(a) Vineyard Row (b) Pear Orchard Row

Figure 7.5: A visual representation of the real world testing environments.

several trials. At the same time, the second simulated crop is composed of curved rows, where the controller slightly worsens the performances achieving an MAE of 0.1947 m. Nevertheless, results are still very promising, taking into account the challenging environment.

Test	N. rows	Min. Error [m]	Max. Error [m]	MAE [m]
Test n. 1	4	0.008	1.395	0.523
Test n. 2	4	0.002	1.105	0.457
Test n. 3	2	0.007	1.320	0.659

Table 7.3: Comparison of minimum, maximum, and Mean Absolute Error (MAE) in three different tests performed in the pear orchard. The second column describes the number of visited rows in the corresponding test.

Test	N. rows	Min. Error [m]	Max. Error [m]	MAE [m]
Test n. 1	4	0.013	0.621	0.296
Test n. 2	6	0.006	0.598	0.218
Test n. 3	6	0.003	0.720	0.204

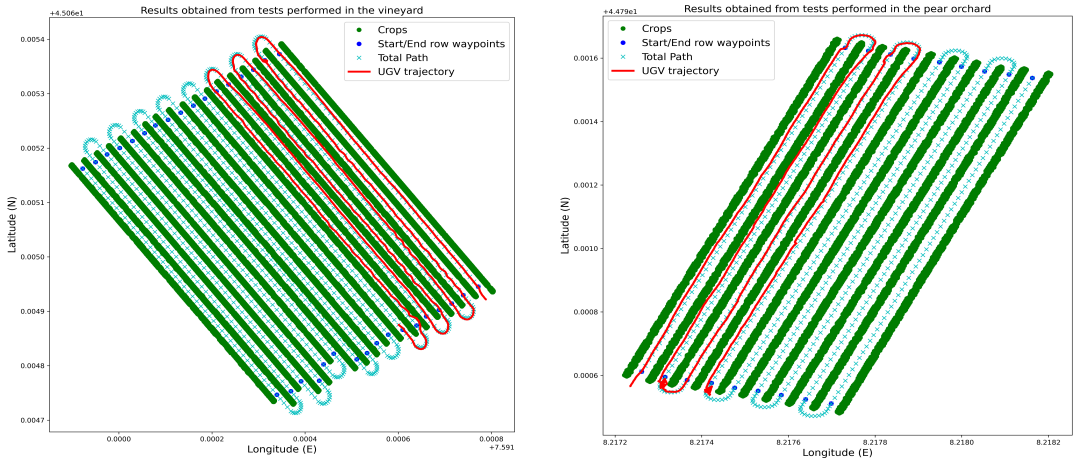
Table 7.4: Comparison of minimum, maximum and Mean Absolute Error (MAE) in three different tests performed in the vineyard. The second column describes the number of visited rows in the corresponding test.

Real environment evaluation

The overall system is tested in two real environment scenarios with multiple experiments in different seasonal periods: a vineyard and a pear orchard, shown in Figure 7.5. Moreover, all the tests have been performed with the same hardware and software setup of the simulation to obtain consistent data: Jackal UGV by Clearpath Robotics, with a 4x4 high-torque drivetrain and an Intel Realsense d435i as RGB-D camera. It is a highly customizable setup, and everything is ROS-compatible, allowing fast deployment and algorithm testing. All the algorithms run on Jackal's onboard Mini-ITX PC with a CPU Intel Core i3-4330TE @2.4GHz and 4GB DDR3 RAM. The vineyard is located in Grugliasco and is managed by the Department of Agricultural, Forestry, and Food Sciences of Università degli studi di Torino (UNITO). In contrast, the pear orchard is located in Montegrosso d'Asti and managed by Mura Mura farm. The first scenario has an inter-row space of about $2.80m$ and a height of about $2.0m$, while the second is organized in rows with an inter-row space of $4.50m$ and a height of about $3.0m$.

The errors, described in Tables 7.3 and 7.4, are computed comparing the RTK-GNSS positions provided by the Piksi Multi [5] and the global path provided to the navigation system. All of this is possible due to the high accuracy GNSS estimated positions, thanks to a clear view of the sky and a good position of the high-end antenna on the UGV. All the collected data are represented in latitude and longitude coordinates. However, they have been transformed in meters for

[5] https://www.spingnss.it/spiderweb/frmIndex.aspx



(a) Test n. 3 in the vineyard scenario (b) Test n. 2 in the orchard scenario

Figure 7.6: A visual representation of the obtained results. Both images contain the path followed by the UGV (red line), the provided global path (cyan x), the start/end row waypoints (blue dots), and the crop (green dots).

analysis purposes with respect to a known GNSS coordinate of the georeferenced occupancy grid map: the top left corner pixel. Tables 7.3 and 7.4, together with the visual representation of Figure 7.6, shows the numerical results obtained by the proposed novel approach and demonstrated that a methodology that exploits semantic segmentation along with a standard navigation approach based on the GNSS is able to provide complete and reliable navigation throughout the whole row-based crop. The results obtained in the pear orchard (Table 7.3) are slightly worse than the vineyard ones in terms of maximum and mean absolute error; however, this effect may be due to the greater inter-row space of the pear orchard with respect to the vineyard one.

7.2 Real-time short-time human action recognition at the edge

Human Action Recognition (HAR) is a problem in computer vision and pattern recognition that aims to detect and classify human actions. The ability to recognize people inside a scene and predict their behavior is fundamental for several robotics applications [206], making autonomous agents able to effectively react and plan their actions. Most previous works dealing with HAR adopted datasets characterized by samples with a long temporal duration [109, 161]. Thus, HAR has been mainly treated as a post-processing operation, classifying complex and long-lasting human actions by exploiting past and future information. Conversely, with this contribution, we focus on short-time HAR, which aims at continuously classifying actions within short past time steps (up to a second). This approach is fundamental to target real-time applications: in robotics, for example, HAR problem should be solved promptly to react to sudden behavioral changes, only relying on near past information.

With this contribution, we propose a new model for HAR called the Action

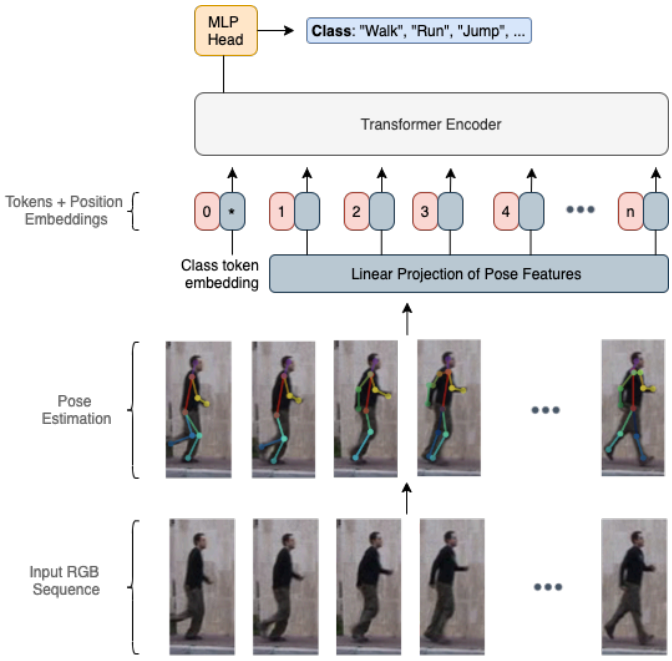


Figure 7.7: Overview of the Action Transformer architecture. Pose estimations are linearly projected to the dimension of the model, and together with the class token, they form the input tokens of the transformer encoder. As for Vision Transformer models [67, 238], a learnable positional embedding is added to each input token. Then, only the output class token is passed through a multi-layer perceptron head to obtain the final class prediction.

Transformer (AcT), schematized in Fig.7.7, inspired by the simple and prior-free architecture of the Vision Transformer [67]. The Transformer architecture [246] has been one of the most important deep learning advances of the last years in NLP. Moreover, multi-head self-attention has proven to be effective for a wide range of tasks besides NLP, e.g. image classification [67, 238], image super-resolution [262], and speech recognition [65]. Furthermore, optimized versions of the Transformer have been developed for real-time and embedded applications [21], proving that this architecture is also suitable for Edge AI purposes. Recently, many models for Human Action Recognition have proposed the integration of attention mechanisms with convolutional and recurrent blocks to improve the accuracy of models. However, solutions that rely exclusively on self-attention blocks have not been investigated for this task yet.

With AcT, we apply a pure Transformer encoder derived architecture to action recognition obtaining an accurate and low-latency model for real-time applications. We study the model at different scales to investigate the impact of the number of parameters and attention heads. We use the new single-person, short-time action recognition dataset MPOSE2021 [170] as a benchmark and exploit 2D human pose representations provided by two existing detectors: OpenPose [35] and PoseNet [190]. Moreover, we compare AcT with other state-of-the-art baselines to highlight the advantages of the proposed approach. To highlight the effectiveness of self-attention, we conduct a model introspection providing visual insights of the results and study how a reduction of input temporal sequence length affects accuracy. We also conduct extensive experimentation on model latency on low-power devices to verify the suitability of AcT for real-time applications.

7.2.1 Methodology

A video input sequence with T frames of dimension $H \times W$ and C channels $\mathbf{X}_{rgb} \in \mathbb{R}^{T \times H \times W \times C}$ is pre-processed by a multi-person 2D pose estimation network

$$\mathbf{X}_{2Dpose} = F_{2Dpose}(\mathbf{X}_{rgb}) \tag{7.11}$$

that extracts 2D poses of dimension $N \times T \times P$, where N is the number of human subjects present in the frame and P is the number of keypoints predicted by the network. The Transformer architecture receives as input a 1D sequence of token embeddings, so each of the N sequences of pose matrices $\boldsymbol{X}_{2Dpose} \in \mathbb{R}^{T \times P}$ is separately processed by the AcT network. Nevertheless, at inference time, all detected poses in the video frame can be batch-processed by the AcT model, simultaneously producing a prediction for all N subjects.

Firstly, the T poses are mapped to a higher dimension D_{model} using a linear projection map $\boldsymbol{W}^{l_0} \in \mathbb{R}^{P \times D_{model}}$. As in BERT [63] and Vision Transformers [67, 238, 21, 36], a trainable vector of dimension D_{model} is added to the input T sequence. This class token [CLS] forces the self-attention to aggregate information

Model	H	D_{model}	D_{mlp}	L	**Parameters**
AcT-μ	1	64	256	4	227k
AcT-S	2	128	256	5	1,040k
AcT-M	3	192	256	6	2,740k
AcT-L	4	256	512	6	4,902k

Table 7.5: Action Transformer parameters for the four version sizes. We fix $D_{model}/\text{H} = 64$, linearly increasing H, D_{mlp}, and L in order to obtain different versions of the AcT network.

into a compact high-dimensional representation that separates the different action classes. Moreover, positional information is provided to the sequence with a learnable positional embedding matrix $\boldsymbol{X}_{pos} \in \mathbb{R}^{(T+1)\times D_{model}}$ added to all tokens.

The linearly projected tokens and [CLS] are fed to a standard Transformer encoder F_{Enc} of L layers with a post-norm layer normalization [246, 43], obtaining

$$\boldsymbol{X}^L = F_{Enc}(\boldsymbol{X}^{l_0}) = F_{Enc}([\boldsymbol{x}^{l_0}_{cls}; \boldsymbol{X}_{2Dpose}] + \boldsymbol{X}_{pos}) \tag{7.12}$$

where $\boldsymbol{X}^L \in \mathbb{R}^{(T+1)\times D_{model}}$ is the overall representation produced by the Transformer encoder at its last layer. Finally, only the [CLS] token $\boldsymbol{x}_{cls}$ is fed into a linear classification head MLP_{head} that performs the final class prediction

$$\hat{\boldsymbol{z}} = \text{MLP}_{head}(\boldsymbol{x}^L_{cls}) \tag{7.13}$$

where $\hat{\boldsymbol{z}}$ is the output logit vector of the model. At training time, the supervision signal comes only from the [CLS] token, while all remaining T tokens are the only input of the model. It is important to notice how the nature of the network makes it possible to accept a reduced number of frames as input even if trained with a fixed T. That gives an additional degree of freedom at inference time, making AcT more adaptive than other existing models. The resulting network is a lightweight solution capable of predicting actions for multiple people in a video stream with high accuracy. The advantage of building on 2D pose estimations enables effective real-time performance with low latency and energy consumption. In order to reduce the number of hyperparameters and linearly scale the dimension of AcT, we fix $D_{model}/\text{H} = 64$, varying the number of transformer encoder heads H, D_{mlp}, and number of layers L to obtain different versions of the network. A simple grid search using train and validation sets is performed to determine lower and upper bounds for the four parameters. In particular, in Table 7.5, we summarize the four AcT versions with their respective number of parameters. The four models (micro, small, medium, and large) differ for their increasing number of heads and layers, substantially impacting the number of trainable parameters.

7.2.2 Experiments and results

This section describes the main experiments conducted to study the advantages of using a fully self-attentional model for 2D pose-based HAR. First, the four variants of AcT described in the previous section are compared to existing state-of-the-art methodologies and baselines on MPOSE2021. Only for a specific comparison with ST-TR [195] and MS-G3D [155], we use additional ensemble versions of our model named AcT-μ (x n), n being the number of ensembled instances. Then, we further analyze the behavior of the network in order to get a visual insight of the attention mechanism and study the performance under a reduction of temporal information. Finally, we study model latency for all the designed architectures with two different CPU types, proving that AcT can easily be used for real-time applications.

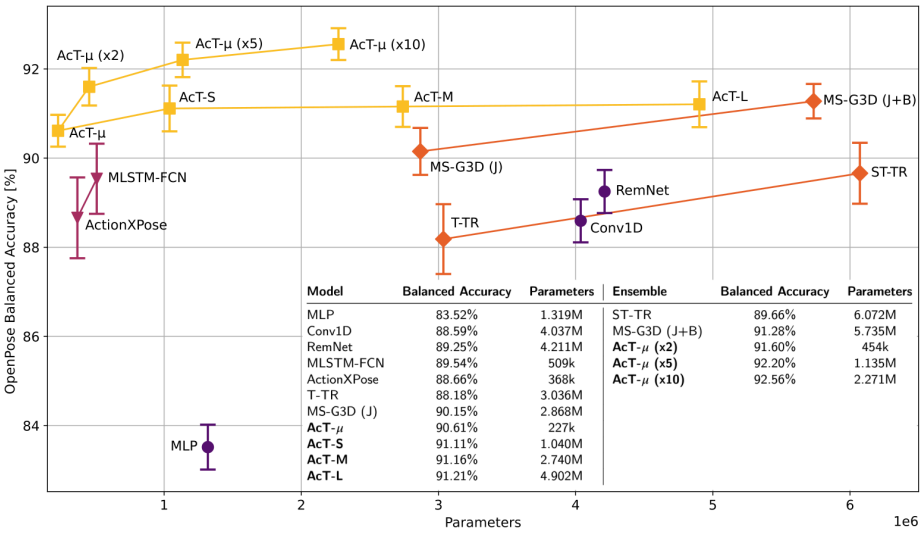
Experimental settings

In the following experiments, we employ both the OpenPose and PoseNet versions of the MPOSE2021 dataset. Either set of data has $T = 30$ and $P = 52$ or 68 features, respectively. In particular, for OpenPose we follow the same preprocessing of [10] obtaining 13 keypoints with four parameters each (position x, y and velocities v_x, v_y), while PoseNet samples contain 17 keypoints with the same information. The training set is composed of 9421 samples and 2867 for testing. The remaining 3141 instances are used as validation to find the most promising hyperparameters with a grid search analysis. All results, training, and testing code for the AcT model are open source and publicly available[6].

In Table 7.5 are summarized all the hyperparameters for the four versions of the AcT architecture and in Table 7.6 all settings related to the training procedure. The AdamW optimization algorithm [158] is employed for all training with the same scheduling proposed in [246], but with a step drop of the learning rate λ to 1e-4 at a fixed percentage (80%) of the total number of epochs. We employ the TensorFlow 2[7] framework to train the proposed network on a PC with 32-GB RAM, an Intel i7-9700K CPU, and an Nvidia 2080 Super GP-GPU. Following the previously defined benchmark strategy, the total training procedure for the four versions takes approximately 32 hours over the three different splits. We exploit publicly available code for what concerns other state-of-the-art models and use the same hyperparameters and optimizer settings described by the authors in almost all the cases. The only exception is made for learning rate, epochs number, and batch size to adapt the methodologies to our dataset and obtain better learning curves.

[6] https://github.com/PIC4SeR/AcT

[7] https://www.tensorflow.org



Model	Balanced Accuracy	Parameters
MLP	83.52%	1.319M
Conv1D	88.59%	4.037M
RemNet	89.25%	4.211M
MLSTM-FCN	89.54%	509k
ActionXPose	88.66%	368k
T-TR	88.18%	3.036M
MS-G3D (J)	90.15%	2.868M
AcT-μ	90.61%	227k
AcT-S	91.11%	1.040M
AcT-M	91.16%	2.740M
AcT-L	91.21%	4.902M

Ensemble	Balanced Accuracy	Parameters
ST-TR	89.66%	6.072M
MS-G3D (J+B)	91.28%	5.735M
AcT-μ (x2)	91.60%	454k
AcT-μ (x5)	92.20%	1.135M
AcT-μ (x10)	92.56%	2.271M

Figure 7.8: Visual representation of the benchmark of different models for short-time HAR on MPOSE2021 splits using OpenPose 2D skeletal representations. For brevity and clearness, the average balanced accuracy on the three splits of MPOSE2021 is reported. The lines connect models that use the same methodology.

Action recognition on MPOSE2021

We extensively experiment on MPOSE2021 considering some baselines, common HAR architectures, and our proposed AcT models. We report the mean and standard deviation of 10 models trained using different validation splits to obtain statistically relevant results. The validation splits are constant for all the models and correspond to 10% of the train set, maintaining the same class distribution. The benchmark is executed for both OpenPose and PoseNet data and repeated for all the three train/test splits provided by MPOSE2021. The baselines chosen for the benchmark are a MLP, a fully convolutional model (Conv1D), and REMNet, which is a more sophisticated convolutional network with attention and residual blocks proposed in [9] for time series feature extraction. In particular, the MLP

Training	
Training epochs	350
Batch size	512
Optimizer	AdamW
Warmup epochs	40%
Step epochs	80%

Regularization	
Weight decay	1e-4
Label smoothing	0.1
Dropout	0.3
Random flip	50%
Random noise σ	0.03

Table 7.6: Hyperparameters used in AcT experiments.

MPOSE2021 Split		OpenPose 1		OpenPose 2		OpenPose 3	
Model	Parameters	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]
MLP	1,319k	82.66 ± 0.33	74.56 ± 0.56	84.41 ± 0.60	74.58 ± 1.0	83.48 ± 0.58	76.60 ± 0.77
Conv1D	4,037k	88.18 ± 0.64	81.97 ± 1.4	88.93 ± 0.43	80.49 ± 0.95	88.67 ± 0.38	83.93 ± 0.58
REMNet [9]	4,211k	89.18 ± 0.51	84.20 ± 0.84	88.77 ± 0.35	80.29 ± 0.88	89.80 ± 0.59	86.18 ± 0.40
ActionXPose [10]	509k	87.60 ± 0.98	82.13 ± 1.5	88.42 ± 0.70	81.28 ± 1.4	89.96 ± 1.0	86.65 ± 1.6
MLSTM-FCN [122]	368k	88.62 ± 0.74	83.55 ± 0.88	90.19 ± 0.68	83.84 ± 1.2	89.80 ± 0.94	87.33 ± 0.67
T-TR [195]	3,036k	87.72 ± 0.87	81.99 ± 1.64	88.14 ± 0.53	80.23 ± 1.19	88.69 ± 0.95	85.03 ± 1.60
MS-G3D (J) [155]	2,868k	89.90 ± 0.50	85.29 ± 0.98	90.16 ± 0.64	83.08 ± 1.1	90.39 ± 0.44	87.48 ± 1.2
AcT-μ	227k	90.86 ± 0.36	86.86 ± 0.50	91.00 ± 0.24	85.01 ± 0.51	89.98 ± 0.47	87.63 ± 0.54
AcT-S	1,040k	91.21 ± 0.48	87.48 ± 0.76	91.23 ± 0.19	85.66 ± 0.58	90.90 ± 0.87	88.61 ± 0.73
AcT-M	2,740k	**91.38 ± 0.32**	**87.70 ± 0.47**	91.08 ± 0.48	85.18 ± 0.80	91.01 ± 0.57	88.63 ± 0.51
AcT-L	4,902k	91.11 ± 0.32	87.27 ± 0.46	**91.46 ± 0.42**	**85.92 ± 0.63**	**91.05 ± 0.80**	**89.00 ± 0.74**
ST-TR [195]	6,072k	89.20 ± 0.71	83.95 ± 1.11	89.29 ± 0.81	81.53 ± 1.39	90.49 ± 0.53	87.06 ± 0.70
MS-G3D (J+B) [155]	5,735k	91.13 ± 0.33	87.25 ± 0.50	91.28 ± 0.29	85.10 ± 0.50	91.42 ± 0.54	89.66 ± 0.55
AcT-μ (x2)	454k	91.76 ± 0.29	88.27 ± 0.37	91.34 ± 0.40	86.88 ± 0.48	91.70 ± 0.57	88.87 ± 0.37
AcT-μ (x5)	1,135k	92.43 ± 0.24	89.33 ± 0.31	91.55 ± 0.37	87.80 ± 0.39	92.63 ± 0.55	89.77 ± 0.35
AcT-μ (x10)	2,271k	**92.54 ± 0.21**	**89.79 ± 0.34**	**92.03 ± 0.33**	**88.02 ± 0.31**	**93.10 ± 0.53**	**90.22 ± 0.31**

Table 7.7: Benchmark of different models for short-time HAR on MPOSE2021 splits using OpenPose 2D skeletal representations.

is designed as a stack of three FC layers with 512 neurons each, followed by a dropout layer and a final FC layer with as many output nodes as classes. Instead, the Conv1D model is built concatenating five 1D convolutional layers with 512 filters alternated with batch normalization stages and followed by a global average pooling operator, a dropout layer, and an FC output stage as in the MLP. Finally, the configuration used for REMNet consists of two Residual Reduction Modules (RRM) with a filter number of 512, followed by dropout and the same FC output layer as in the other baselines.

Regarding state-of-the-art comparisons, four popular models used for multivariate time series classification, and in particular HAR, are reproduced and tested. Among those, MLSTM-FCN [122] combines convolutions, spacial attention, and an LSTM block, and its improved version ActionXPose [10] uses additional preprocessing, leading the model to exploit more correlations in data and, hence, be more robust against noisy or missing pose detections. On the other hand, MS-G3D [155] uses spatial-temporal graph convolutions to make the model aware of spatial relations between skeleton keypoints, while ST-TR [195] joins graph convolutions with Transformer-based self-attention applied to both space and time. As the last two solutions also propose a model ensemble, these results are further compared to AcT ensembles made of 2, 5, and 10 single-shot models. We also report the achieved balanced accuracy for each model and use it as the primary evaluation metric to account for the uneven distribution of classes. The results of the experimentation for OpenPose are reported in Table 7.7 and, in synthesis, in Fig.7.8. The fully convolutional baseline strongly outperforms the MLP, while REMNet proves that introducing attention and residual blocks further increases accuracy. As regards MLSTM-FCN and ActionXPose, it is evident that explicitly modeling both spatial and temporal correlations, made possible by the two separate branches,

MPOSE2021 Split		PoseNet 1		PoseNet 2		PoseNet 3	
Model	**Parameters**	**Accuracy [%]**	**Balanced [%]**	**Accuracy [%]**	**Balanced [%]**	**Accuracy [%]**	**Balanced [%]**
Conv1D	4,062k	85.83 ± 0.71	79.96 ± 1.1	87.47 ± 0.35	78.51 ± 0.78	87.46 ± 0.67	81.31 ± 0.58
REMNet [9]	4,269k	84.75 ± 0.65	77.23 ± 0.94	86.17 ± 0.68	75.79 ± 1.3	86.31 ± 0.60	79.20 ± 0.79
ActionXPose [10]	509k	75.98 ± 0.72	64.47 ± 1.1	79.94 ± 1.1	67.05 ± 1.4	77.34 ± 1.4	66.86 ± 1.4
MLSTM-FCN [122]	368k	76.17 ± 0.84	64.75 ± 1.1	79.04 ± 0.72	65.62 ± 1.4	77.84 ± 1.3	67.05 ± 1.2
AcT-μ	228k	86.66 ± 1,10	81.56 ± 1.60	87.21 ± 0.99	79.21 ± 1.60	87.75 ± 0.53	82.99 ± 0,87
AcT-S	1,042k	**87.63 ± 0.52**	**82.54 ± 0.87**	88.48 ± 0.57	81.53 ± 0.68	88.49 ± 0.65	83.63 ± 0.99
AcT-M	2,743k	87.23 ± 0.48	82.10 ± 0.66	**88.50 ± 0.51**	**81.79 ± 0.44**	**88.70 ± 0.57**	**83.92 ± 0.96**

Table 7.8: Benchmark of different models for short-time HAR on MPOSE2021 splits using PoseNet 2D skeletal representations.

slightly improves the understanding of actions with respect to models like REMNet. MS-G3D, in its joint-only (J) version, brings further accuracy improvement by exploiting graph convolutions and giving the network information on the spatial relationship between keypoints. On the other hand, ST-TR shows performance comparable to all other single-shot models, despite, as MS-G3D, taking advantage of graph information. The proposed AcT model demonstrates the potential of pure Transformer-based architectures, as all four versions outperform other methodologies while also showing smaller standard deviations. Moreover, even the smallest AcT-μ (227k parameters) is able to extract general and robust features from temporal correlations in sequences. Increasing the number of parameters, a constant improvement in balanced accuracy can be observed for splits 3, while split 1 and 2 present oscillations. The difference between splits reflects how much information is conveyed by training sets and how much the model can learn from that. So, it is

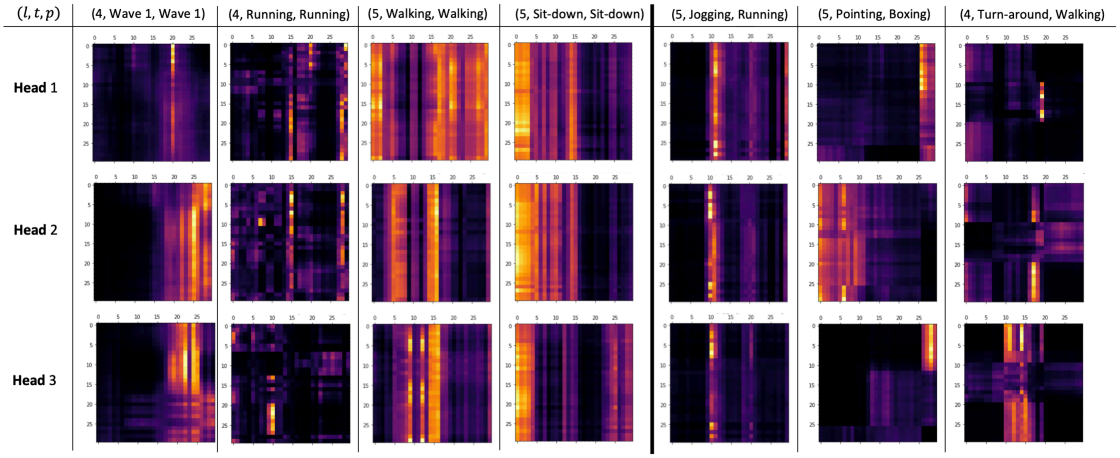


Figure 7.9: Self-attention weights $\boldsymbol{A}$ of MPOSE2021 test samples. (l, t, p) represents the AcT-M l-th layer, the true label and the prediction respectively. The three rightmost columns show three attention maps of a failed prediction and the other columns are from correct classifications. It is clear from all examples how the model focuses on certain particular frames of the series in order to extract a global representation of the scene.

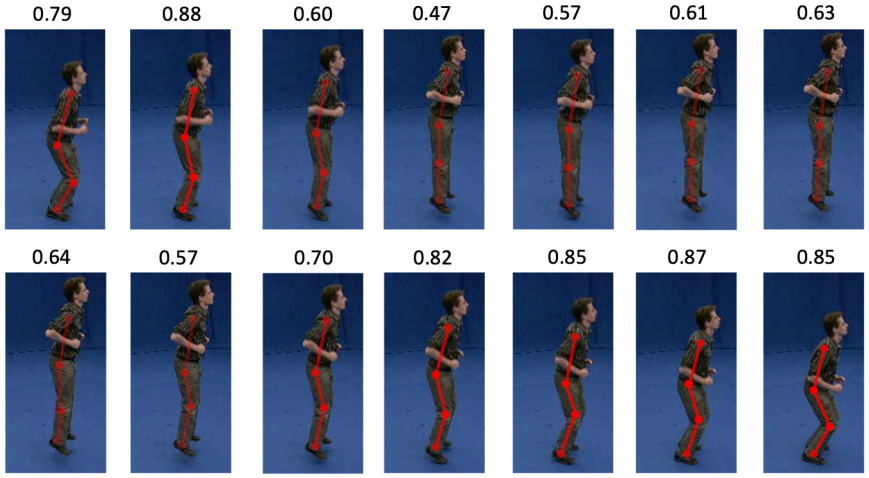


Figure 7.10: Self-attention of the [CLS] token, computed as the normalized sum of the last layer of the different heads. Scores give a direct insight into the frames exploited by AcT to produce the classification output. The example clearly shows how bending positions are more insightful for the network to predict the jumping-in-place action. In the image, the attention score defines the skeleton alpha channel.

evident that AcT scales best on split three because it presents complex correlations that a bigger model learns more easily. On the contrary, it seems AcT-μ is able to extract almost all the information relevant for generalization from split 2, as the accuracy only slightly increases going towards more complex models.

On the other hand, ST-TR exploits an ensemble of two networks modeling spatial and temporal sequence correlations, respectively. Moreover, MS-G3D leverages further information such as skeleton graph connections and the position of bones in one of the ensembled networks. Ensembles are very effective in reducing model variance and enhancing performance by exploiting independent representations learned by each network, so it is unfair to compare them with single architectures. For this reason, we create three ensemble versions of AcT-μ to have an even confront, with 2, 5, and 10 instances, respectively. To compute ensemble predictions, we average the output logits of the network instances before passing through a softmax function. The results reported at the bottom of Table 7.7 show that AcT-μ (x2) outperforms MS-G3D (J+B) in all the benchmarks except for balanced accuracy in split 3, despite having less than one-tenth of its parameters. Finally, the ensembles AcT-μ (x5) and AcT-μ (x10), made of 5 and 10 instances respectively, achieve even higher accuracy on all the splits with only around 1 to 2 million parameters. That proves how the balancing effect of the ensemble enhances model predictions even without feeding the network additional information.

As PoseNet data is mainly dedicated to real-time and Edge AI applications, only the models designed for this purpose have been considered in the benchmark,

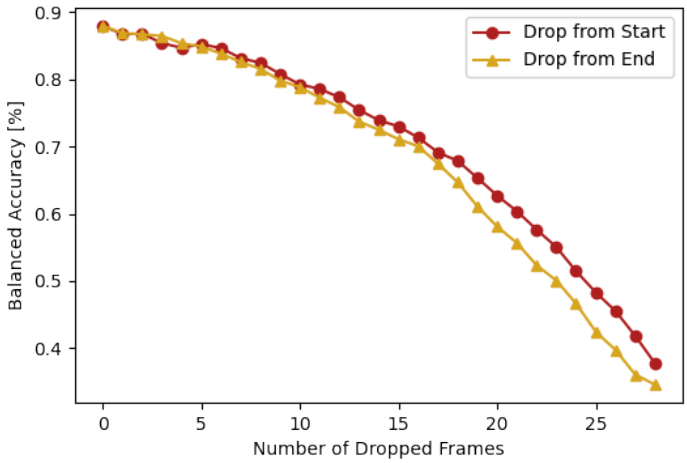


Figure 7.11: AcT-M balanced accuracy with an incremental reduction of temporal information. Due to the intrinsic nature of the network, it is possible to reduce the number of temporal steps without a retraining or any kind of explicit adaptation.

excluding MS-G3D, ST-TR, and AcT-L. In general, the results give similar insights. The tested models are the same as the previous case, with all the necessary modifications given by the different input formats. In the MLP case, however, performance seriously degrades as networks strongly tend to overfit input data after a small number of epochs, so the results are not included in Table 7.8. That is caused by the fact that PoseNet is a lighter methodology developed for Edge AI, and hence noisy and even missing keypoint detections are more frequent. That results in less informative data and emphasizes the difference between sequences belonging to different sub-datasets, confusing the model and inducing it to learn very specific but unusable features. The MLP is too simple and particularly prone to this kind of problem. Naturally, all the models are affected by the same problem, and the balanced accuracy on PoseNet is generally lower. The same considerations made for OpenPose apply in this case, where AcT outperforms all the other architectures and demonstrates its ability to give an accurate and robust representation of temporal correlations. Also, it is interesting to notice that Conv1D performs better than REMNet, proving to be less prone to overfitting, and that standard deviations are more significant than in the OpenPose case.

Model introspection

In order to have an insight into the frames of the sequence the AcT model attends to, we extract the self-attention weights at different stages of the network. In Fig.7.9, MPOSE2021 test samples are propagated through the AcT-M model, and attention weights $\boldsymbol{A}$ of the three distinct heads are presented. It can be seen that

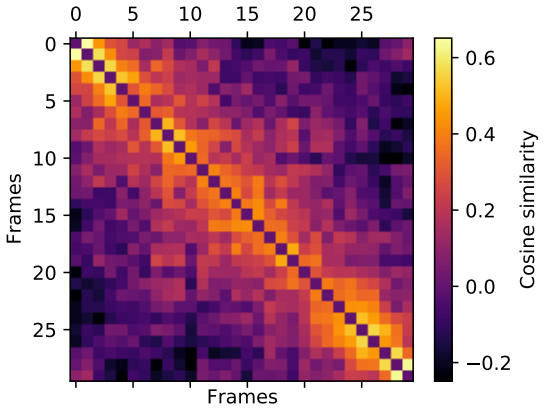


Figure 7.12: Cosine similarities of the learned T position embeddings of AcT-M model.

the model pays attention to specific frames of the sequence when a specific gesture defines the action. On the other hand, attention is much more spread across the different frames for more distributed actions such as walking and running. Moreover, it is clear how the three heads mostly focus on diverse frames of the sequence. Finally, the rightmost columns show three attention maps of failed predictions. In these last cases, attention weights are less coherent, and the model cannot extract a valid global representation of the scene.

Instead, in Fig.7.10, the last-layer self-attention scores of the [CLS] token are shown together with the RGB and skeleton representations of the scene. The scores are computed as the normalized sum of the three attention heads and give a direct insight into the frames exploited by AcT to produce the classification output. It can be seen that bent poses are much more informative for the model to predict the jumping-in-place action.

Moreover, we analyze the behavior of the network under a progressive reduction of temporal information. That can be easily done without retraining due to the intrinsic nature of AcT. In Fig.7.11, we present how the test-set balanced accuracy is affected by frame dropping. The two curves show a reduction starting from the beginning and the end of the temporal sequence, respectively. It is interesting to notice how the performance of AcT degrades with an almost linear trend. That highlights the robustness of the proposed methodology and demonstrates the possibility of adapting the model to applications with different temporal constraints.

Finally, we also study the positional embeddings of the AcT-M model by analyzing their cosine similarity, as shown in Fig.7.12. Very nearby position embeddings demonstrate a high level of similarity, and distant ones are orthogonal or in the opposite direction. This pattern is constant for all T frames of the sequence, highlighting how actions are not particularly localized and that relative positions are essential for all the frames.

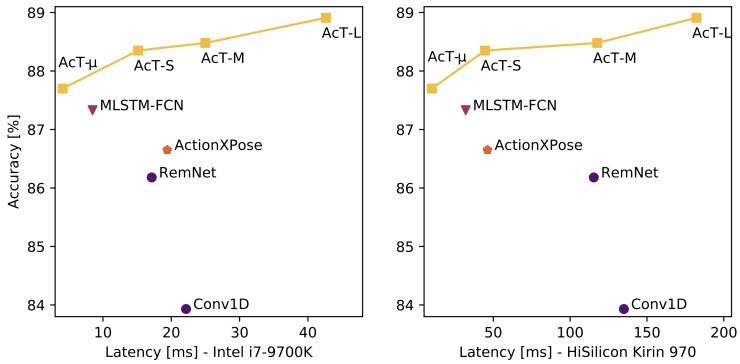


Figure 7.13: Study of the latency of different tested models on a high-performance Intel CPU and on a mobile phone equipped with an ARM-based CPU.

Latency measurements

We test the performance of all the considered models for real-time applications. To do so, we use the TFLite Benchmark[8] tool, which allows running TensorFlow Lite models on different computing systems and collecting statistical results on latency and memory usage. In our case, two CPUs are employed to measure model speed both on a PC and a mobile phone: an Intel i7-9700K for the former and the ARM-based HiSilicon Kirin 970 for the latter. In both experiments, the benchmark executes 10 warm-up runs followed by 100 consecutive forward passes, using 8 threads.

The results of both tests are reported in Fig. 7.13, where only the MLP has been ignored because, despite being the fastest-running model, its accuracy results are much lower than its competitors. The graph shows the great computational efficiency of Transformer-based architectures, whereas convolutional and recurrent networks result in heavier CPU usage. Indeed, in the Intel i7 case, REMNet achieves almost the same speed as AcT-S, but its accuracy is 2% lower. Moreover, AcT-μ is able to outperform REMNet, running at over four times its speed. MLSTM-FCN and ActionXPose, being smaller models, achieve lower latencies than the baselines: the former stays between AcT-μ and Act-S, while the latter performs similarly to AcT-S. Those results are remarkable but still outperformed by AcT-μ both on accuracy and speed.

The difference with the baselines is even more evident on the ARM-based chip, as convolutional architectures seem to perform poorly on this kind of hardware. Indeed, REMNet and Conv1D run as fast as AcT-M with significantly lower accuracies, and AcT-μ is ten times quicker. Nothing changes for what concerns MLSTM-FCN and ActionXPose, less accurate and three times slower than AcT-μ.

[8] https://www.tensorflow.org/lite/performance/measurement

MPOSE2021 Split		OpenPose 1		OpenPose 2		OpenPose 3	
Model	**Parameters**	**Accuracy [%]**	**Balanced [%]**	**Accuracy [%]**	**Balanced [%]**	**Accuracy [%]**	**Balanced [%]**
AcT-μ	227k	90.86 ± 0.36	86.86 ± 0.50	91.00 ± 0.24	85.01 ± 0.51	89.98 ± 0.47	87.63 ± 0.54
AcT-S	1,040k	91.21 ± 0.48	87.48 ± 0.76	91.23 ± 0.19	85.66 ± 0.58	90.90 ± 0.87	88.61 ± 0.73
AcT-M	2,740k	**91.38 ± 0.32**	**87.70 ± 0.47**	91.08 ± 0.48	85.18 ± 0.80	91.01 ± 0.57	88.63 ± 0.51
AcT-μ ⚗	227k	91.75 ± 0.25	87.26 ± 0.41	92.03 ± 0.34	86.11 ± 0.32	90.02 ± 0.37	88.14 ± 0.44
AcT-S ⚗	1,040k	91.11 ± 0.68	87.02 ± 0.86	91.43 ± 0.39	85.96 ± 0.27	90.95 ± 0.81	88.72 ± 0.63
AcT-M ⚗	2,740k	91.27 ± 0.22	87.55 ± 0.41	**91.88 ± 0.32**	**86.18 ± 0.70**	**92.01 ± 0.67**	**89.23 ± 0.71**

Table 7.9: Accuracy of models trained with and without knowledge distillation on MPOSE2021 splits using OpenPose 2D skeletal representations. Distillation models are denoted with alembic symbol, ⚗, and are trained with AcT-L as a teacher.

MPOSE2021 Split		PoseNet 1		PoseNet 2		PoseNet 3	
Model	**Parameters**	**Accuracy [%]**	**Balanced [%]**	**Accuracy [%]**	**Balanced [%]**	**Accuracy [%]**	**Balanced [%]**
AcT-μ	228k	86.66 ± 1,10	81.56 ± 1.60	87.21 ± 0.99	79.21 ± 1.60	87.75 ± 0.53	82.99 ± 0,87
AcT-S	1,042k	87.63 ± 0.52	82.54 ± 0.87	88.48 ± 0.57	81.53 ± 0.68	88.49 ± 0.65	83.63 ± 0.99
AcT-μ ⚗	228k	87.82 ± 0,91	82.47 ± 1.04	87.91 ± 0.49	79.91 ± 1.42	88.15 ± 0.43	83.18 ± 0,81
AcT-S ⚗	1,042k	**88.93 ± 0.42**	**83.84 ± 0.57**	**88.91 ± 0.67**	**82.67 ± 0.49**	**89.39 ± 0.55**	**84.63 ± 0.87**

Table 7.10: Accuracy of models trained with and without knowledge distillation on MPOSE2021 splits using PoseNet 2D skeletal representations. Distillation models are denoted with alembic symbol, ⚗, and are trained with AcT-M as a teacher.

Distilling knowledge for improving accuracy

Lastly, we apply knowledge distillation to boost the accuracy of small models further, levering the potential of larger ones. As introduced by Hinton et al. [99], knowledge distillation uses a pre-trained teacher's predictions to provide an auxiliary loss to the student model being trained. With data efficient transformer (DeiT) [238], Touvron et al. demonstrated the benefits of distillation applied to transformers, especially in the case of a small data regime. They proposed adding a second learnable distillation token to the input and fed a linear classifier with it. They tested either soft distillation, minimizing the Kullback-Leibler divergence between the softmax of the teacher and the softmax of the student model, and the hard-label distillation, where the cross-entropy of the softmax of the student and one-hot-encoding of the teacher is minimized. Regarding this second distillation method, let Z_{sc} be the logits of the student class token and Z_{sd}, Z_t the logits of the student distillation token and teacher model, respectively. Therefore, the overall loss to be minimized is:

$$\mathcal{L} = \frac{1}{2}\mathcal{L}_{CE}(\phi(Z_{sc}), y) + \frac{1}{2}\mathcal{L}_{CE}(\phi(Z_{sd}), y_t) \tag{7.14}$$

where $\mathcal{L}_{CE}$ is the cross-entropy loss, ϕ is the softmax function and $y_t = \text{argmax}(Z_t)$. Finally, the class and distillation token predictions are averaged to produce a single prediction at inference time.

In Table 7.9 and 7.10 are reported accuracy results on MPOSE2021 using OpenPose 2D and PoseNet 2D skeletal representations, respectively. In both cases, the larger pre-trained AcT version is used as a teacher. It is noticeable how knowledge distillation can further improve the accuracy of small models while preserving all latency a power efficiency of them.

Part III

Chapter 8

Generalization in Deep Learning: Background and Concepts

The problem of induction has a central role within the learning process. Without generalization, machine learning algorithms would exhibit useful behaviors only in situations identical to the ones previously experienced [241]. Deep neural networks are powerful models capable of extracting subtle regularities from training data and have had a profound impact on the conceptual bases of ML and AI [125]. Indeed, neural networks show far more expressiveness with respect to classical ML models, having universality [145] and exponential advantages over hand-crafted features [17]. Moreover, further theoretical research has presented how trainable deep hypothesis spaces reveal structural properties that may enable non-convex optimization [48]. Nevertheless, the expressiveness and trainability of the hypothesis space do not guarantee good performance in predicting novel data points because over-fitting is a significant source of concern. That is especially true in the case of DL, in which models are usually overparameterized concerning the training data[1] and the variance error component could be determinant. Indeed, several works in the literature [265, 12] have pointed out that deep hypothesis spaces have sufficient capacity and degree of freedom to memorize random datasets of the dimension of ImageNet1k [211]. That leads to the fundamental study and analysis of generalization. The rest of this chapter are firstly presented background information on the two main facets of generalization discussed in the literature:

- Generalization gap produced by the difference between train and test error of a function optimized over a defined source domain. That is the result of minimizing a non-computable expected risk. Trading off variance and bias are the two components to control this generalization error.

[1] It is interesting to notice that human brains are also largely overparameterized concerning the average life expectancy. 150.000 billion synapses and barely 2.5 billion seconds of life.

- Generalization gap caused by testing a function over a target domain different from the source training domain.

Finally, we conclude with a high-level framework that uses the notions of equivariant and invariant functions to bring both generalization aspects together.

8.1 Non-computable expected risk

Given the input random variable X with values $x \in \mathcal{X}$, the target random variable Y with values $y \in \mathcal{Y}$, and $\mathcal{L}$ a generic loss function, it is possible to define $\mathcal{R}[f]$ the expected risk of a function f as $\mathcal{R}[f] = \mathbb{E}_{x,y\sim P(X,Y)}[\mathcal{L}(f(x), y)]$, where $P(X, Y)$ is the joint probability distribution P_{XY}, or $P(X, Y)$, over $\mathcal{X}\mathrm{x}\mathcal{Y}$. On the other hand, let $f_{\mathcal{E}(\mathcal{D})} : \mathcal{X} \to \mathcal{Y}$ be a model learned by a learning algorithm $\mathcal{E}$ applied to a training set $\mathcal{D} = (x_i, y_i)_{i=1}^N$ of size N. The algorithm $\mathcal{E}$ is the conjunction of the architecture $\mathcal{A}$ and the optimization procedure $\mathcal{T}$. Since $\mathcal{T}$ is stochastic, $f_{\mathcal{E}(\mathbb{D})}$ is a set $\mathbb{F}$ composed by all hypothesis reachable by $\mathcal{E}$ applied to $\mathcal{D}$ such that $\mathbb{F} = \{f \mid \exists \mathcal{D} \text{ such that} f \in \mathcal{E}(\mathcal{D})\}$. Thus, let $\mathcal{R}_{\mathcal{D}}[f]$ be the empirical risk of f as $\mathcal{R}_{\mathcal{D}}[f] = \frac{1}{m}\sum_{i=1}^N \mathcal{L}(f(x_i, y_i)$. Finally, let define $\mathcal{L}_{\mathbb{F}}$ be a family of loss functions associated with $\mathbb{F}$ such that $\mathcal{L}_{\mathbb{F}} = \{g : f \in \mathcal{F}, g(x, y) = \mathcal{L}(f(x), y)\}$.

The typical objective in a machine learning framework is the minimization of the expected risk $\mathcal{R}[f_{\mathcal{E}(\mathcal{D})}]$. However, the joint probability distribution P_{XY} is generally non available and consequently $\mathcal{R}[f_{\mathcal{E}(\mathcal{D})}]$ is non-computable. Thus, the problem is repurposed in the minimization of the related computable empirical risk $\mathcal{R}_{\mathcal{D}}[f_{\mathcal{E}(\mathcal{D})}]$, also known as empirical risk minimization (ERM). Therefore, a fundamental problem of generalization is to justify that minimizing $\mathcal{R}_{\mathcal{D}}[f_{\mathcal{E}(\mathcal{D})}]$ is consistent with the minimization of $\mathcal{R}[f_{\mathcal{E}(\mathcal{D})}]$. In the literature are discussed different approaches to handle this dependence and decoupling $f_{\mathcal{E}(\mathcal{D})}$ from the particular $\mathcal{D}$. For instance, hypothesis-space complexity considers the worst-case gap for functions in the hypothesis space as:

$$\mathcal{R}[f_{\mathcal{E}(\mathcal{D})}] - \mathcal{R}_{\mathcal{D}}[f_{\mathcal{E}(\mathcal{D})}] \leq \sup_{f\in\mathcal{F}} \mathcal{R}[f] - \mathcal{R}_{\mathcal{D}}[f] \tag{8.1}$$

and by carefully analyzing the right-hand side. However, usually the cardinality of $\mathbb{F}$ is infinite and a direct use of the union bound over all elements in $\mathbb{F}$ yields a vacuous bound [125], leading to the need to consider different quantities to characterize $\mathbb{F}$. Thus, approaches on the Rademacher complexity [176], Vapnik-Chervonenkis dimension [27], stability [29] and robustness aims [259] at more tightly characterize this bound.

On the other hand, the non-computability of the estimating risk has led to the classical framework of generalization that aims at evaluating the generalization gap $\mathcal{R}[f_{\mathcal{E}(\mathcal{D})}] - \mathcal{R}_{\mathcal{D}}[f_{\mathcal{E}(\mathcal{D})}]$ using a subset $\mathcal{D}_t$ of the available train set $\mathcal{D}_{tr}$ ($\mathcal{D} = \mathcal{D}_{tr} + \mathcal{D}_t$)

to compute a test error $\mathcal{R}_{\mathcal{D}_t}[f_{\mathcal{E}(\mathcal{D}_{tr})}]$. Indeed, depending on the specific hypothesis space, the process of minimization over $\mathcal{R}_{\mathcal{D}_{tr}}[f_{\mathcal{E}(\mathcal{D}_{tr})}]$ could introduce high variability, resulting in a significant generalization gap. So, the classical framework of generalization decomposes the test error of a trained model f as:

$$\mathcal{R}_{\mathcal{D}_t}[f_{\mathcal{E}(\mathcal{D}_{tr})}] = \mathcal{R}_{\mathcal{D}_{tr}}[f_{\mathcal{E}(\mathcal{D}_{tr})}] + \underbrace{[\mathcal{R}_{\mathcal{D}_t}[f_{\mathcal{E}(\mathcal{D}_{tr})}] - \mathcal{R}_{\mathcal{D}_{tr}}[f_{\mathcal{E}(\mathcal{D}_{tr})}]]}_{\text{Generalization gap}} \tag{8.2}$$

and analyses each part separately [245, 27, 218].

However, deep hypothesis spaces pose some challenging all proposed classical framework. Indeed, modern neural networks can easily interpolate, reaching a $\mathcal{R}_{\mathcal{D}_{tr}}[f_{\mathcal{E}(\mathcal{D}_{tr})}] \approx 0$ and collapsing equation (8.2). In this regard, some works [182] tries to differently define equation (8.2) introducing the notion of a "Real World" model f^{rl}, trained bootstrapping a finite source of data (e.g. $\mathcal{D}_{tr}$), and of a "Ideal World" model f^{iid} obtained directly optimizing on the original population. Thus, in this framework equation (8.2) can be rewritten as

$$\mathcal{R}_{\mathcal{D}_t}[f^{rl}] = \underbrace{\mathcal{R}_{\mathcal{D}_t}[f^{iid}]}_{\text{A: Online learning}} + \underbrace{[\mathcal{R}_{\mathcal{D}_t}[f^{rl}] - \mathcal{R}_{\mathcal{D}_t}[f^{iid}]]}_{\text{B: Bootstrap error}} \tag{8.3}$$

where two different components define the error $\mathcal{R}_{\mathcal{D}_t}[f^{rl}]$. The first component takes into account how quickly models optimize the population loss in the infinite-data regime. On the other hand, the second contribution defines how closely models behave in the finite-data vs. infinite-data regime. The presented decomposition results are actually useful because, from empirical evidence, the bootstrap error is often very small. That implies that the performance of a real classifier is largely captured by its performance in the "Ideal World," reducing the problem to a question in online stochastic optimization. The discussion on this theme is further elaborated in Appendix B, introducing a novel ML project that moves from bootstrap learning to plain online learning.

8.2 Beyond the source population: domain generalization

Domain Generalization (DG) aims at training models able to generalize to out-of-distribution (OOD) data at the same problem. The access to a set of source datasets provides a predictor with the ability to extract and learn general invariant patterns, which are, hypothetically, also recognizable in the target domain dataset [25]. As an extension of supervised learning, this approach minimizes empirical risk at training time to extrapolate an overall probability distribution from source datasets that enables accurate OOD data classification.

DG has a higher degree of difficulty regarding the generalization related to ERM. Indeed, in DG a set of different K source domains $\mathbb{S} = (S_k)_{k=1}^{K}$ is used to learn a classifier $f_{\mathcal{E}(\mathcal{S})}$ that aims at generalizing well on an unknown target domain T. Each source domain is associated with its joint probability distribution $P_{(XY}^{k}$, whereas $P_{XY}^{\mathbb{S}}$ indicates the overall source distribution learned by the classifier [271]. Indeed, the goal of DG is to enable the classifier to predict well on out-of-distribution data, namely on the target domain distribution P_{XY}^{T}, by learning an overall domain invariant distribution from the source domains seen during training. Thus, DG aims at reducing the domain generalization gap $\mathcal{R}_{\mathcal{S}_t}[f_{\mathcal{E}(\mathcal{S}_{tr})}] - \mathcal{R}_T[f_{\mathcal{E}(\mathcal{S}_{tr})}]$ where $\mathcal{S}_{tr}$ and $\mathcal{S}_t$ are the source training and testing dataset, respectively.

It is clear how the DG gap could be significantly affected by the distance of the source and target distributions. Therefore, it is essential to define metrics that measure the distance between the two distributions. Maximum Mean Discrepancy (MMD) is a statistical test originally proposed in [88] and is one of the metrics most widely adopted to compute the distance between two distributions. MMD is also primarily used in domain adaptation since it perfectly fits the need to understand whether the source and the target domain extracted features overlap. Formally, MMD is a kernel-based difference between feature means. Given a set of m samples X with a probability measure P, the feature mean can be expressed as:

$$\mu_p\ (\phi(X)) = [E[\phi(X_1], \cdots, E[\phi(X_m]]^T \tag{8.4}$$

where $\phi(X)$ is the feature map that maps X to a new feature space $\mathcal{Z}$. If it satisfies the necessary theoretical conditions, a kernel based approach can be used to compute the inner product of two distributions of samples $X \sim P$ and $Y \sim Q$:

$$\langle \mu_P\ (\phi(X), \mu_Q\ (\phi(Y)\rangle_{\mathcal{F}} = E_{P,Q}\ [\langle \phi(X), \phi(Y)\rangle_{\mathcal{Z}}] = E_{P,Q}\ [k(X,Y)] \tag{8.5}$$

At this point the MMD can be defined as the distance between the feature means of $X \sim P$ and $Y \sim Q$:

$$MMD^2(P,Q) = \|\mu_P - \mu_Q\|_{\mathcal{Z}}^2 \tag{8.6}$$

which can be expressed more in detail by using Equation (8.5):

$$MMD^2(P,Q) = E_P\ [k(X,X)] - 2E_{P,Q}\ [k(X,Y)] + E_Q\ [k(Y,Y)] \tag{8.7}$$

However, an empirical estimate of MMD needs to be computed since only samples are available in a real case instead of the explicit formulation of the distributions. It is possible to obtain the MMD expression by considering the empirical estimates of the feature means based on their samples:

$$\begin{aligned} MMD^2(X,Y) = \frac{1}{m(m-1)} \sum_i \sum_{j \neq i} k(\mathbf{x_i}, \mathbf{x_j}) \\ -2\frac{1}{m.m} \sum_i \sum_j k(\mathbf{x_i}, \mathbf{y_j}) + \frac{1}{m(m-1)} \sum_i \sum_{j \neq i} k(\mathbf{y_i}, \mathbf{y_j}) \end{aligned} \tag{8.8}$$

Where $\mathbf{x_i}$ and $\mathbf{y_i}$ in this case are the image samples from source, and target domains, m is the number of samples of the considered subsets. Finally, we specifically use a gaussian kernel with the following expression:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp\left(\frac{-\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma^2}\right) = \exp\left(\frac{-1}{\sigma^2}[\mathbf{x_i}^\intercal \mathbf{x_i} - 2\mathbf{x_i}^\intercal \mathbf{x_j} + \mathbf{x_j}^\intercal \mathbf{x_j}]\right) \tag{8.9}$$

In the last decade, aware of the tremendous impact of generalization on computer vision applications, the DG research community has tackled the problem with algorithms that primarily aim at finding invariances that hold with novel domains. Among the constellation of proposed approaches, we identify the principal broad strategies adopted for domain generalization in augmenting the source domain [219], aligning domain distributions [79], meta-learning [15], self-supervised learning [7], and regularization strategies [213].

8.3 The equivariance and invariance framework

The ML and DL communities have mainly tackled classical and OOD generalization as two different problems. Nevertheless, both aspects of generalization can be unified by a high-level framework that considers the factor of variations as the source of different distributions. Indeed, in the case of "natural" distributions, different P^k_{XY} for the same problem differ for a number of transformations that determine the actual distance between the distributions. Therefore, we can define a set $\mathbb{T}$ that contains $k \times k$ number of transformations $\mathbb{T} = \{\mathrm{T}_{i,j}\}$ where $i, j \in [0, \ldots, k]$ and $i, j \in \mathbb{N}$. Each transformation of the set $\mathbb{T}$ can be applied to a certain probability distribution i-th to project data from the i-th to j-th joint probability distribution $\mathrm{T}_{ij}(P^i_{XY}) = P^j_{XY}$. Under this framework, the distance between two distributions, $\mathcal{M}$, is determined by the transformation $\mathrm{T}_{i,j}$ implied in the generation of the distribution P^j_{XY}. Consequently, there are transformations that maintain a close distance from the origin distribution, such as affine transformations (e.g., rotations, translations) and transformations that create a considerable gap between the two distributions involved (e.g., style transformations).

The given high-level definitions of transformations set $\mathbb{T}$ and joint probability distributions $P^k_X Y$ can be used to guide the search of $\mathcal{A}$. However, we need to introduce some terminology from group theory, particularly the notions of equivariant and invariant functions.

Definition VII.1 (Equivariance). *A function $f : X \to Y$ is said to be equivariant to the actions g* of a group $\mathcal{G}$ if

$$f(g \circ x) = g \circ f(x) \qquad \textit{for all } x \in X, g \in \mathcal{G}$$

Definition VII.2 (Invariance). *A function $f : X \to Y$ is said to be invariant to the actions g* of a group $\mathcal{G}$ if

$$f(g \circ x) = f(x) \qquad \textit{for all } x \in X, g \in \mathcal{G}$$

Therefore, an ideal architecture to effectively generalize a problem should be invariant to different transformations. Nevertheless, a function f invariant to each transformation T_{ij} would not be able to distinguish from different factors of variations. Indeed, that is a significant limitation that would not allow extrapolating the right semantic from the data. On the other hand, an ideal architecture should first be able to disentangle all variation factors and be equivariant to them. It should generate internal representations z at different hierarchical levels that would transform accordingly to the input. On the other hand, a second smaller part of the model attached to the different levels should be invariant to these representations. A much lighter and volatile network can easily map z variations to the right space. That would lead to a more robust deep learning model with more generalization power.

Chapter 9

Investigating Architectural Priors for a Robust and Efficient Perception

This chapter is discussed two different contributions to generalization for perception. Indeed, sensory perception is mostly revolutionized by DL techniques but still struggles with systematic generalization. Either under transformations $\mathrm{T}_{i,j}$ with a limited impact on the source joint probability distribution P^i_{XY}, Figure 9.1, or transformations that create a considerable distribution shift, DL models performances can be strongly affected. Priors in the architectural structure $\mathcal{A}$ or differences in data $\mathcal{D}$ and training procedure $\mathcal{T}$ are the three factors that can introduce a certain degree of variability to the generalization problem. Indeed, as explained in the previous chapter, a learned function $f_{\mathcal{E}(\mathbb{D})}$ is the result of a learning algorithm $\mathcal{E}$ applied to $\mathbb{D}$. Moreover, $\mathcal{E}$ is the conjunction of the architecture $\mathcal{A}$ and the optimization procedure $\mathcal{T}$, making them tightly bound and difficult to disentangle. Thus, it is always possible to perform a separate analysis of the different factors, but it is crucial to keep in mind this dependence.

In the first case, in which transformations have a limited impact on the source probability distribution, affine transformations are an exemplary example: translations, rotations, reflections are only some of the transformations that should not influence the outcome of a network. Convolutions are a perfect example of how it is possible to achieve translation equivariance, endowing the network to recognize an object in different parts of the field of view. Nevertheless, the same architectural prior does not contribute to all other transformations. Indeed, if not present in some form inside the training distribution, all other transformations can significantly affect the robustness of the network. Therefore, in order to provide invariance to these transformations, researchers make extensive usage of data augmentation [221], generating additional feature detectors for all transformations. However, even if residual architectures [93] and regularization techniques have made

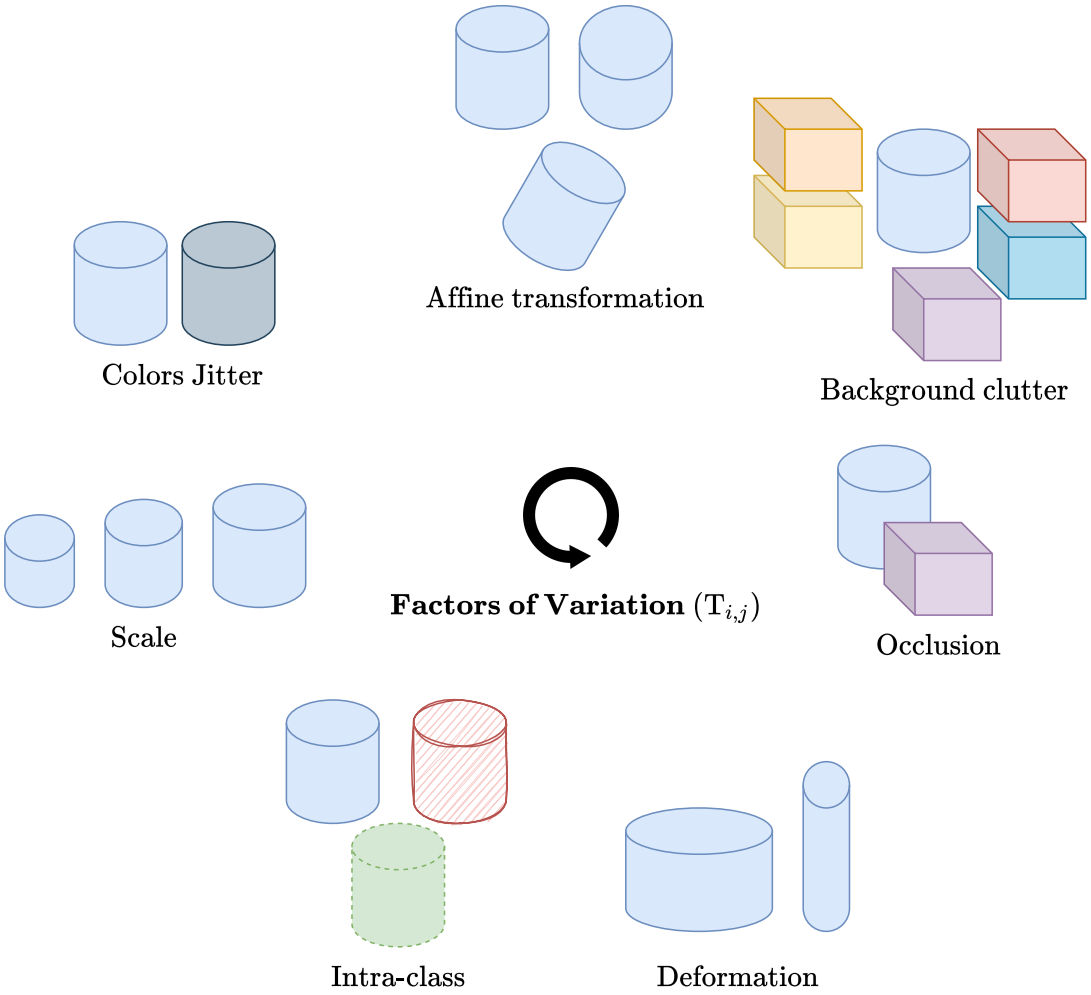


Figure 9.1: A visual perception model needs to be robust to different factors of variation. The blue cylinder is the training target.

it possible to train larger and deeper neural networks, present methodologies would difficultly scale to animal-level perception. Indeed, even if there are research works claiming super-human-level performance on some dataset [94], the reality is that we are far from systems that show similar performance outside their "comfort zone" dataset. Moreover, invariance to transformations prevents models from generating different representations, making them unable to distinguish different factors of variations from each other. Conversely, a proper functioning architecture should demonstrate equivariance to all transformations, leveraging invariances in the systems physic (e.g., transformations between reference frames of objects parts). On the other hand, transformations $T_{i,j}$ that produce a significant distribution shift between source and target have an even bigger impact on networks' performance. Similarly to the last case, being invariant to all new possible domains seems a

good solution, but it could difficultly scale to animal-level perception and human abstraction capabilities. Nevertheless, the extent of $\mathcal{D}$, $\mathcal{T}$, or $\mathcal{A}$ is not clear for OOD. However, following the equivariance and invariance framework, a hypothetical architecture should aim to generate disentangled representations of all factors of variations that would potentially generalize better and create later connections in the multi-dimensional space.

In this context, the two presented contributions investigate robustness to transformations with respect to visual perception. Nevertheless, all concepts are not limited to this domain but only glimpse how architectural priors could intensely work towards systematic generalization. Both works primarily focus on the importance of architectural priors: they set aside training procedures and the importance of data but without ignoring their strict bound. On the other hand, Appendix B discusses a preliminary research project that focuses on data $\mathcal{D}$ and learning algorithms $\mathcal{T}$ for visual perception generalization.

The first contribution of this chapter discusses a novel architecture that exploits reference frames between parts and whole of objects to be efficient and robust to affine transformations [169]. It builds over the original Geoffrey Hinton idea [212], presenting a lightweight model that incorporates a self-attention mechanism [246] to deliver the same performance with only 2% of the original capsule model parameters [212]. On the contrary, the second work further investigates the role of architectures in visual perception in the presence of significant distribution shifts. We demonstrate how the out-of-distribution task is strictly bounded with an overall comprehensive view of generalization, supporting the high-level idea of the equivariance and invariance framework presented in the previous chapter.

9.1 Efficient-CapsNet: toward efficiency and robustness to affine transformations

In the last decade, CNNs drastically changed artificial visual perception, achieving remarkable results in all core fields of computer vision, from image classification [93, 106] to object detection [201, 154, 172] and instance segmentation [95]. In contrast to other DNNs, the main characteristic of a CNN is its capability to efficiently replicate the same knowledge at all locations in the spatial dimension of an input image. Indeed, using translated replicas of learned feature detectors, features learned at one spatial location are available at other locations. Local shared connectivity coupled with spatial reduction layers, such as max-pooling, extract local translation-invariant features. So, as shown in Figure 9.2, object translations in the input space do not affect high-level activations of because max-pooling layers can rout low-level features between the layers. Nevertheless, translation invariance achieved by CNN comes at the expense of losing the precise encoding of objects' locations. Moreover, CNNs are not invariant to all other affine transformations.

Over the years, different techniques have been developed to counterbalance that problem. Most of the adopted common solutions leverages an increased number of features in such a way that the network is endowed with enough detectors for all additional transformations. Data augmentation techniques are used to produce different poses to be learned [221], and residual connections and normalization techniques [114] allow to enlarge networks filter capacity [93]. However, all those additional mechanisms only partially make up for the intrinsic limitations of CNN. Indeed, CNNs trained on large datasets have a massive redundancy of detectors and difficulties scaling to thousands of objects with their respective viewpoints.

Hinton et al.[100] proposed to make neurons cooperate in a new form of unit, dubbed capsules, where individual activations inside them do not represent the presence of a specific feature but different properties of the same entity anymore. In their paper, they showed that groups of neurons, if adequately trained, can produce a whole vector of numbers, explicitly representing the pose of the detected entity as in classical hand-engineered features[160]. After six years, Sabour et al. [212] presented a first architecture, named CapsNet, that introduced capsules inside a CNN. The central insight of the paper is that viewpoint changes have complicated effects on the pixel space, but simple linear effects on the pose represent the relationship between an object-part and the whole. Exploiting groups of neuron activations to make predictions and assess their reciprocal agreement is a much more effective way to capture covariance and should lead to models with a considerably reduced number of parameters and far better generalization capabilities.

Nevertheless, little attention has been given to the efficiency aspect of capsule networks and their intrinsic capability to represent knowledge object transformations better. Indeed, all model solutions presented so far account for a large number of parameters that inevitably hide the intrinsic generalization capability that capsules should provide. Therefore, we propose Efficient-CapsNet, an extreme architecture with barely 160K parameters and a 85% TOPs improvement upon the original CapsNet model that is perfectly capable of achieving state-of-the-art results on three distinct datasets, maintaining all critical aspects of capsule networks. With extensive experimentation with traditional CNNs and other capsule implementations, we proved the effectiveness of our methodology and the important contribution lead by capsules inside a network. Moreover, we propose a novel non-iterative routing algorithm that can easily cope with a reduced number of capsules exploiting a self-attention mechanism. Indeed, attention, as also max-pooling layers, can be seen as a way to route information inside a network. Our proposed solution exploits similarities between low-level capsules to cluster and routs them to more promising high-level capsules. All of our training and testing code are open source and publicly available[1].

[1] https://github.com/EscVM/Efficient-CapsNet

9.1.1 Methodology

The overall architecture of Efficient-CapsNet is depicted in Figure 9.3. As a high-level description, the network can be broadly divided into three different parts in which the first two are the main instruments of the primary capsule layer to interact with the input space. Indeed, each capsule exploits the below convolutional layer filters to convert pixel intensities into a vectorial representation of the feature it acts for. So, the activities of neurons within an active capsule embody the various properties of the entity it learnt to represent during the training process. As stated in Sabour et al. [212], these properties can include many different types of instantiation parameter such as pose, texture, deformation, and among those the existence of the feature itself. In our implementation, the length of each vector is used to represent the probability that the entity represented by a capsule is present. That is compatible with our self-attention routing algorithm that does not require any sensible objective function minimization. Moreover, it makes biological sense as it does not use large activities to represent absent entities. Finally, the last part of the network operates under the self-attention algorithm to rout low-level capsules to the whole they represent.

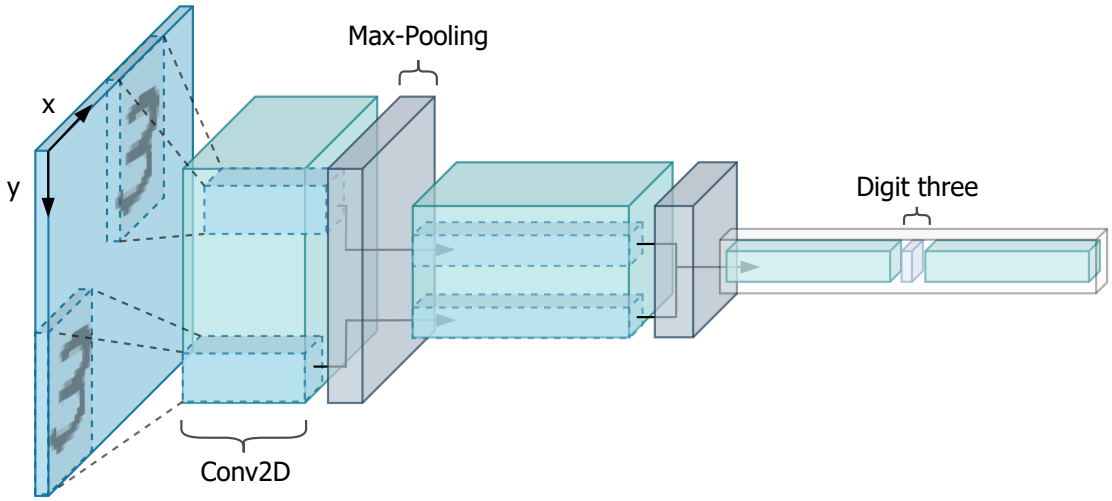


Figure 9.2: Compressed representation of a simple CNN with max-pooling layers for spatial reduction and two input objects obtained with a plain spatial translation. Max-pooling operations are schematized in such a way that their primitive routing role is highlighted for both digits. Low-level features detected in the earlier stage of the network are progressively routed to common high-level features. So, the model is translation invariant but gradually loses relevant object localization information.

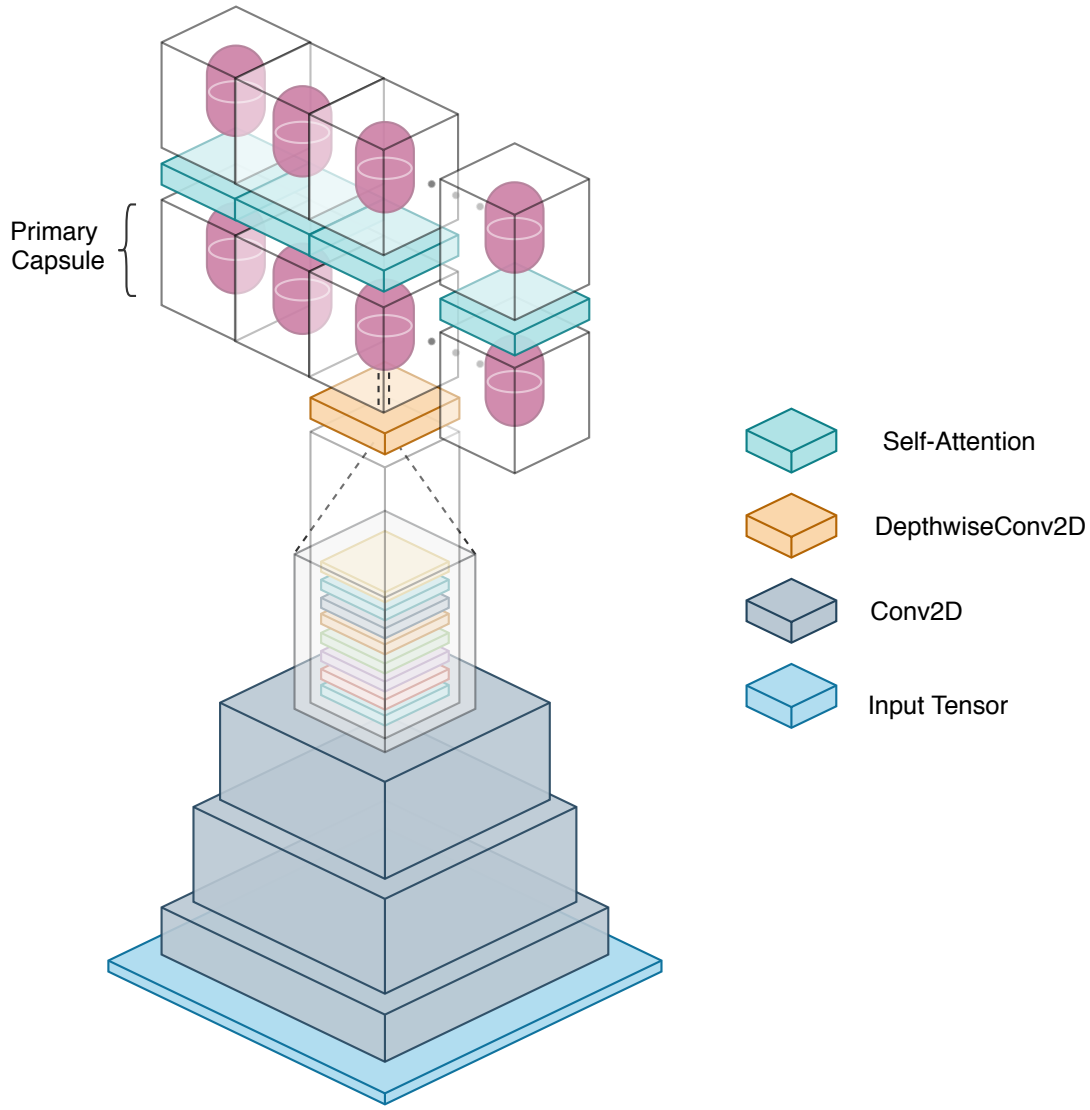


Figure 9.3: Schematic representation of the overall architecture of Efficient-CapsNet. Primary capsules make use of depthwise separable convolution to create a vectorial representation of the features they represent. On the other hand, the first stack of convolutional layers maps the input tensor onto a higher-dimensional space, facilitating capsules creation.

Network Architecture

More in detail, in the case of a single instance (i), the model takes as input an image that can be represented as a tensor $\mathbf{X}$ with a shape $H \times W \times F$ where H, W and C are the height, width, and channels/features of the single input image. Before entering the primary caps layer, we extract local features from the input image $\mathbf{X}$ by means of a set of convolutional and Batch Normalization layers [114].

Each output of a convolution layer l is constituted by a convolutional operation with a certain kernel dimension k, number of feature maps f, stride $s = 1$ and ReLU as activation function:

$$\mathrm{F}^{l+1}(\mathbf{X}^l) = \mathrm{ReLU}\left(Conv_{k\times k}(\mathbf{X}^l)\right) \tag{9.1}$$

Overall, the first convolutional part of the network can be modelled as a single function H_{Conv} that maps the input image onto a higher dimensional space that facilitates the capsule creation. On the other hand, the second part of the network is the main instrument used by primary capsules to create a vectorial representation of the features they represent. It is a depthwise separable convolution with linear activation that performs just the first step of a depthwise spatial convolution operation, acting separately on each channel. Moreover, imposing a kernel dimension $k \times k$ and a number of filters f equal to the output dimensions $H \times W$ and F of the H_{Conv} function, it is possible to obtain the primary capsule layer $\boldsymbol{S}^l_{n,d}$ where n^l and d^l are the number of primary capsules and their individual dimension of the $l-th$ layer, respectively. The depthwise separable convolution is an efficient operation that greatly simplifies and reduces the number of parameters required for the capsule creation process. We leave it to discriminative learning to make good use of its filters to smartly extract all capsule properties.

After that operation, location information is not anymore "place-coded" but "rate-coded" in the properties of the capsules. So, the base element of the network is not anymore a single neuron but a vector-output capsule. Indeed, the first operation applied to the primary capsule layer is a capsule-wise activation function. In order to encode the probability that a certain entity exists with the length of vectors and let active capsules make predictions for the instantiation parameters of higher-level capsules, two important properties should be satisfied by the activation function; it should preserve a vector orientation and maintain the length between zero and one. Efficient-CapsNet makes use of a variant of the original activation function, dubbed squash operation:

$$\mathrm{squash}(\boldsymbol{s}^l_n) = \left(1 - \frac{1}{e^{||\boldsymbol{s}^l_n||}}\right)\frac{\boldsymbol{s}^l_n}{||\boldsymbol{s}^l_n||} \tag{9.2}$$

where we refer to a single capsule as $\boldsymbol{s}^l_n$, which are the individual entries n^l of $\boldsymbol{S}^l_{(n,:)}$[2] with $\boldsymbol{s}^l_n \in \mathbb{R}^{d^l}$. The capsule-wise squash function of Eq. (9.2), satisfies the required two properties and is much more sensitive to small changes near zero, providing a boost to the gradient during the training phase [257]. So, after the squash activation we obtain a new matrix $\boldsymbol{U}^l_{n,d}$ with all n^l entries $\boldsymbol{u}^l_n$ with the same dimensionality and properties of $\boldsymbol{s}^l_n$, but with a length "squashed" between zero and one. Indeed

[2] $\boldsymbol{s}^l_{n_0} := \{\boldsymbol{S}^l_{n,d} | n^l = n^l_0\}$

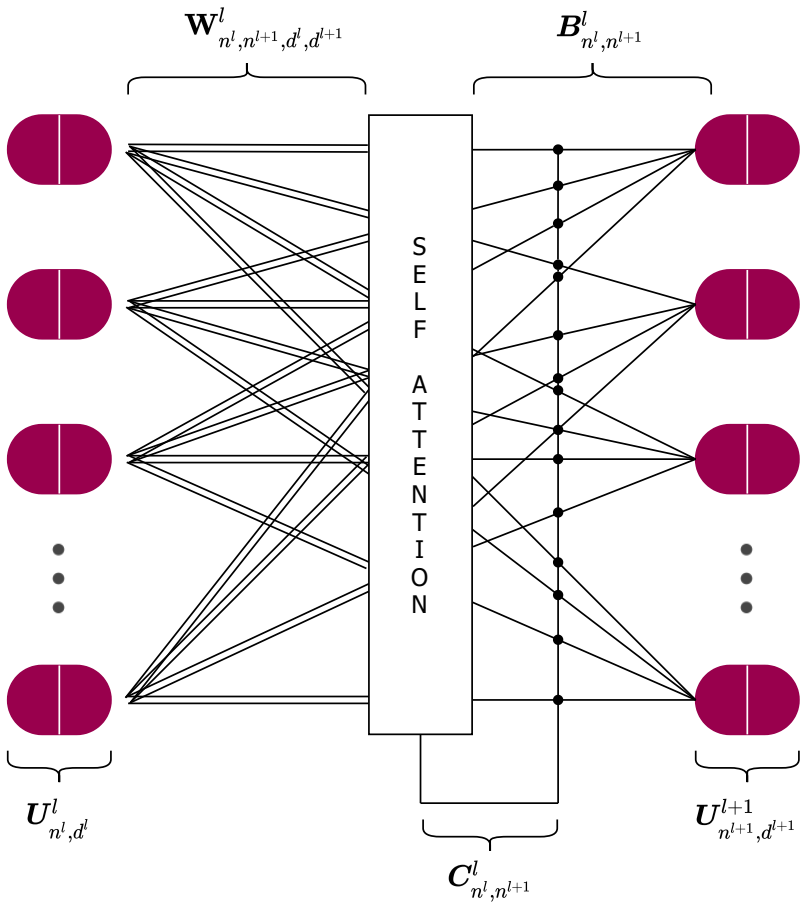


Figure 9.4: Capsules of the layer $l-th$ make predictions of the whole they could be part of. All predictions obtained with the weight tensor $\mathbf{W}^l_{n^l,n^{l+1},d^l,d^{l+1}}$ are collected in $\hat{\mathbf{U}}^l_{n^l,n^{l+1},d^{l+1}}$ that is subsequently used in conjunction with the priors $\boldsymbol{B}^l_{n^l,n^{l+1}}$ and coupling coefficients $\boldsymbol{C}^l_{n^l,n^{l+1}}$ matrices to obtain all capsules $\boldsymbol{s}^{l+1}_n$ of layer $l+1$.

the non-linearity ensure that short vectors get shrunk to almost zero length and long vectors get shrunk to a length slightly below one.

Self-Attention routing

In order to rout active capsules to the whole they belong, we make use of our self-attention routing algorithm. As shown in Figure 9.4, despite the additional dimension, the overall architecture is very similar to a fully-connected network with an additional branch brought by the self-attention algorithm. Indeed, the total input of a capsule in the above layer, $\boldsymbol{s}^{l+1}_n$, is a weighted sum over all "prediction vectors" from the capsules $\boldsymbol{u}^l_n$ in the layer below. That is produced by a matrix

multiplication of each capsule, $\boldsymbol{u}_n^l$, belonging to $\boldsymbol{U}_{n,d}^l$, for a weight matrix. Intuitively, the whole tensor $\mathbf{W}_{n^l,n^{l+1},d^l,d^{l+1}}^l$ that contains all weight matrices, embeds all affine transformation between capsule of two adjacent layers. So, each capsule of the layer l, in order to make its projections for the layer above, follows Eq. 9.3

$$\hat{\mathbf{U}}_{(n^l,n^{l+1},:)}^l = \boldsymbol{u}_n^{\mathrm{T}l} \times \mathbf{W}_{(n^l,n^{l+1},:,:)}^l \tag{9.3}$$

where $\hat{\mathbf{U}}_{n^l,n^{l+1},d^{l+1}}^l$ contains all predictions of $l-th$ capsules. Indeed, each n^l capsule, by means of the weight matrix, predicts the properties of all n^{l+1} capsules. Indeed, capsules of the above layer, $\boldsymbol{s}_n^{l+1}$, can be computed with Eq. 9.4

$$\boldsymbol{s}_n^{l+1} = \hat{\mathbf{U}}_{(:,n^{l+1},:)}^{\mathrm{T}l} \times \left(\boldsymbol{C}_{(:,n^{l+1})}^l + \boldsymbol{B}_{(:,n^{l+1})}^l\right) \tag{9.4}$$

where $\boldsymbol{B}_{n^l,n^{l+1}}^l$ is the log priors matrix containing all weights discriminatively learnt at the same time as all the other weights. On the other hand, $\boldsymbol{C}_{n^l,n^{l+1}}^l$ is the matrix containing all coupling coefficients produced by the self-attention algorithm. So, the priors help to create biases towards more linked capsules and the self-attention routing dynamically assigns detected shapes to the whole they represent in the specific (i) instance taken into account. The coupling coefficients are computed starting from the self-attention tensor $\mathbf{A}_{n^l,n^l,n^{l+1}}^l$ using Eq. 9.5

$$\mathbf{A}_{(:,:,n^{l+1})}^l = \frac{\hat{\mathbf{U}}_{(:,n^{l+1},:)}^l \times \hat{\mathbf{U}}_{(:,n^{l+1},:)}^{\mathrm{T}l}}{\sqrt{d^l}} \tag{9.5}$$

which contains a symmetric matrix $\mathbf{A}_{:,:,n^{l+1}}^l$ for each capsule n^{l+1} of the layer above. The term $\sqrt{d^l}$ stabilizes training and helps maintaining a balance between coupling coefficients and log priors. Each self-attention matrix contains the score agreement for each combination of the n^l capsules predictions, and so, they can be used to compute all coupling coefficients. In particular, Eq.9.6 is used to compute the final coefficients that can be used in Eq. 9.4 to obtain all capsules $\boldsymbol{S}_{n,d}^{l+1}$ of the layer $l+1$.

$$\boldsymbol{C}_{(:,n^{l+1})}^l = \frac{exp\left(\sum_{n^l} \mathbf{A}_{(:,n^l,n^{l+1})}^l\right)}{\sum_{n^{l+1}} exp\left(\sum_{n^l} \mathbf{A}_{(:,n^l,n^{l+1})}^l\right)} \tag{9.6}$$

So, the coupling coefficients between a capsule of layer l and all the capsules in the layer above, $l+1$, sum to one. Successively, initial log prior probabilities are add to the coupling coefficients to obtain the final routing weights. The procedure remains unchanged in presence of multiple capsule layers, stacked on top of each other in order to create a deeper hierarchy.

Margin Loss and reconstruction regularizer

The output layer is not anymore represented by a scalar, but by a vector as well. Indeed, a capsule of the final layer does not only represent the probability that a certain object class exists, but also all its properties extracted from its individual parts. The length of the instantiation vector is used to represent the probability that a capsule's entity exists. Its length should be close to one if and only if the entity it represents is the only one present in the image. So, to allow multiple-class, we compute Eq. 9.7 for each class represented by a capsule n^L of the last layer L:

$$\mathcal{L}_{n^L} = T_{n^L} \max\left(0, m^+ - ||\boldsymbol{u}_n^L||\right)^2 + \lambda\left(1 - T_{n^L}\right) \max\left(0, ||\boldsymbol{u}_n^L|| - m^-\right)^2 \tag{9.7}$$

where T_{n^L} is equal to one if the class n^L is present and m^+, m^- and λ are hyperparameters to be tuned. Then, the separate margin loss $\mathcal{L}_{n^L}$ are summed to compute the final score during the training phase.

Finally, we adopt the reconstruction regularizer as in[212] to encourage all final capsules to encode robust and meaningful properties. So, the output capsules $\{\boldsymbol{u}_n^L\}_{n=1,\dots,N}$ are fed to the reconstruction decoder and the mean of L2 loss between an input image and the decoder output is added to the marginal loss scaled by a factor r.

9.1.2 Experiments and results

We aim to simply demonstrate that a properly working capsule network should achieve higher results with a considerably lower number of parameters due to its intrinsic capability to embed information better and efficiently. In this section, we test the proposed methodology in an experimental context, assessing its generalization capabilities and efficiency respect to traditional CNNs and similar works present in literature. On this purpose, we test our proposed methodology with three of the most used dataset for capsule-based networks assessment: Modified National Institute of Standards and Technology (MNIST), smallNORB and MultiMNIST. On all datasets, we demonstrate a remarkable difference with traditional solutions and comparable accuracy levels with similar methodologies but with a fraction of the trainable parameters in most cases. All experimentation clearly shows that a capsule network is capable to achieve higher results with a considerably lower number of parameters count. Moreover, we show how a simple ensemble of a few instances of Efficient-CapsNet can easily establish state-of-the-art results in all the three datasets. Finally, using principal component analysis (PCA), we give an introspect to the inner representations of the network and its capability to encode visual information.

Experimental settings

In all experiments, in order to map input samples onto an higher dimensional space, we adopt four convolutional layers with $k = 5$ for the first convolution and $k = 3$ for all others. On the other hand, f is equal to 32, 64, 64 and 128, respectively. ReLU is used in all layers, but leaky-ReLU is a valuable alternative. As previously discussed, the number of capsules depend by the number of feature maps, f, of the last convolutional layer. Indeed, the depthwise separable operation has a kernel dimension $k \times k$ equal to the output dimension $H \times W$ of the H_{Conv} function and a number of filters f equal to its filter dimension F. The first layer of primary capsules, $\boldsymbol{S}^1_{n,d}$, has $n^1 = 16$ capsules with a dimension d^1 of 8. Multiple fully-connected capsule layers can be added to increase the capacity of the network. However, we adopt only two layers of capsules due to the relative simplicity of the dataset investigated. Finally, the output layer of the network has a number of capsules n^L equal to the classes of the specific dataset taken into account. Since that higher-level capsules represent more complex entities with more degrees of freedom, their capsules dimensionality increases.

All loss parameters are obtained by CapsNet[212] training. So, for all experimentation m^+, m^- and λ are set to 0.9, 0.1 and 0.5, respectively. Moreover, the scaling factor r for the reconstruction regularizer is set to 0.392. Indeed, since we use the mean of L2 loss, while CapsNet uses the sum of L2 loss, $0.392 = 0.0005 * 784$. All experimentations are carried out on a workstation with an Nvidia RTX2080 GPGPU with 8GB of memory and 64GB of DDR4 SDRAM. We use the TensorFlow 2.x framework with CUDA 11. All result statistics are obtained with a mean of 30 trials.

In Table 9.1 is presented a comparison between the architecture of Efficient-CapsNet and other similar methodologies. Our model has a much lower number of parameters count, and it is much more efficient in terms of operations required. So, it can clearly highlight the generalization capability of capsules with respect to traditional CNN.

Method	**Parameters** [K]	**OPS**$\vert_{1batch}$ [G]	**Improvement**$\vert_{1batch}$ (%)
CapsNet[212]	6800	0.401	84.96
AR CapsNet[46]	5310	0.098	38.66
Matrix-CapsNet with EM routing[101]	310	0.086	29.56
Efficient-CapsNet	161	0.06	-

Table 9.1: Comparison of the computational cost in terms of necessary operations between Efficient-CapsNet and other similar methodologies present in literature. Efficient-CapsNet, besides having a reduced number of trainable parameters, is much more efficient.

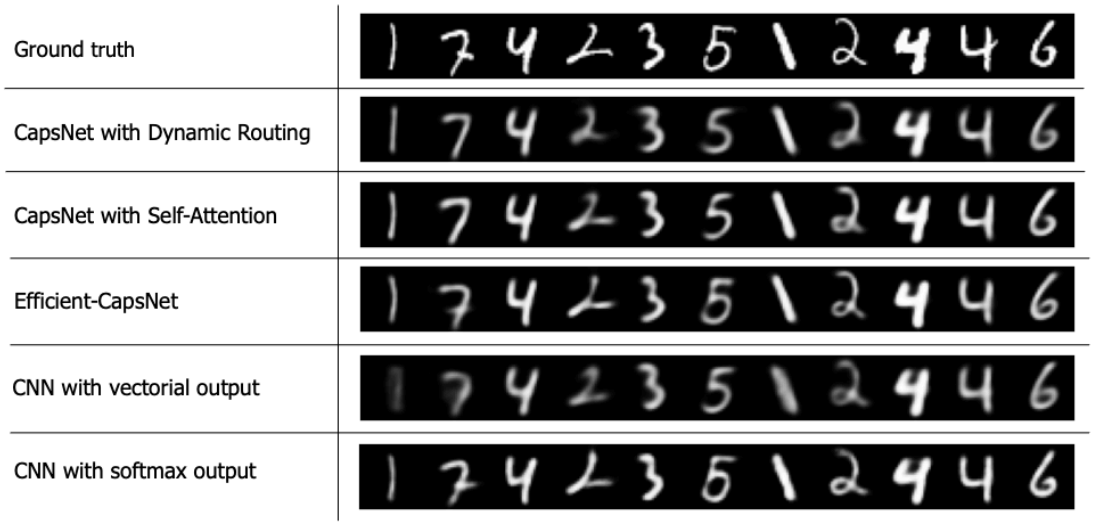


Figure 9.5: Digit reconstruction with different tested methodologies. Even with different architecture strategies and training objectives, all networks are able to embed different properties of the input digits keeping only important details.

Efficient-CapsNet on MNIST

The MNIST dataset [140] is composed of 70000, 28×28, images divided in 60000 and 10000 for training and testing, respectively. We adopt the same data augmentation proposed in Byerly et al.[31]. The reconstruction network is a simple fully-connected network with two hidden layers with 512 and 1024 neurons. We test our methodology and compare it with different models and two custom CNN baseline. In particular, our baseline is identical to Sabour et al.[212] with the

Method	**Reconstruction**	**Parameters** [K]	**MNIST** [%]
Our Baseline	no	173	0.48
Base-CapsNet	no	183	0.54
Our Baseline	yes	173	0.4
Base-CapsNet	yes	183	0.39
Efficient-CapsNet	yes	161	$0.26_{\pm 0.0002}$
Baseline[212]	no	35400	0.39
CapsNet[212]	yes	6800	$0.25_{\pm 0.005}$ $(0.36_{\pm 0.04})$*
Matrix-CapsNet with EM routing[101]	no	310	0.44
DA-CapsNet [110]	yes	7000	0.47
AR CapsNet [46]	yes	5310	0.54
HFCs [31]	no	1514	$0.25_{\pm 0.0002}$

Table 9.2: Test error (%) on the MNIST classification task. All methodologies are reported with their number of parameters and the presence of the reconstruction regularizer during the training phase. * indicates the results from our experiments.

Method	**Year**	**Test Error** [%]
Multi-Column Deep Neural Networks for Image Classification [49]	2012	0.23
Regularization of Neural Networks using DropConnect [250]	2013	0.21
RMDL:Random Multimodel Deep Learning for Classification [135]	2018	0.18
Base-Branching & Merging CNNw/HFCs [31]	2020	0.16
Efficient-CapsNet	2021	0.16

Table 9.3: Test error (%) on the MNIST classification task of state-of-the-art methodologies based on ensemble over the years.

exception of a reduced number of feature maps and layers, in order to keep the number of parameters as close as possible to Efficient-CapsNet. On the other hand, "Base-CapsNet" is a CNN but with a vectorial output as in a capsule-based network. So, it is also trained with the marginal loss function. That is specifically devised to assess the role of the reconstruction network and its impact on the overall accuracy. Our networks are trained for 100 epochs, batch size of 16, Adam [133] optimizer and an initial learning rate of $\eta = 5e - 4$ with exponential decay 0.98. All hyperparameters are selected with a small percentage of validation data.

In Table 9.2 are reported parameters and test errors of the different tested architectures. It is evident the gap between all baseline CNNs and all other capsule-based networks. Moreover, even if Efficient-CapsNet has barely 161K parameters, it is comparable with all other methodologies present in the literature so far. It achieves a mean accuracy of 0.9974 with a min value of 0.9971 and a max one of 0.9978. Finally, a network with a vectorial output receives a significant boost in performance using the reconstruction regularizer. In Figure 9.5 are presented some images generated by the reconstruction networks of the different tested methodologies. It also worth to notice that, even in the presence of an adaptive gradient descent method, Efficient-CapsNet does not overfit the training set but register a similar accuracy with the test set after the training.

As previously stated, we also demonstrate that a simple ensemble of Efficient-CapsNet models can easily establish a state-of-the-art result. Indeed, we exploit the 30 trained networks for test score statistics to produce an ensemble prediction. In particular, we average all network predictions with an accuracy greater than 0.9973, obtaining a final test error of 0.16. In Table 9.3 are summarized results of top MNIST leaderboard methodologies. The considerable gap between the mean single network test score, 0.26, and the ensemble one, 0.16, is due to the uncertainty on predictions of all remaining digits. Indeed, Efficient-CapsNet predicts the output class using the length of its output vector. So, unlike the exclusive softmax function, most of the ambiguous digits are reflected in the uncertainty of the network outputs. The ensemble simply steers predictions on the most probable answer. That is a clear sign of the strong knowledge of the dataset encapsulated by the network during the training. Indeed, analyzing the misclassified digits and their prediction

scores in the case of a single model clarifies the correctness of its answers despite the given labels. As shown in Figure 9.6, misclassified examples are ambiguous and classifying them correctly is only a matter of pure luck. In our opinion, it is for this reason that networks capable of achieving Efficient-CapsNet level of accuracy have modelled every important aspect of the MNIST dataset and further improvements in the test score have no significant meaning.

Efficient-CapsNet on smallNORB

The dataset smallNORB is a collection of 48600 stereo, grayscale images ($96 \times 96 \times 2$), representing 50 toys belonging to 5 generic categories: human, airplanes, trucks, cars and four-legged animals. Each toy was photographed by two cameras under 6 lighting conditions, 9 elevations, and 18 azimuths. The dataset is split in half; 5 instances of each category for the training and the remaining ones for the testing.

Efficient-CapsNet has the same structure described in the "MNIST results" section with the only exception of Instance Normalization [240] in place of Batch Normalization layers. That greatly helps the network to deal with different lighting conditions and make the network training as independent as possible of the contrast and brightness differences among the input images. On the other hand, we follow the same data augmentation and pre-processing proposed in Hinton et. al[101] with the only exception of the input dimension: we scale the original images to 64×64 using patches of 48×48. We train for 200 epochs, with a batch size of 16, Adam optimizer and an initial learning rate of $\eta = 5e - 4$ with exponential decay of 0.99.

In Table 9.4 are summarized the results of the baseline networks, Efficient-CapsNet and some capsule-based methodologies present in literature. As for the

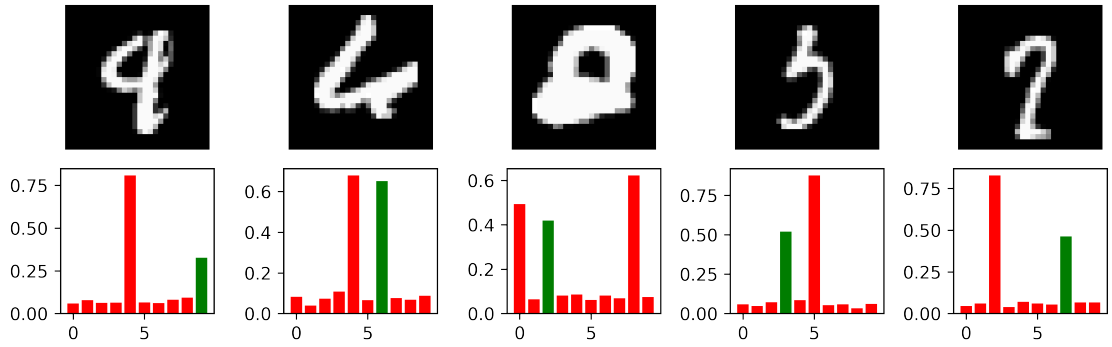


Figure 9.6: Example of Efficient-CapsNet misclassified digits. Green bars represent correct labels and their high the corresponding capsule length. The ambiguity of these remaining questionable examples is reflected in the uncertainty of the network predictions.

MNIST dataset, also for smallNORB is evident the gap between classical CNN and capsule-based networks. Moreover, again our methodology has comparable results with all other similar methodologies but with half of the parameters. It achieves a mean accuracy of 0.974 with a min value of 0.97 and a max one of 0.983. Finally, as before we exploit the 30 networks, trained for statistical evidence, to produce an ensemble prediction. We select only the two networks with the lowest test error, and we adopt for both a 40 patch prediction[101] before averaging their results. We obtain a test accuracy of 1.23, setting a new state-of-the-art result for this dataset.

Efficient-CapsNet on MultiMNIST

The MultiMNIST dataset has been proposed by Sabour et al. [212] and is based on the superposition of couples of shifted digits from the MNIST dataset. Each original image is first padded to a 36×36 pixels dimension. A MultiMNIST sample is generated by overlaying two padded digits, which shifts up to 4 pixels in both dimensions, resulting in an average 80% overlap. The only condition to be met is that the two digits are of different classes. In the labels, both indexes corresponding to the two classes are set to 1. In this way, the network aim is to detect both the digits concurrently. During training, the output capsules corresponding to the target classes are selected one at a time and used to reconstruct the two input images, while during testing we select the two most active capsules, i.e. the longest. Ideally, the network should be able to segment the two digits that have generated the MultiMNIST sample and independently reconstruct them. During training, for each epoch, we randomly generate 10 MultiMNIST images for each original MNIST example. We train the model 5 times independently for about 100 epochs, with a batch size of 64, Adam optimizer and an initial learning rate of $\eta = 5e-4$ with exponential decay of 0.97. Since we generate two reconstruction images for each input sample, we divide the reconstruction regularizer by half. During testing, we

Method	**Reconstruction**	**Parameters** [K]	**smallNORB** [%]
Our Baseline	no	198	5.9
Base-CapsNet	no	167	4.58
Our Baseline	yes	198	4.59
Base-CapsNet	yes	167	4.33
Efficient-CapsNet	yes	151	$2.54_{\pm 0.003}$
Baseline [101]	no	4200	5.2
Matrix-CapsNet with EM routing [101]	no	310	1.8 ($4.4_{\pm 0.004}$)*
CapsNet [212]	yes	6800	3.77
VB-Routig [203]	yes	310	$1.6_{\pm 0.06}$

Table 9.4: Test error (%) on the smallNORB classification task. All methodologies are reported with their number of parameters andthe presence of the reconstruction regularizer during the training phase. * indicates the results from our experiments.

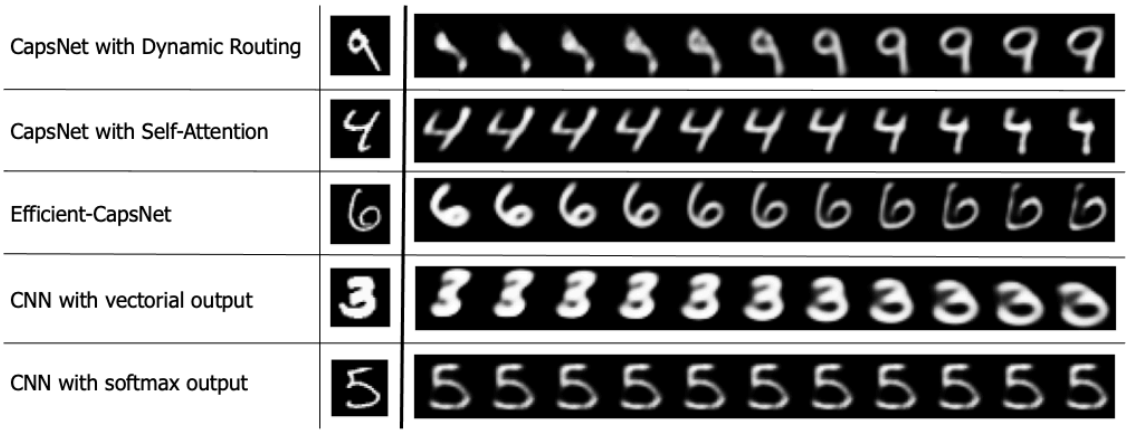
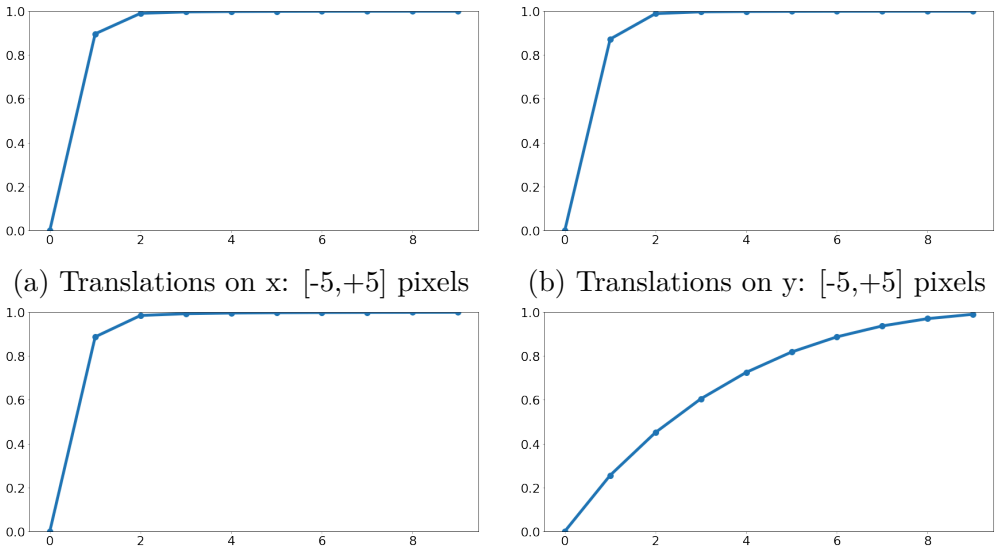


Figure 9.7: Effect on the digit reconstruction of the addition of perturbations to the output capsule values with different tested methodologies. All networks are able to embed shape, position and orientation information of the input digit except for the classical CNN with softmax output. That suggests that the capsule structure of the output, in which each class has its feature vector, is fundamental to get interpretable output embeddings.

generate 1000 MultiMNIST images for each MNIST digit to have a fair comparison with the work by Sabour et al. [212], for a total of 10 million samples. We get a mean test error of $5.1\%_{\pm 0.005}$ with our model of 154K parameters, in comparison to the original work test error of 5.2% with more than 9M parameters. Moreover, with an ensemble of the three models that get an accuracy greater than a threshold of 0.9470, we get a reduction of the test error to 3.8%. These results show how our methodology is able to correctly detect and recognize highly overlapping digits encoding information about their position and style in the output layer capsules.

Affine transformations embedding

To understand what kind of information is embedded in the output capsules, we can perturb the prediction and observe how the reconstruction is affected. We select the capsule with the longest length and we add small positive and negative contributions to its single elements. Figure 9.7 shows some example of perturbed images with different methodologies. We can observe how Efficient-CapsNet is behaving similarly to the original CapsNet [212], with the ability to encode combinations of different transformations of the digit. Retraining CapsNet also obtains similar behaviour with the proposed self-attention routing. A Convolutional Neural Network with a fake capsule layer, i.e. a vector instead of a scalar for each output class, also demonstrates the ability to encode actual shape, position and orientation information. On the other hand, considering the last features of a classical CNN, we are not able to reproduce this behaviour. That suggests that a capsule organization of



(a) Translations on x: [-5,+5] pixels

(b) Translations on y: [-5,+5] pixels

(c) Rotations: [-25,+25] degrees

(d) Random

Figure 9.8: Test set average cumulative variance explained with different numbers of PCA components by Efficent-CapsNet output capsule. It is clearly visible how the model is able to linearly embed affine transformations in the output space.

the output, in which each digit has its instantiation parameters and the activation is measured by the length of the vector, is fundamental for a meaningful embedding of the information.

To further investigate the ability of the proposed model to capture meaningful information in the components of the output capsules, we study the equivariance to transformations with a method similar to the one proposed by Choi et al. [46]. For each test image we generate the images corresponding to the 11 translations between [-5,+5] pixels on both the axes and to the 51 rotations between [-25,+25] degrees. If the model is behaving as expected, we should see that each affine transformation (translation on x, translation on y, rotation) is independently linearly encoded in the activations of the correct output capsule. We verify it, by computing the Principal Component Analysis on the output vectors for each type of transformation. We denote as K the number of transformed images and with N the number of output classes and we collect the output predictions $\boldsymbol{u}_i$, $i = 1, ..., K$. We center the data points and we compute the Singular Value Decomposition on the covariance matrix C:

$$\boldsymbol{z}_i = \boldsymbol{u}_i - \overline{\boldsymbol{u}} \tag{9.8}$$

$$\boldsymbol{C} = \frac{1}{K} \sum_{i=1}^{K} \boldsymbol{z}_i \, \boldsymbol{z}_i^T \tag{9.9}$$

$$\boldsymbol{C} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{U}^T \tag{9.10}$$

Method	**Translations on** x	**Translations on** y	**Rotations**
Random	$25.57\%_{\pm 0.028}$	$25.54\%_{\pm 0.028}$	$13.49\%_{\pm 0.009}$
CapsNet [212]	$83.78\%_{\pm 0.006}$	$79.82\%_{\pm 0.009}$	$88.01\%_{\pm 0.006}$
Efficient-CapsNet	$89.69\%_{\pm 0.005}$	$87.28\%_{\pm 0.008}$	$88.75\%_{\pm 0.005}$

Table 9.5: Average percentage of variance captured by the first component of PCA performed on the output capsule vectors of the different transformations applied to test set images.

As a linearity metric, we consider the fraction of the first eigenvalue σ_1 of the matrix $\boldsymbol{\Sigma}$ over the sum of all its eigenvalues. Since the eigenvalues represent the variance of the original data points explained by each component of the PCA, if the transformations are linearly encoded, we should have a high fraction of the variance captured with just a single component, thus a high first eigenvalue ratio.

$$r = \frac{\sigma_1}{\sum_{j=1}^{N} \sigma_j} \tag{9.11}$$

We perform this analysis on both the original CapsNet[212] and our model. The average results on all the test images are shown in Table 9.5, along with a comparison with the PCA performed on randomly generated vectors with the same dimension. Efficient-CapsNet shows higher linearity with respect to the original CapsNet in the encoding of affine transformations in the output capsule space. Figure 9.8 presents the average cumulative variance explained increasing the number of PCA components on the whole test set. For all the three transformations, Effienct-CapsNet is able to capture all the information with just two components, showing an almost perfectly linear behaviour with respect to the random example. That shows how our architecture can correctly embed position and orientation information of the recognized digit in the output vector components.

9.2 Architectural priors for out-of-distribution generalization

Classical DG on literature aims at training models able to generalize to OOD data. The access to a set of source datasets provides a predictor with the ability to extract and learn general invariant patterns, which are, hypothetically, also recognizable in the target domain dataset [180]. As an extension of supervised learning, this approach aims at minimizing empirical risk at training time to extrapolate from source datasets an overall probability distribution that enables accurate classification of OOD data. In the last decade, aware of the tremendous impact of generalization on computer vision applications, the DG research community has tackled the problem, primarily focusing on training algorithms. They have proposed methodologies that primarily aim to find invariances that hold novel domains. Among the constellation of proposed approaches, we identify the principal broad strategies adopted for domain generalization in augmenting the source domain [219], aligning domain distributions [149], meta-learning [268], self-supervised learning [7], and regularization strategies [217, 132].

Although methodologies have given meaningful insights about the nature of DG over the years, only recent research contributions have proposed a rigorous testing benchmark to fairly evaluate and compare the advantages provided by DG training algorithms. With DomainBed [90], the results obtained by the most relevant solutions have been critically analyzed over DG datasets, unmasking the marginal positive or negative improvement obtained in most cases with respect to naive ERM. Nevertheless, the study has been carried out uniquely with ResNet50 [92] as a feature extractor. Indeed, new DG algorithms are still proposed overlooking a fundamental aspect of practical deep learning applications: the importance of the architectural priors. Every year, competitive deep learning architectures emerge and outperform past models on popular datasets such as ImageNet [60]. However, the DG community has neglected the generalization power of existing backbones, which promotes sophisticated algorithms combined with outdated feature extractors such as ResNet18 [92] or AlexNet [137].

Before proceeding, it is essential at this point to define what a backbone is. Indeed, as previously stated, we focus our research on the importance of the architecture, but without neglecting the DG literature and the strong correlation between data and training procedure. Therefore, the real object assessed by this study is the backbone, $\mathcal{B} = f(\mathcal{A}, \mathcal{T}_B, \mathcal{D})$ that is a function of three elements: the model architecture $\mathcal{A}$, the training procedure $\mathcal{T}_B$ (including optimization, regularization, and data augmentation), and the training data $\mathcal{D}$. Consequently, in a classical DG setting all three factors introduce a certain degree of variability to the domain generalization accuracy:

$$\mathrm{DG}_{\mathrm{accuracy}}\ (\mathcal{S},\ T) = g(\mathcal{B}, \mathcal{T}_{DG}, \mathcal{N}_{exp})$$

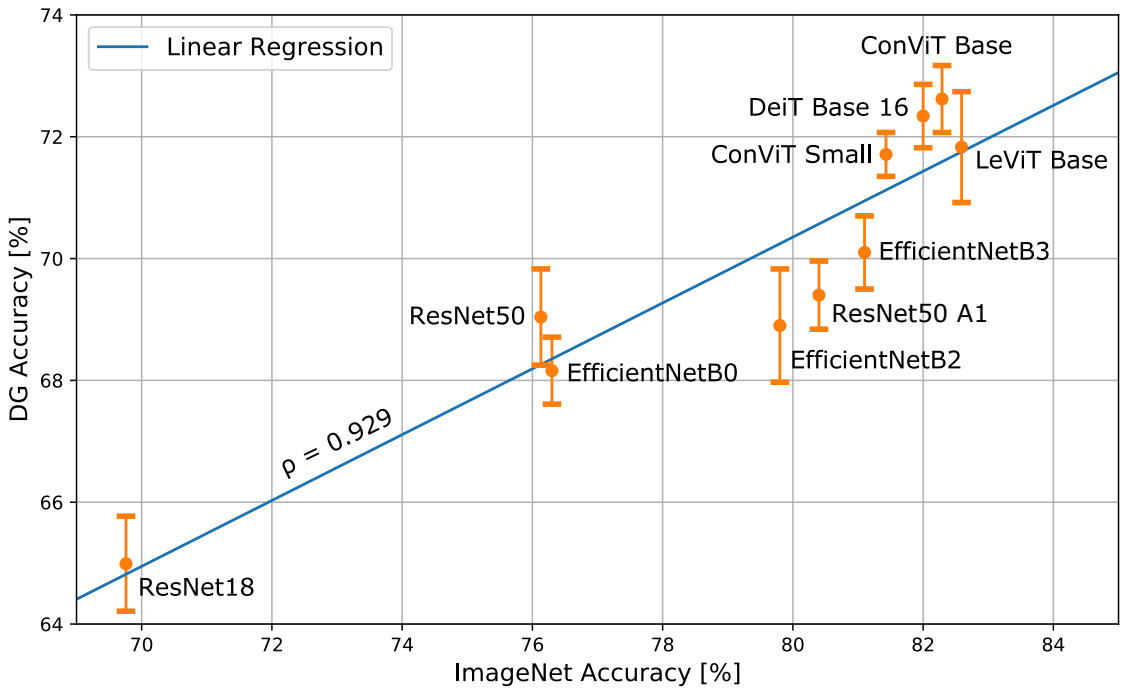


Figure 9.9: DG accuracy achieved by tested backbones compared with their performance on ImageNet, with error bars. We find a strong linear correlation between the two metrics ($\rho = 0.929$), regardless of different architectures and priors.

where $\mathcal{T}_{DG}$ is the adopted DG training procedure and $\mathcal{N}_{exp}$ is the experimentation noise. $\mathcal{T}_{DG}$ usually includes a dedicated algorithm to cope with domain shifts. $\mathcal{N}_{exp}$ comprehends a systematic error due to the adopted model selection strategy and a random component caused by the stochasticity in the training process.

In this study, we conduct extensive experimentation on the principal DG datasets and assess a wide variety of backbones architectures, from novel vision transformers to standard convolutional models. As summarized in Figure 9.9, our results demonstrate an evident linear correlation between large-scale single-domain classification accuracy (only presence of $\mathrm{T}_{i,j}$ with limited impact on the domain shift) and domain generalization performance (OOD). Moreover, we achieve state-of-the-art results in DG with naive ERM and simple data augmentation, remarking that, under fair testing conditions, the most promising DG algorithms presented so far give no substantial advantage in comparison with architectural priors.

We reinforce the experimentation with a visual analysis of the features extractors. Using the t-distributed Stochastic Neighbor Embedding (t-SNE) manifold learning technique [243] on extracted features, we show that novel backbones map closer same-class samples in the feature space and outperform older architectures when trained in a DG framework. We propose a quantitative evaluation of this difference by fitting a k-Nearest Neighbors (k-NN) classifier on the extracted features.

As an outcome of this work, we release Back-to-Bones[3], a testbed to encourage the deep learning community to evaluate and compare the domain generalization performance of newly proposed backbones.

9.2.1 Experimental settings

We set up our experimental benchmark to run a detailed analysis on the role of backbones in domain generalization. Besides choosing architectures, datasets, and DG algorithms to evaluate, particular attention is given to model selection strategy and statistical interpretation to obtain a fair and accurate benchmark. In the following subsections, we provide details on our experimental setup.

Backbones

To be consistent with previous works, we include ResNet18 and ResNet50 [92] in the benchmark and compare them with some of the most successful architectures proposed in recent image classification research. We also consider the latest ResNet50 A1 [254], trained using the most recent practices in optimization and data augmentation and reaching a remarkable 80.4% top-1 accuracy on Imagenet1K. We include different sizes for each network to glimpse the effects of model dimension on DG accuracy. EfficientNet [233] demonstrated that systematical model scaling and dimension balancing yield remarkable results with a reduced number of parameters. For this reason, we select three versions of the network, namely B0, B2, and B3. Finally, transformers [246] recently revolutionized deep learning by proving the effectiveness of self-attention for feature extraction; hence four transformer-based architectures are included in the comparison. In particular, we choose DeiT Base [238], ConViT [57] (both in its Small and Base configurations), and LeViT Base [87]. To provide further insights on the effect of additional pretraining data besides standard ImageNet [60], we also include Vision Transformer (ViT) [68] trained on ImageNet21K in its Small and Base versions. Regarding ViT Base, a configuration with a 32x32 patch size has been added to the standard 16x16 format to test the impact of patch size on DG.

Datasets

Among the various datasets created explicitly for DG in the last years, we use four of the most widely adopted ones for our primary experimentation. VLCS [75] considers four previous classification datasets as domains, while PACS [146] and Office-Home [247] focus more on style shifts (e.g., from photos to cartoons, sketches, and paintings). TerraIncognita [18] comprehends several animal photos

[3] https://github.com/PIC4SeR/BacktoBones

taken with camera traps placed in different locations by day and by night. To those, we add DomainNet [192], a bigger and more recent dataset that contains six domains divided by style and 345 classes. We use it to further stress the generalization capability of the best-performing backbones in the presence of more transfer learning data and fewer samples per class. We omit Rotated MNIST [82] and Colored MNIST [11] since we consider them too distant from any practical application. Moreover, from our perspective, simple rotation and colorization fall under transformations $T_{i,j}$ with a limited impact on the source joint probability distribution.

DG Algorithms

We choose three of the most promising DG algorithms in recent research, particularly considering their performance on DomainBed [90]. Moreover, we select them to explore different approaches to the DG problem. CORAL [229], indeed, focuses on aligning the extracted features through second-order statistics (covariance). On the other hand, Mixup [261] works directly on input images interpolating samples from different domains and considering the loss coming from both precursors. RSC [111], instead, introduces a heuristic that discards dominant features in the label determination, stimulating the model to rely on weaker data correlations.

Data Augmentation

Many research works prove that data augmentation plays a fundamental role in DG, as it can partially compensate for certain domain shifts. That is particularly true in the presence of style changes, as popular data augmentation strategies involve the alteration of saturation, hue, and contrast. The de-facto standard augmentation strategy for DG, which we use in our benchmark, includes random cropping keeping at least 80% of the original image, horizontal flipping with 50% probability, image grayscaling with 10% chance, and random changes in color brightness, contrast, saturation, and hue, with a maximum of 40%. Since all the models are pretrained on ImageNet1K or ImageNet21K, input images are further normalized according to the mean and standard deviation of that datasets.

Model Selection

To assess the DG capability of the considered pretrained networks, we fine-tune each of them on a set of K source domains $\mathcal{S}$ and test them on a target domain T. As pointed out by [90], "a domain generalization algorithm should be responsible for specifying a model selection method" and avoid any improper comparison between results obtained adopting different selection methods. In total agreement with their recommendations, we use the *training-domain validation set* strategy, which picks the model maximizing the accuracy on a validation split of the training set (in our

case 10%, uniform across domains) at the end of each epoch. This selection method assumes that the average distribution of source domains is similar to that of the target domain, on which the best model is tested.

Hyperparameter Search

We conduct a random search for each backbone and dataset to determine the optimal training hyperparameters for the baselines. We define a range of values for continuous arguments and a set of choices for discrete ones, running approximately 32 iterations for each search and selecting the best combination via the previously defined model selection strategy. The learning rate is bounded in the range $[10^{-6},10^{-2}]$, choosing its scheduler among step (90% reduction after 80% of the epochs), exponential (with a decay in the range [0.9,1)), and cosine annealing. The batch size and the number of training epochs are the same for all the experimentation, fixing their values at 32 and 30, respectively. Finally, we use cross-entropy loss and select the optimizer among SGD (with a momentum of 0.9) and Adam, keeping the weight decay to $5 \cdot 10^{-4}$.

Experimental Framework

Our benchmarks are developed in Python 3 using the deep learning framework PyTorch. As the experimentation applies transfer learning to pretrained models, we use existing implementations of the considered backbones. Only the classification head is changed, adapting the network to the different number of classes. In particular, standard ResNets are taken from the PyTorch library *torchvision*[4], EfficientNets from *EfficientNet-PyTorch*[5], transformers and ResNet50 A1 from *timm*[6]. The implementations of DG algorithms are taken from DomainBed[7] and adapted to work with the architectures under test.

We repeat each training three times with different and randomly generated seeds to give more statistical information about accuracy results. In this way, both hyperparameter search and benchmarks cannot take advantage of the repeatability of trials, as data splitting, augmentation, and weight initialization change from an iteration to the next. Therefore, each of the results of our benchmark is reported as the mean over three repetitions along with its standard deviation.

Backbone	PACS	VLCS	Office-Home	TerraIncognita	Average	ImageNet
ResNet18	80.51 ± 0.29	74.64 ± 0.61	63.87 ± 0.36	40.93 ± 1.85	64.99 ± 0.78	69.76
ResNet50 [90]	85.50 ± 0.20	77.50 ± 0.40	66.50 ± 0.30	46.10 ± 1.80	68.90 ± 0.68	76.13
ResNet50	83.85 ± 0.77	76.21 ± 1.20	68.79 ± 0.21	47.32 ± 0.97	69.04 ± 0.79	76.13
ResNet50 A1	84.52 ± 0.68	78.37 ± 0.56	72.47 ± 0.13	42.23 ± 0.87	69.40 ± 0.56	80.40
EfficientNetB0	85.46 ± 0.65	75.16 ± 0.34	67.27 ± 0.27	44.76 ± 0.94	68.16 ± 0.55	76.30
EfficientNetB2	87.02 ± 1.37	75.44 ± 0.20	69.35 ± 0.24	43.80 ± 1.90	68.90 ± 0.93	79.80
EfficientNetB3	86.71 ± 0.30	78.14 ± 0.18	69.84 ± 0.08	45.70 ± 1.84	70.10 ± 0.60	81.10
DeiT Base 16	88.10 ± 0.48	79.80 ± 0.32	76.03 ± 0.28	45.45 ± 1.00	72.34 ± 0.52	82.00
ConViT Small	87.10 ± 0.33	80.00 ± 0.34	73.90 ± 0.17	45.83 ± 0.61	71.71 ± 0.36	81.43
ConViT Base	87.27 ± 0.40	**80.31 ± 0.67**	76.51 ± 0.25	46.37 ± 0.89	72.62 ± 0.55	82.29
LeViT Base	87.55 ± 1.50	78.91 ± 0.50	75.16 ± 0.13	45.68 ± 1.50	71.83 ± 0.91	82.59
ViT Small 16*	83.59 ± 0.43	79.96 ± 0.60	77.25 ± 0.33	44.12 ± 1.07	71.23 ± 0.61	81.40
ViT Base 32*	84.00 ± 1.17	78.46 ± 0.64	76.84 ± 0.17	36.71 ± 2.07	69.00 ± 1.01	80.72
ViT Base 16*	**88.48 ± 1.22**	80.05 ± 0.15	**81.47 ± 0.21**	**49.77 ± 1.28**	**74.94 ± 0.72**	84.53

Table 9.6: Baselines comparison of different backbones for DG. For each model, we report the average accuracy over three runs and the associated standard deviation. We include the results achieved by DomainBed with ResNet50 for reference. The models marked with * are pretrained on Imagenet21K instead of ImageNet1K. The rightmost column indicates the accuracy of the networks on ImageNet1K.

9.2.2 Baselines Benchmark

The first analysis of our work consists of a precise and fair benchmark of the DG capabilities of recent deep learning architectures for image classification, trying to determine what solutions work best and, possibly, why. Every pretrained backbone, after a hyperparameter search, is trained following the standard DG *leave-one-domain-out* procedure using the previously described model selection strategy. Our benchmark results are reported in Table 9.6 as the mean and standard deviation over three iterations.

Firstly, our benchmark highlights a strong correlation between DG accuracy and ImageNet performance. As depicted in Figure 9.9, we find a direct proportionality between the two metrics (excluding the ViT models due to their different pretraining). Applying linear least-square regression, we obtain a Pearson correlation coefficient (ρ) of 0.929. Indeed, a quick look at the results is sufficient to notice how newer and more performing backbones tend to achieve a higher DG accuracy on nearly all the datasets. That is primarily true for different sizes of the same architecture. ResNet50 reaches better results than ResNet18 for all the datasets, and the same happens for EfficientNet, ConViT, and ViT variants. For ResNet50, we

[4] pytorch.org/vision/stable/models

[5] github.com/lukemelas/EfficientNet-PyTorch

[6] github.com/rwightman/pytorch-image-models

[7] github.com/facebookresearch/DomainBed

Backbone	C	I	P	Q	R	S	Avg
ResNet50[90]	58.1	18.8	46.7	12.2	59.6	49.8	40.9
DeiT Base 16	69.1	25.0	55.8	17.1	69.3	57.0	48.9
ConViT Base	69.5	24.3	55.7	**17.7**	69.3	57.0	48.9
ViT Base 16*	**74.9**	**28.9**	**60.8**	17.5	**77.3**	**61.8**	**53.5**

Table 9.7: Baseline comparison of a selection of the best backbones on DomainNet (*Clipart*, *Infograph*, *Painting*, *Quickdraw*, *Real*, and *Sketch* domains). We include the results achieved by DomainBed with ResNet50 for reference. The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

also compare our results with those obtained by DomainBed and find comparable values. ResNet50 A1 proves to benefit from its stronger pretraining, largely improving the accuracy obtained by the standard model on VLCS and Office-Home. However, TerraIncognita seems to penalize the network with its peculiar light conditions, resulting in a slight overall enhancement. As regards different architectures, EfficientNetB2 performs very similarly to ResNet50 while the B3 version gains an additional 1% on them. Transformer-based models bring further improvements exploiting their self-attention-based feature extraction, even in the case of ConViT Small. In particular, they strongly outperform EfficientNet on OfficeHome by over 4%, while the only dataset without any significant progress is TerraIncognita. That is probably due to the peculiarity of the domains, comprehending many night shots that can be challenging even for humans and rewarding less an effective ImageNet pretraining. Among other transformers, DeiT Base 16 and ConViT Base prove to be the best, the latter being slightly more performing. Finally, the three ViT models show that pretraining on a more significant amount of data results in better DG. However, only ViT Base 16 registers a considerable step forward, suggesting that the abundance of data is fully exploited only by larger models. Nonetheless, ConViT Small performs similarly to the same-sized ViT Small 16, while larger patches demonstrate to degrade the accuracy of ViT Base 32. In conclusion, our results show how better DG comes from the union of a good feature extractor architecture and an optimal pretraining, as none of the two is sufficient alone.

Finally, we conduct an additional benchmark on the larger DomainNet dataset. Although representing a significant challenge for large-scale generalization, we choose to include DomainNet only in this second stage of the study due to its demanding computational nature and its strong class unbalancing. Indeed, our main intention is to promote a practical and accessible benchmark, which aims to become a widespread reference for DG. We select only the best three models from the previous tests for this one (DeiT Base 16, ConViT Base, and ViT Base 16). In Table 9.7 we report the results achieved on each test domain, including those obtained by DomainBed on ResNet50 for reference. It is well evident that the feature extraction capabilities of modern backbones bring substantial improvement in all the

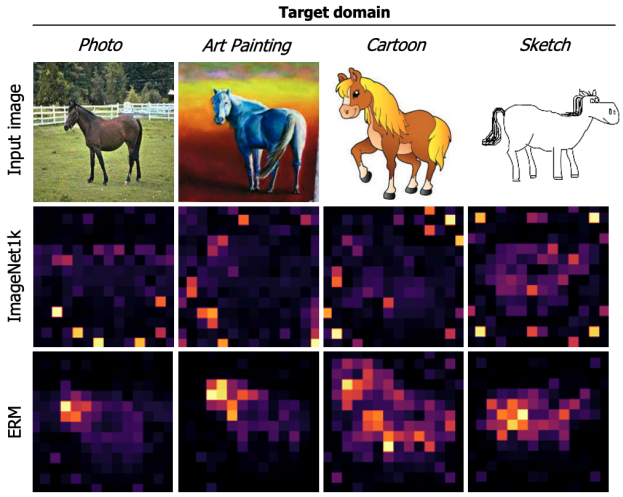


Figure 9.10: DeiT Base attention maps when using the `[CLS]` token as a query for the different heads in the last layer. We select the same head for all examples. ERM encourages the backbone to focus on domain-invariant features, highly mitigating pretraining noise.

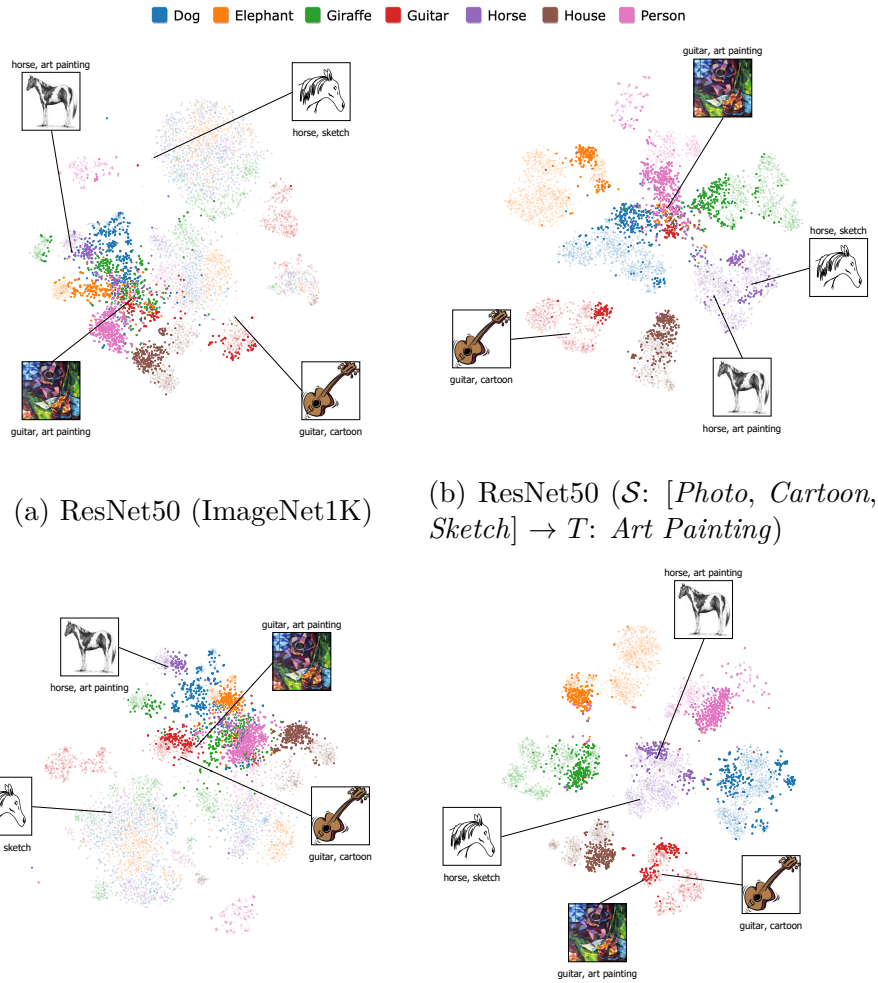
domains, with an average increase in DG accuracy up to 12.6%. Moreover, ViT further enhances the results by exploiting its stronger pretraining.

9.2.3 Models Introspection

After assessing the DG performance of different backbones, we propose a series of insights on the way different architectures leverage training data to create their own inner representation. First, we investigate the benefits of ImageNet pretraining for DG with a k-NN classifier, comparing ResNet50 and the best models from our benchmark. Then, we apply t-SNE [243] on the same extracted features to visualize how close same-class and same-domain samples are, and the effect of fine-tuning on DG datasets. Finally, we inspect the attention maps of one of the transformer-based models to have a qualitative insight on the region of the images it focuses on.

k-NN Evaluation

Firstly, we take ResNet50 and the best-performing models from our benchmark and evaluate their ability to tackle DG without fine-tuning. To do so, we use ImageNet weights to extract features from training domains and a k-NN (with $k = 5$) to fit that data. Then, we use test-domain images for the evaluation. To have a fair comparison with our benchmark, we use the same amount of training data leaving out 10% of samples from source domains. The results in Table 9.8 show an overall difference of about 5% between ResNet50 and transformer-based models pretrained



(a) ResNet50 (ImageNet1K)

(b) ResNet50 ($\mathcal{S}$: [*Photo*, *Cartoon*, *Sketch*] $\rightarrow$ T: *Art Painting*)

(c) ConViT Base (ImageNet1K)

(d) ConViT Base ($\mathcal{S}$: [*Photo*, *Cartoon*, *Sketch*] $\rightarrow$ T: *Art Painting*)

Figure 9.11: Backbone features visualization with t-SNE on PACS. Target domain *Art Painting* samples are highlighted. For better interpretability, some image examples from different domains and classes are visualized. After the fine-tuning, the ConViT Base architecture achieves a better class separation with respect to ResNet50, clustering together same-class samples of different domains.

on ImageNet1k. This outcome is consistent with the generalization boost achieved in the standard DG framework (Table 9.6), although k-NN results tend to oscillate among different datasets. On the same trend, ViT Base 16 gains an additional 10% average accuracy, thanks to its pretraining on the larger ImageNet21K dataset. This outcome suggests that learning a wider overall source distribution $P^{\mathcal{S}}_{XY}$ is always needed to effectively tackle a substantial domain gap, and that pretraining alone does not guarantee the ability to extract domain-invariant features.

Backbone	PACS	VLCS	Office-Home	TerraInc.	Avg
ResNet50	56.04	69.57	56.26	14.75	49.16
DeiT Base 16	56.27	65.50	65.57	27.06	53.60
ConViT Base	56.83	64.50	66.63	**27.96**	53.98
ViT Base 16*	**75.14**	**75.14**	**82.72**	25.64	**64.66**

Table 9.8: Comparison of different feature extractors without fine-tuning, using a k-NN classifier ($k = 5$). The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

Feature Mapping Visualization

To further enlighten the role of backbones in extracting meaningful and invariant features to deal with DG, we can visualize the distributions in the feature space by projecting them in a two-dimensional space using t-SNE. Table 9.11 shows t-SNE visualization for ResNet50 and ConViT Base, pretrained on ImageNet1K and fine-tuned on PACS, targeting the *Art Painting* domain. For each model, we remove the classification head and extract the features for the whole dataset. The more clustered the same class features appear in the t-SNE, the more separable from other classes they are in the original space.

Figure 9.11a shows how ResNet50 pretrained on ImageNet tends to map together same-domain samples and not same-class ones, being therefore unsuitable for DG without fine-tuning. After the fine-tuning process (Figure 9.11b), the model

Backbone	Algorithm	PACS	VLCS	Office-Home	TerraIncognita	Average
ResNet50 [90]	ERM	85.50 ± 0.20	77.50 ± 0.40	66.50 ± 0.30	46.10 ± 1.80	68.90 ± 0.68
	CORAL	**86.20 ± 0.30**	**78.80 ± 0.60**	**68.70 ± 0.30**	47.60 ± 1.00	**70.33 ± 0.55**
	Mixup	84.60 ± 0.60	77.40 ± 0.60	68.10 ± 0.30	**47.90 ± 0.80**	69.50 ± 0.58
	RSC	85.20 ± 0.90	77.10 ± 0.50	65.50 ± 0.90	46.60 ± 1.00	68.60 ± 0.83
DeiT Base 16	ERM	**88.10 ± 0.48**	**79.80 ± 0.32**	**76.03 ± 0.28**	45.45 ± 1.00	**72.34 ± 0.52**
	CORAL	85.13 ± 0.82	78.34 ± 0.86	75.72 ± 0.20	46.33 ± 1.83	71.38 ± 0.93
	Mixup	85.67 ± 0.61	78.25 ± 0.60	74.73 ± 0.22	**46.63 ± 0.49**	71.32 ± 0.48
	RSC	85.37 ± 1.30	77.27 ± 0.51	75.83 ± 0.29	45.41 ± 1.50	70.97 ± 0.90
ConViT Base	ERM	**87.27 ± 0.40**	**80.31 ± 0.67**	76.51 ± 0.25	**46.37 ± 0.89**	**72.62 ± 0.55**
	CORAL	86.24 ± 0.24	79.62 ± 0.38	75.33 ± 0.22	44.41 ± 1.33	71.40 ± 0.54
	Mixup	86.00 ± 0.45	80.00 ± 0.76	76.48 ± 0.16	43.95 ± 0.18	71.61 ± 0.39
	RSC	85.73 ± 0.81	79.05 ± 0.61	**76.77 ± 0.26**	44.94 ± 1.47	71.62 ± 0.79
ViT Base 16*	ERM	88.48 ± 1.22	80.05 ± 0.15	81.47 ± 0.21	49.77 ± 1.28	74.94 ± 0.72
	CORAL	84.60 ± 1.31	**80.89 ± 0.49**	81.17 ± 0.19	**50.58 ± 0.26**	74.31 ± 0.56
	Mixup	**88.62 ± 0.54**	80.77 ± 1.28	**82.93 ± 0.07**	48.59 ± 0.92	**75.23 ± 0.70**
	RSC	86.58 ± 2.14	79.59 ± 0.63	78.74 ± 0.64	40.79 ± 1.41	71.42 ± 1.20

Table 9.9: Comparison between ERM and three promising DG algorithms on the best-performing backbones of our benchmark. For each model, we report the average accuracy over three runs and the associated standard deviation. We include the results achieved by DomainBed with ResNet50 for reference. The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

achieves a better separation of source domains classes. However, a high number of target domain samples is still mapped in the same space, far from the same-class source clusters (e.g., the *Art Painting* guitar example). Similarly to ResNet50, without fine-tuning domains dominate the features space distribution of ConViT (Figure 9.11c), causing several clusters of the same class, but different domains, to emerge in different locations (e.g., horse samples). However, some same-class samples of more similar domains, such as the guitars of *Cartoon* and *Art Painting*, are effectively clustered together. The fine-tuning process (Figure 9.11d) distinctly pushes together same-class clusters, resulting in good generalization over the target domain, too. This analysis suggests that the ConViT backbone is more suited for DG than ResNet50, since it tends to give more similar representations to same-class samples from different domains. Additional visualizations are presented in the supplementary material.

Self-attention Visualization

In literature, DG algorithms are often presented with a qualitative analysis, highlighting the regions the network focuses on using interpretation methods such as GradCAM [216]. Indeed, heat maps are brought as evidence of their capability to push attention towards more localized and domain-invariant features. Nevertheless, this section shows that competitive backbones with naive ERM are perfectly capable of localizing class-discriminative regions. In particular, Figure 9.10 shows the attention maps extracted using the `[CLS]` token as a query for the different heads in the last layer of the DeiT Base architecture. We provide four random examples for different target domains of PACS showing the same attention head map before and after DG fine-tuning. It is remarkable how naive ERM is able to redirect attention towards more invariant features.

9.2.4 Algorithms Evaluation

Domain generalization research manly focuses on the study of non-trivial algorithms to get a reduction of the domain shifts. However, these algorithms are uniquely proposed in combination with outdated backbones such as ResNet50, ResNet18, or even AlexNet. According to the results in Table 9.6, recent backbones can provide significant improvements with respect to ResNet50 with simple ERM. At this point, it is worth determining whether the application of DG algorithms brings a further boost in generalization to our baselines. To do so, we combine some of the most promising algorithms available on DomainBed with three of our best baselines. We evaluate CORAL, Mixup, and RSC using ViT Base 16, DeiT Base 16, and ConViT Base as backbones, repeating each training three times. Table 9.9 reports the obtained results, composed of average accuracy and associated standard deviation. The overall performance of ERM is unbeaten for

both DeiT Base and ConViT Base, while, for ViT Base, Mixup slightly outperforms it but remains in the same uncertainty range. Results with ResNet50 are also reported directly from DomainBed for the same group of datasets. It is evident how DG algorithms improve generalization properties marginally for ResNet50 and even negatively for transformer-based backbones. This outcome extends the recent findings of DomainBed to other baselines and strongly reinforces the belief that choosing an effective backbone is the first step towards filling domain gaps. Adopting an outdated or poorly trained baseline is not the correct way to demonstrate the improvement derived from a DG algorithm.

Chapter 10

Conclusions

Learning, intelligence, and consciousness are three fuzzy concepts that highly distinguish us as humans. There are three concepts that artificial intelligence research and studies would like to bring to our machines despite the lack of a shared agreement on their definitions. Artificial machines, projections of our own consciousness, that maybe someday will dialogue and think as our peers. Will it ever be possible?

Except for consciousness or the articulated mechanisms behind its origin, this dissertation presented how studies in the field of artificial intelligence are starting to produce tools to endow machines with intelligence. Machines that will be capable of making the right choices in different situations and adapting to our dynamic and diversified world. Our own imagination only limits applications of such technology: intelligent buddies to help in our daily works, assistance for elderly and fragile persons, exploring inhospitable places, and bringing equity to our world are only some visions of this future technology. Among all fields of application of AI, the multidisciplinary branch of service robotics better represents the incarnation of this future vision.

The different chapters of the dissertation have presented how data-driven techniques can be incorporated with service robotics. A particular vision of service robotics in which similar machine learning and deep learning approaches can be exploited to the whole data ecosystem, ultimately guiding navigation and decision-making of autonomous machines. In particular, the first part of the dissertation described how it is possible to use data-driven approaches to extract useful information from remote sensing data sources to support edge vehicles missions. All selected contributions brought as an example in the first part are united by a precision agricultural application. However, similar data ecosystems can be found in every service robotics field of application. Subsequently, part two presented how similar deep learning methodologies can be brought directly onboard machines, enhancing their perception and controlling their instinctive actions. It completes the precision agriculture example, presenting how everything comes together and

how all extracted knowledge can be integrated. At the same time, part two raises the lack of efficiency of current deep learning techniques. Indeed, model accuracy is juxtaposed with power consumption and efficiency at this level, and algorithms need to be modified accordingly. Cloud is not an option for most practical applications, and deep learning algorithms require building a strict bond with dedicated hardware. Finally, the dissertation invites reflection on the generalization problem that assumes a relevant role for autonomous machines. Indeed, efficiency is not enough, and generalization always constitutes an essential piece of the problem for service robotics. Systematic generalization and domain generalization are discussed in the last part of the dissertation, reflecting on the problem of bridging simulation to reality gap, overcoming current architecture, training procedure, and data limitations.

10.1 Further works and future directions

Efficiency and generalization are core problems for AI. They primarily affect current deep learning approaches and prevent the creation of truly intelligent and autonomous machines. In the last decade, important achievements of deep learning attracted huge interest from research on all fields of science that have poured in an attempt to make their own contribution. However, this running could have come at the expense of pause and pondering on the principles and foundations of the field itself. Moreover, the latest success on attention applications could mislead research to investigate more complicated and audacious goals as reasoning.

Instead, further works should be more concerned with devising efficient models since the origin. Models that are not only efficient during inference, but also during training. Starting back from the neuron model and investing sparsity as a building block for the creation of future architectures could be a good point to investigate. On the other hand, if our brain requires less than 20 W to operate, there is undoubtedly a large margin to improve what we are doing.

In the same way, also the generalization problem in deep learning is usually limited to showing promising results on a particular test dataset. Research groups which go a little bit further with generalization are usually more concerned at showing fancy methodologies that score a little bit better on benchmarks with an end in themselves. Conversely, researchers should go deeper in investigating the fundamental interactions between architectural priors, training algorithms, and data for unlocking AI from the bottleneck it is inevitably going towards.

Appendix A

Supplementary Materials RerefyNet Evaluation

The performance of the proposed RarefyNet in extending the refinement task also to other imagery from a time-series, even if trained only with one single UAV-driven dataset, is thus further assessed by refining other temporal raw Sentinel-2 maps. The effectiveness of refined maps $\hat{X}^{III}$ and $\hat{X}^{IV}$ (obtained by refining maps X^{III}_{raw} and X^{IV}_{raw}) in describing the vigour level of the vineyard expressed in reference UAV-driven maps Y^{III}_{UAV} and Y^{IV}_{UAV} is investigated with ANOVA. The results of this analysis, together with the ones performed on $\hat{X}^{I}$ and $\hat{X}^{II}$ for completeness, are organized in Table 4. The boxplots of the groups of pixels from the refined satellite maps ($\hat{X}^{I}$, $\hat{X}^{II}$, $\hat{X}^{III}$, $\hat{X}^{IV}$), clustered according to the three vigour classes "L," "M" and "H" defined in the UAV-driven clustered maps Y^{I}_{UAV}, Y^{II}_{UAV}, Y^{III}_{UAV} and Y^{IV}_{UAV}, respectively, are shown in Figure 5. The ANOVA results reported in Table 4 confirmed the excellent coherence of all four refined Sentinel-2 maps with their respective reference maps, with p-values showing the significance of the differences among groups means. The results achieved by the performed analysis provide an opportunity to use the freely and frequently available, low-resolution satellite imagery to describe the variability of vineyards by refining the Satellite driven vegetation index.

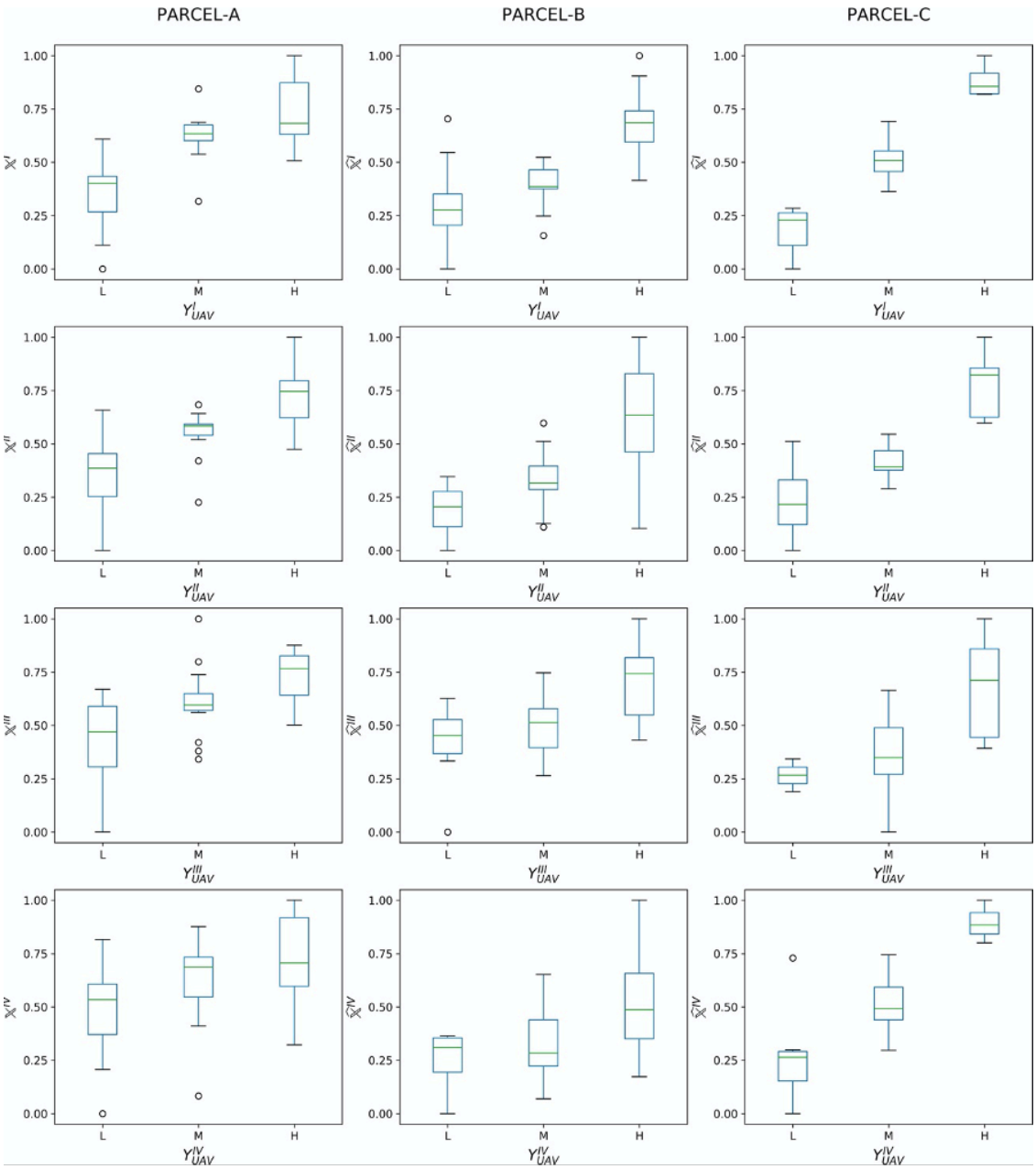


Figure A.1: Pixel groups boxplots from refined satellite maps ($\hat{X}^{I}$, $\hat{X}^{II}$, $\hat{X}^{III}$, $\hat{X}^{IV}$), clustered according to the three vigour classes "L," "M" and "H" defined in the UAV-driven clustered maps Y^{I}_{UAV}, Y^{II}_{UAV}, Y^{III}_{UAV} and Y^{IV}_{UAV}, respectively. The boxplots are computed individually for each parcel (A, B, and C).

Datasets	**Parcel**	**Source**	**DF**	**SS**	**MS**	**F-Value**	**P-Value**
$\hat{X}^{I}(Y^{I}_{UAV})$	Parcel-A	Classes	2	0.7907	0.7907	32.4702	2.60E-06
		Error	31	0.7792	0.0243		
		Total	33	1.5699			
	Parcel-B	Classes	2	1.39025	1.3902	78.7860	9.31E-13
		Error	63	1.1293	0.0176		
		Total	65	2.5196			
	Parcel-C	Classes	2	1.1914	1.1917	113.4301	1.14E-08
		Error	15	0.1681	0.0105		
		Total	17	1.3596			
$\hat{X}^{II}(Y^{II}_{UAV})$	Parcel-A	Classes	2	0.6968	0.6968	31.9907	2.94E-06
		Error	31	0.6970	0.0218		
		Total	33	1.3939			
	Parcel-B	Classes	2	1.5536	1.5536	58.4472	1.36E-10
		Error	63	1.7012	0.0266		
		Total	65	3.2548			
	Parcel-C	Classes	2	0.7978	0.7978	35.3635	2.05E-05
		Error	15	0.3609	0.0225		
		Total	17	1.1587			
$\hat{X}^{III}(Y^{III}_{UAV})$	Parcel-A	Classes	2	0.4195	0.4194	13.4022	0.000898
		Error	31	1.0015	0.0313		
		Total	33	1.4210			
	Parcel-B	Classes	2	0.6561	0.6560	29.8767	8.10E-07
		Error	63	1.4054	0.0219		
		Total	65	2.0614			
	Parcel-C	Classes	2	0.1808	0.1808	2.1895	0.158372
		Error	15	1.3218	0.0826		
		Total	17	1.5026			
$\hat{X}^{IV}(Y^{IV}_{UAV})$	Parcel-A	Classes	2	0.2441	0.2441	4.6372	0.038924
		Error	31	1.6846	0.0526		
		Total	33	1.9287			
	Parcel-B	Classes	2	0.6649	0.6649	20.8288	2.33E-05
		Error	63	2.0431	0.0319		
		Total	65	2.7081			
	Parcel-C	Classes	2	0.8174	0.8173	25.5642	0.000117
		Error	15	0.5116	0.0319		
		Total	17	1.3289			

Table A.1: ANOVA results of refined datasets $\hat{X}^{I}$, $\hat{X}^{II}$, $\hat{X}^{III}$ and $\hat{X}^{IV}$, grouped according to reference UAV-drive vigour maps Y^{I}_{UAV}, Y^{II}_{UAV}, Y^{III}_{UAV} and Y^{IV}_{UAV}.

Appendix B

BabyAgent: Beyond Internet Datasets

There is a subtle but remarkable difference between how modern deep learning algorithm learns with respect to "living beings": they always work with a fixed and pre-processed dataset. That has two main implications: firstly, models acquire knowledge constantly cycling over the same patterns. Secondly, datasets contain biases introduced by the authors. For instance, in a typical image classification problem, a dataset usually features well-defined subjects in the center of the field of view; blurring, chromatic aberrations, and other image defects are not present and only simulated through data augmentation. On the other hand, signals acquired by animals are much richer, and their learning algorithms have to deal with challenges that current deep learning models do not have to face. For instance, deal with multiple and not precise subjects in a scene. In other words, they learn from real-world scenes and not an imitation of them.

In June 2021, driven by these intuitions, we opened a preliminary research project, dubbed BabyAgent, to investigate the role of datasets and learning algorithms over visual perception generalization. It is crucial to premise that at the time of writing, the presented research is an active and ever-changing project, but with clear, precise goals: apply the same concepts to a stream of images and architectural priors equivariant to affine transformations. Indeed, we are convinced that object movements in the real world are key to learning strong and robust representations.

The first objective of BabyAgent was to device a system able to learn from an ever-changing stream of real-world data scenes and then introspect its learned representations. So, not working anymore with a defined dataset, but similar to living beings, with images captured on a daily basis. Nevertheless, it is essential to reiterate that all data are intended as preliminary results, and this appendix is more devoted to introducing the main ideas under the presented research.

Figure B.1: BabyAgent's first prototype placed at the center of the photo collage. Since its creation, the device has been carried around the real world by researchers of the PIC4SeR.

B.1 A plastic shell with two eyes

The first step of the research was to design and construct a device able to continuously capture real-world scenes. Figure B.1 shows at the center of the photo collage the first prototype of the BabyAgent project. It is a cylinder shell printed in Polylactic Acid (PLA) featuring two HD cameras at the top. It is powered by a large power bank of 20.000 mAh and a Raspberry Pi 3B+ to carry out computations. A practical USB connector on the top of the device allows charging the battery at the end of the day.

The agent is programmed to capture images from its two cameras automatically. An internal algorithm continuously checks the similarity between previous pictures, and it discards new acquisitions if above a particular similarity score. After that, when the number of stereoscopic images acquired reaches a predefined value, the device exploits its network adapter and a known wireless network to send data to a central server through the File Transfer Protocol (FTP). The server acts as a data collector and training core. Indeed, once the device is in "sleep" mode, it trains the current model with data acquired during the day. Subsequently, it stores exploited images and brings some percentage of random samples from memory for future training.

The practical structure of the device allows bringing it to different places around the world easily. Three examples of acquired images are presented in Figure B.2. It is clear how real-world scenes usually feature multiple subjects or a not clear subject at all. Therefore, a learning algorithm should not only deal with the lack of labels but also with all those challenges brought by real-world scenes.

B.2 Methodology

For this first part of the BabyAgent research project, we focused on the training methodology and data, living aside the specific model architecture. Therefore, the reference backbone is not relevant. Nevertheless, even if the learning algorithm is adaptable for any deep learning architecture, for all experimentation, we adopt the DeiT architecture [238].

Inspired by [36] and [44], we develop a self-supervised methodology that is capable to learn representations from real-world data. The summary of the methodology is depicted in Figure B.3. As for [36], the training procedure includes a network g_{θ_m} and its past version, g_{θ_p}. The main aim of the training procedure is to make the model find coherence between its present and past representations.

We start with a pair of stereoscopic images that have a similar field of view but slightly different subjects depending on the depth of the scene. In order to solve the multi-object problem, we adopt the inner attention mechanism of the Past network to identify points of interest in the scene. In the specific case of the

Figure B.2: Example of scenes acquired by the first BabyAgent prototype. It is clear how a familiar real word scene contains multiple subjects or not a clear subject at all.

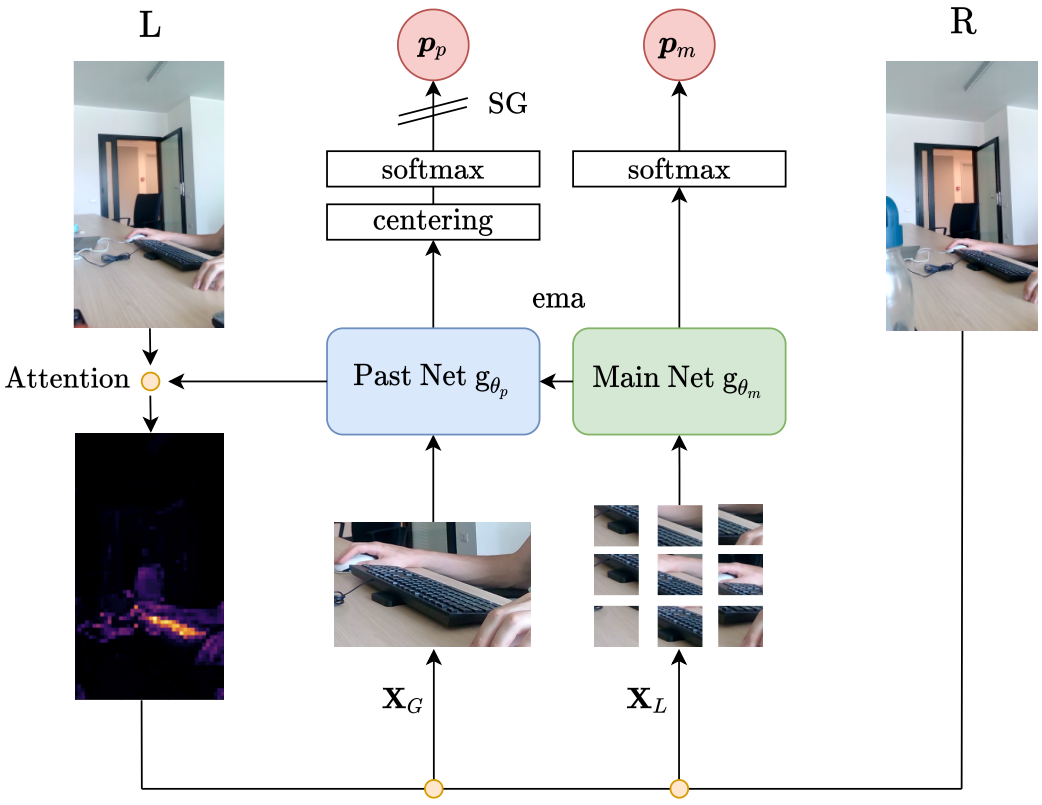


Figure B.3: Graphical representation of the self-supervised methodology that drives BabyAgent learning. It builds on [36] in which a network has to find coherence between its present and past representations. The attention of the Past network is exploited to find the most suitable crop for the scene. Subsequently, the past and main networks are fed with crops performed in the exact location on the left and right eyes, respectively.

DeiT architecture, we look at the attention map when using the `[CLS]` token as a query for the different heads. That produces an attention map for each head as shown in Figure B.4. We randomly select one of the heads, and then we fit a Gaussian distribution on the selected map to find the attention center and spread through the mean and standard deviation of the model. Subsequently, we exploit the acquired information to perform a focused crop of the left image, $\mathbf{X}_G$, and N small crops in the same location in the right acquisitions, $\mathbf{X}_L$. The latest feed the Main network, g_{θ_m} and the large crop feeds the Past network, g_{θ_p}. Both branches process their inputs, and through two projection heads with softmax placed on top of their respective networks, they produce $N+1$ probability distributions. Then, using the following loss function

$$\sum_{\mathbf{X}\in\mathbf{X}_L} -P_p(\mathbf{X}_G)\ln(P_m(\mathbf{X})) + \lambda \sum_{\mathbf{X}_i\in\mathbf{X}_L} -\ln \frac{1}{\sum_{j=1}^{N} \mathbb{1}_{[j\neq i]} e^{(\mathrm{sim}(\mathbf{X}_i,\mathbf{X}_k)/\tau)}} \tag{B.1}$$

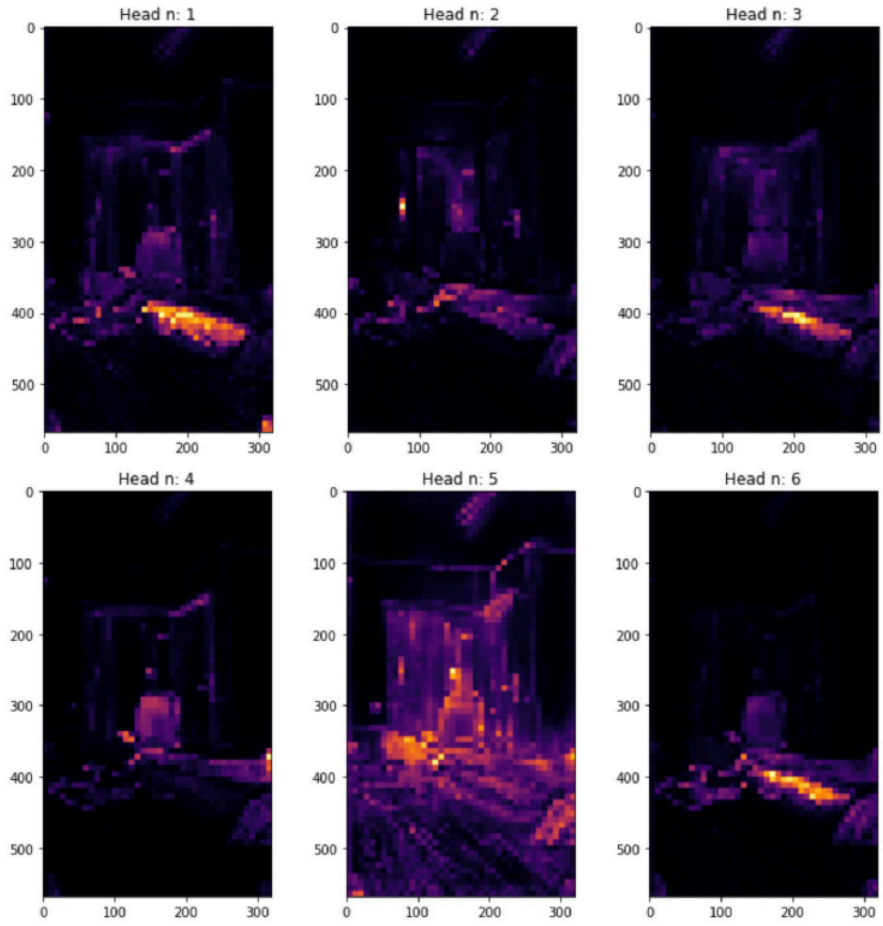


Figure B.4: Example of attention maps computed by the Past network after two months of training. We look at the attention map when using the `[CLS]` token as a query for the different heads in the last layer of the DeiT architecture. Each head focuses on a different portion of the scene.

it is possible to train the Main network through backpropagation. $\text{sim}(\boldsymbol{u}, \boldsymbol{v})$ denotes the dot product between l_2 normalized $\boldsymbol{u}$ and $\boldsymbol{v}$, $\text{sim}(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{v}/ \parallel \boldsymbol{u} \parallel\parallel \boldsymbol{v} \parallel$, P_p and P_m are the probability distributions produced by the Past and Main networks projection heads, respectively. Finally, λ is a balancing coefficient between the two loss contributions and τ denotes a temperature parameter. The first loss component is a cumulative cross-entropy function between the distribution produced by the Past network and all distributions generated by the Main network fed with the set of small local crops $\mathbf{X}_L$. On an intuitive point of view, it aims at bringing closer local patches representations with the global one, creating coherence between the Past and Main networks. On the other hand, the second contribution weighted by λ, pushes slightly apart all local patches among them. That is needed because local

patches could not present the same subject. Therefore, it acts as a regularization mechanism. Finally, the Past network is not updated by backpropagation (it is adopted a stop-gradient operation (SG)) but with an exponential moving average of the Main network weights: $\theta_p \leftarrow \alpha\theta_m + (1-\alpha)\theta_m$.

As in [36], in order to avoid collapse of the self-supervised training, probability distributions P_p generated by the Past network are centered and sharpened. The first is similar to a bias added to the output softmax and updated with an EMA; it prevents one dimension from dominating and encourages collapse to the uniform distribution. Therefore, to further avoid this problem, we follow the same data augmentations of BYOL [89] for global and local patches. On the other hand, the sharpening operation is obtained using a low value for the Past network softmax temperature parameter. It contrasts the centering effect, balancing the overall training.

B.3 Preliminary experiments

As previously stated, the presented research is an active project, and all data are intended as preliminary results. At the time of writing, the network has gone a dozen training iterations, and the device acquires new images every day.

B.3.1 Experimental settings

We started collecting data at the end of July 2021, acquiring an average of 600 images per day. The training procedure is triggered once 1.5k samples are acquired, and the device is gone offline for at least one hour. The memory recovery is set to 15% of the number of training samples. The network architecture selected is the DeiT-S [238]. For each new training, the dataset is bootstrapped for 50 epochs with a batch size of 128. The learning rate is set to 5e-4 with a cosine scheduler [246]. We adopt AdamW [159] as optimizer with a weight decay driven by a cosine scheduler that starts from 0.001 and ends at 0.2. The temperature of the Main net is set to 0.05 and λ to a fixed value of 0.1. We employ the PyTorch[1] framework to train the proposed network on a PC with 64-GB RAM, an Intel i7-9700K CPU, and two Nvidia 2080 Ti GP-GPU.

B.3.2 Evaluation protocol and learning follow-up

Without a standard dataset, it is impossible to assess the learning process with a validation or test set. Therefore, we adopt the standard protocol for evaluating self-supervised training: learn a linear classifier on top of frozen features of some

[1] https://pytorch.org/

Day	Linear						
	MNIST	**CvD**	**CIFAR**	**Art**	**Cartoon**	**Sketch**	**Photo**
0	0,8383	0,6632	0,1508	0,3125	0,6496	0,3542	0,5781
1	0,8539	0,6691	0,1714	0,4219	0,6830	0,4570	0,6750
2	0,8635	0,7324	0,1930	0,4297	0,6875	0,5065	0,7563
3	0,8793	0,7504	0,1972	0,4505	0,7054	0,5508	0,7563
4	0,8936	0,7313	0,2147	0,4479	0,6674	0,6185	0,7438
5	0,8786	0,7526	0,2107	0,4818	0,6875	0,6224	0,7781
6	0,8947	0,7237	0,2260	0,4948	0,6897	0,5326	0,7719
7	0,8803	0,7578	0,2300	0,4557	0,7076	0,4284	0,7625
8	0,8928	0,7541	0,2358	0,4948	0,6897	0,5703	0,7688
9	0,9012	0,7628	0,2492	0,5156	0,6987	0,5938	0,7763

Table B.1: BabyAgent learning progression over a temporal window of 10 days. A linear classifier is learned on top of frozen features for some control datasets.

control datasets. Moreover, in order to more directly verify the effectiveness of the latent space generated by the network, we fit a simple nearest neighbor classifier (k-NN) on a small percentage of samples of the control datasets. The nearest neighbor classifier then matches the feature of an image to the k nearest stored features, and it votes for the label. We adopt the cosine similarity as the distance metric. In contrast to other distance metrics, the cosine similarity loses a dimension, but it gives the possibility to accelerate the computation with GP-GPU drastically.

To follow-up the training procedure, we select four small datasets, MNIST[140], Cats vs Dogs[2], CIFAR10 [136] and the four domains of the PACS dataset [146]. Moreover, we also consider the validation sets of ImageNet [211], and CIFAR100 [136] in order to evaluate the network on more challenging tasks. Therefore, the new Main model is automatically assessed on those datasets with the evaluation protocol described before after a training session.

In Table B.1 and Table B.2 are reported the preliminary results for the linear and k-NN classifiers, respectively. In both cases, it is clear how the network is progressively learning more discriminating representations that let better separate classes and make a closer cluster of more similar features. It is worth noticing how representations are remarkably generic: selected control datasets have a very different nature, but an improvement is perceptible in all of them. Moreover, some preliminary tests on representations robustness showed encouraging results. Indeed, the network has also been tested applying rotations of different degrees (up to 40°) to all control datasets. Surprisingly, it was not observed a noticeable deviation: results have an average standard deviation of $\sigma = 0.01$.

[2] https://www.kaggle.com/c/dogs-vs-cats

Day	**k-NN**						
	MNIST	**CvD**	**CIFAR**	**Art**	**Cartoon**	**Sketch**	**Photo**
0	0,7358	0,6215	0,0825	0,3359	0,5714	0,4362	0,5406
1	0,7819	0,6615	0,1054	0,3724	0,5960	0,4609	0,6344
2	0,8312	0,6764	0,1271	0,4036	0,6317	0,4948	0,6938
3	0,8558	0,7096	0,1503	0,3906	0,6027	0,5130	0,7094
4	0,8646	0,7146	0,1549	0,4010	0,6496	0,5365	0,7063
5	0,8644	0,7177	0,1579	0,4036	0,6384	0,5273	0,7031
6	0,8576	0,7120	0,1642	0,4245	0,6429	0,5560	0,7000
7	0,8565	0,7214	0,1675	0,4036	0,6451	0,5586	0,7219
8	0,8624	0,7329	0,1774	0,4115	0,6496	0,5547	0,7125
9	0,8635	0,7331	0,1787	0,4271	0,6563	0,5742	0,7344

Table B.2: BabyAgent learning progression over a temporal window of 10 days. A simple nearest neighbor classifier (k-NN) is fitted on 10% of samples of the selected control datasets. The nearest neighbor classifier then matches the feature of an image to the k nearest stored features, and it votes for the label.

It has already been stressed many times: BabyAgent is an ongoing project, and the primary aim of this appendix is to share the idea behind this research. However, preliminary results show very promising data, giving confidence in its current form and future developments. Works on its evolution have already begun, scaling the proposed framework to real-world video sequences and adopting an architecture filled with the necessary priors to embed a visual model of our world efficiently.

List of acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Action Potential
AutoML	Automated machine learning
AcT	Action Transformer
ANOVA	Analysis of Variance
AP	Average Precision
ARC-PG	Adaptive Row Crops Path Generator
ASPP	Atrous Spatial Pyramid Pooling
BDRF	Bidirectional Reflectance Distribution Function
BERT	Bidirectional Encoder Representations from Transformers
BN	Batch Normalization
BOA	Bottom of Atmosphere
CE	Cross Entropy
CLS	Class
CNN	Convolutional Neural Network
CPU	Central Processing Unit
COCO	Common Objects in Context
COCO	Common Objects in Context
DeiT	Data Efficient Transformer
DF	Degree of Freedom
DG	Domain Generalization
DL	Deep Learning
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DVI	Difference Vegetation Index
EBM	Energy-Based Model
ELU	Exponential Linear Unit
ESA	European Space Agency
EMA	Exponential Moving Average
ERM	Empirical Risk Minimization

GAP	Global Average Pooling
GELU	Gaussian Error Linear Units
GNSS	Global Navigation Satellite System
GO	Graph Optimization
GPU	Graphics Processing Units
GP-GPU	General-Purpose Graphics Processing Units
GRU	Gated Recurrent Unit
GSD	Ground Sampling Distance
FC	Fully Connected
FN	False Negative
FNN	Feedforward Neural Network
FP	False Positive
FPGA	Field Programmable Gate Array
FR	Fault Rate
FTP	File Transfer Protocol
HAR	Human Action Recognition
HR	High-Resolution
HSI	Hyperspectral Imaging
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IMU	Inertial Measurement Units
IoU	Intersection over Unit
KL	Kullback-Leibler
k-NN	k-Nearest Neighbors
L	Soil Conditioning Index
LR	Low-Resolution
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MISR	Multi-Image Super-Resolution
ML	Machine Learning
MLP	Multi-Layer Perceptron
MMD	Maximum Mean Discrepancy
MNIST	Modified National Institute of Standards and Technology
MS	Magnetic Sensor
MS-G3D	Multi-Scale Graph 3D Convolution
MSE	Mean Squared Error
NAS	Neural Architecture Search
NCS	Neural Compute Stick
NDVI	Normal Difference Vegetation Index
NIR	Near-Infrared
NLP	Natural Language Processing
OOD	Out-Of-Distribution
PCA	Principal Component Analysis

PLA	Polylactic Acid
PSNR	Peak Signal-to-Noise Ratio
R	Red
RAM	Random Access Memory
RAMS	Residual Attention Multi-Image Super-Resolution
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RGB-D	RGB-Depth
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RRM	Residual Reduction Modules
RRT*	Rapidly-exploring Random Tree
RS	Remote Sensing
RSC	Row Segmentation Control
RTX	Ray Tracing Texel eXtreme
RTK	Real Time Kinematic
RVI	Ratio Vegetation Index
SAR	Synthetic Aperture Radar
SAV	Soil-Adjusted Vegetation Index
SISR	Single-Image Super-Resolution
SSL	Self-Supervised Learning
SNAP	Science Toolbox Exploitation Platform
SR	Super-Resolution
SS	Sum of Squares
SSIM	Structural Similarity Index Measure
ST-TR	Spatial Temporal Transformer Network
TLU	Threshold Logic Unit
TOPS	Trillion Operations per Second
TP	True Positive
TPU	Tensor Processing Units
t-SNE	t-distributed Stochastic Neighbor Embedding
UAS	Unmanned Aircraft System
UAV	Unmanned Aerial Vehicle.
UGV	Unmanned Ground Vehicle
URDF	Unified Robotic Description Format
URDF	Unified Robotic Description Format
USB	Universal Serial Bus
UWB	Ultra-WideBand
VOC	Vissual Object Classes
VPU	Vision Processing Units
WP	Weight Precision
WSE	Wafer-Scale Engine

Bibliography

[1] Martin Abadi et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[2] Pia Addabbo et al. "Contribution of Sentinel-2 data for applications in vegetation monitoring". In: (2016).

[3] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. "Autonomous Navigation in Vineyards with Deep Learning at the Edge". In: *International Conference on Robotics in Alpe-Adria Danube Region*. Springer. 2020, pp. 479–486.

[4] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. "Local motion planner for autonomous navigation in vineyards with a rgb-d camera-based algorithm and deep learning synergy". In: *Machines* 8.2 (2020), p. 27.

[5] Diego Aghi et al. "Deep semantic segmentation at the edge for autonomous navigation in vineyard rows". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 3421–3428.

[6] Subutai Ahmad and Jeff Hawkins. "Properties of sparse distributed representations and their application to hierarchical temporal memory". In: *arXiv preprint arXiv:1503.07469* (2015).

[7] Isabela Albuquerque et al. "Improving out-of-distribution generalization via multi-task self-supervised pretraining". In: *arXiv preprint arXiv:2003.13525* (2020).

[8] Khaliq Aleem. "Advancements in Multi-temporal Remote Sensing Data Analysis Techniques for Precision Agriculture". PhD thesis. Politecnico di Torino, 2020. URL: http://hdl.handle.net/11583/2839838.

[9] Simone Angarano et al. "Robust ultra-wideband range error mitigation with deep learning at the edge". In: *Engineering Applications of Artificial Intelligence* 102 (2021), p. 104278.

[10] Federico Angelini et al. "2D Pose-Based Real-Time Human Action Recognition With Occlusion-Handling". In: *IEEE Transactions on Multimedia* 22.6 (2020), pp. 1433–1446.

[11] Martin Arjovsky et al. “Invariant risk minimization”. In: *arXiv preprint arXiv:1907.02893* (2019).

[12] Devansh Arpit et al. “A closer look at memorization in deep networks”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 233–242.

[13] David Arthur and Sergei Vassilvitskii. *k-means++: The advantages of careful seeding*. Tech. rep. Stanford, 2006.

[14] Lei Jimmy Ba, Jamie Ryan Kiros, and Jeoffrey E. Hinton. “Layer Normalization”. In: *CoRR* abs/1607.06450 (2016).

[15] Yogesh Balaji, Swami Sankaranarayanan, and Rama Chellappa. “Metareg: Towards domain generalization using meta-regularization”. In: *Advances in Neural Information Processing Systems* 31 (2018), pp. 998–1008.

[16] Ron Banner et al. “Scalable methods for 8-bit training of neural networks”. In: *arXiv preprint arXiv:1805.11046* (2018).

[17] Andrew R Barron. “Universal approximation bounds for superpositions of a sigmoidal function”. In: *IEEE Transactions on Information theory* 39.3 (1993), pp. 930–945.

[18] Sara Beery, Grant Van Horn, and Pietro Perona. “Recognition in Terra Incognita”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[19] Pavel P Belonozhko. “Assembly and Service Robotic Space Module. Mathematical Model of the Reduced System”. In: *Cyber-Physical Systems: Advances in Design & Modelling*. Springer, 2020, pp. 337–347.

[20] Pawel Benecki et al. “Evaluating super-resolution reconstruction of satellite images”. In: *Acta Astronautica* 153 (2018), pp. 15–25.

[21] Axel Berg, Mark O’Connor, and Miguel Tairum Cruz. “Keyword Transformer: A Self-Attention Model for Keyword Spotting”. In: *Proc. Interspeech 2021*. 2021, pp. 4249–4253.

[22] Ran Bezen, Yael Edan, and Ilan Halachmi. “Computer vision system for measuring individual cow feed intake using RGB-D camera and deep learning algorithms”. In: *Computers and Electronics in Agriculture* 172 (2020), p. 105345.

[23] BK Bhattacharya and C Chattopadhyay. “A multi-stage tracking for mustard rot disease combining surface meteorology and satellite remote sensing”. In: *Computers and electronics in agriculture* 90 (2013), pp. 35–44.

[24] Mario Bijelic, Tobias Gruber, and Werner Ritter. “A benchmark for lidar sensors in fog: Is detection breaking down?” In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2018, pp. 760–767.

[25] Gilles Blanchard, Gyemin Lee, and Clayton Scott. "Generalizing from several related classification tasks to a new unlabeled sample". In: *Advances in neural information processing systems* 24 (2011), pp. 2178–2186.

[26] Pieter M. Blok et al. "Robot navigation in orchards with localization based on Particle filter and Kalman filter". In: *Computers and Electronics in Agriculture* 157 (2019), pp. 261–269. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2018.12.046. URL: https://www.sciencedirect.com/science/article/pii/S0168169918315230.

[27] Anselm Blumer et al. "Learnability and the Vapnik-Chervonenkis dimension". In: *Journal of the ACM (JACM)* 36.4 (1989), pp. 929–965.

[28] Anna Boschi et al. "A Cost-Effective Person-Following System for Assistive Unmanned Vehicles with Deep Learning at the Edge". In: *Machines* 8.3 (2020), p. 49.

[29] Olivier Bousquet and André Elisseeff. "Stability and generalization". In: *The Journal of Machine Learning Research* 2 (2002), pp. 499–526.

[30] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541.

[31] Adam Byerly, Tatiana Kalganova, and Ian Dear. "A branching and merging convolutional network with homogeneous filter capsules". In: *arXiv preprint arXiv:2001.09136* (2020).

[32] Jose Caballero et al. "Real-time video super-resolution with spatio-temporal networks and motion compensation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4778–4787.

[33] Flavio Callegati et al. "Autonomous Tracked Agricultural UGV Configuration and Navigation Experimental Results". In: 2018.

[34] James B Campbell and Randolph H Wynne. *Introduction to remote sensing*. Guilford Press, 2011.

[35] Zhe Cao et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields". In: *IEEE transactions on pattern analysis and machine intelligence* 43.1 (2019), pp. 172–186.

[36] Mathilde Caron et al. "Emerging Properties in Self-Supervised Vision Transformers". In: *CoRR* abs/2104.14294 (2021).

[37] Mathilde Caron et al. "Unsupervised learning of visual features by contrasting cluster assignments". In: *arXiv preprint arXiv:2006.09882* (2020).

[38] Isabel Luisa Castillejo-Gonzalez et al. "Object-and pixel-based analysis for mapping crops and their agro-environmental associated measures using QuickBird imagery". In: *Computers and Electronics in Agriculture* 68.2 (2009), pp. 207–215.

[39] Miguel Gumersindo Castro Gomez. "Joint use of Sentinel-1 and Sentinel-2 for land cover classification: A machine learning approach". In: *Lund University GEM thesis series* (2017).

[40] Simone Cerrato et al. *A Deep Learning Driven Algorithmic Pipeline for Autonomous Navigation in Row-Based Crops*. 2021. arXiv: 2112.03816 [cs.RO].

[41] Simone Cerrato et al. "An Adaptive Row Crops Path Generator with Deep Learning Synergy". In: *2021 6th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*. IEEE. 2021, pp. 6–12.

[42] Liang-Chieh Chen et al. "Rethinking atrous convolution for semantic image segmentation". In: *arXiv preprint arXiv:1706.05587* (2017).

[43] Mia Xu Chen et al. "The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation". In: *CoRR* abs/1804.09849 (2018).

[44] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[45] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[46] Jaewoong Choi et al. "Attention routing between capsules". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2019.

[47] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[48] Anna Choromanska et al. "The loss surfaces of multilayer networks". In: *Artificial intelligence and statistics*. PMLR. 2015, pp. 192–204.

[49] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3642–3649.

[50] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[51] Lorenzo Comba, Paolo Gay, and Davide Ricauda Aimonino. "Robot ensembles for grafting herbaceous crops". In: *Biosystems engineering* 146 (2016), pp. 227–239.

[52] Lorenzo Comba et al. "Leaf Area Index evaluation in vineyards using 3D point clouds from UAV imagery". In: *Precision Agriculture* 21.4 (2020), pp. 881–896.

[53] Lorenzo Comba et al. "Unsupervised detection of vineyards by 3D point-cloud UAV photogrammetry for precision agriculture". In: *Computers and Electronics in Agriculture* 155 (2018), pp. 84–95.

[54] Lorenzo Comba et al. "Vineyard detection from unmanned aerial systems images". In: *computers and Electronics in Agriculture* 114 (2015), pp. 78–87.

[55] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[56] Brett A Cruden, Dinesh Prabhu, and Ramon Martinez. "Absolute radiation measurement in venus and mars entry conditions". In: *Journal of Spacecraft and Rockets* 49.6 (2012), pp. 1069–1079.

[57] Stéphane d'Ascoli et al. "Convit: Improving vision transformers with soft convolutional inductive biases". In: *arXiv preprint arXiv:2103.10697* (2021).

[58] Tao Dai et al. "Second-order attention network for single image super-resolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11065–11074.

[59] Kahneman Daniel. *Thinking, fast and slow*. 2017.

[60] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[61] Deepak Deshmukh et al. "Design and Development of Intelligent Pesticide Spraying System for Agricultural Robot". In: *International Conference on Hybrid Intelligent Systems*. Springer. 2020, pp. 157–170.

[62] Michel Deudon et al. "HighRes-net: Recursive Fusion for Multi-Frame Super-Resolution of Satellite Imagery". In: *arXiv preprint arXiv:2002.06460* (2020).

[63] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186.

[64] Chao Dong et al. "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2015), pp. 295–307.

[65] Linhao Dong, Shuang Xu, and Bo Xu. "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5884–5888. DOI: 10.1109/ICASSP.2018.8462506.

[66] Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. "Attention is not all you need: Pure attention loses rank doubly exponentially with depth". In: *arXiv preprint arXiv:2103.03404* (2021).

[67] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

[68] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

[69] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[70] Tom Duckett et al. "Agricultural robotics: the future of robotic agriculture". In: *arXiv preprint arXiv:1806.06762* (2018).

[71] Abhishek Dutta and Andrew Zisserman. "The VIA Annotation Software for Images, Audio and Video". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: ACM, 2019. DOI: 10.1145/3343031.3350535. URL: https://doi.org/10.1145/3343031.3350535.

[72] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. "Training generative neural networks via maximum mean discrepancy optimization". In: *arXiv preprint arXiv:1505.03906* (2015).

[73] Thomas Eppenberger et al. "Leveraging stereo-camera data for real-time dynamic obstacle detection and tracking". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10528–10535.

[74] Mark Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[75] Chen Fang, Ye Xu, and Daniel N. Rockmore. "Unbiased Metric Learning: On the Utilization of Multiple Datasets and Web Images for Softening Bias". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2013.

[76] Sina Farsiu et al. "Fast and robust multiframe super resolution". In: *IEEE transactions on image processing* 13.10 (2004), pp. 1327–1344.

[77] Aijing Feng et al. "Yield estimation in cotton using UAV-based multi-sensor imagery". In: *Biosystems Engineering* 193 (2020), pp. 101–114.

[78] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[79] Yaroslav Ganin et al. "Domain-adversarial training of neural networks". In: *The journal of machine learning research* 17.1 (2016), pp. 2096–2030.

[80] Aurelien Geron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[81] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10 (2000), pp. 2451–2471.

[82] Muhammad Ghifary et al. "Domain generalization for object recognition with multi-task autoencoders". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2551–2559.

[83] Amir Gholami et al. "A survey of quantization methods for efficient neural network inference". In: *arXiv preprint arXiv:2103.13630* (2021).

[84] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[85] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[86] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[87] Benjamin Graham et al. "LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference". In: *CoRR* abs/2104.01136 (2021).

[88] Arthur Gretton et al. "A kernel two-sample test". In: *The Journal of Machine Learning Research* 13.1 (2012), pp. 723–773.

[89] Jean-Bastien Grill et al. "Bootstrap your own latent: A new approach to self-supervised learning". In: *arXiv preprint arXiv:2006.07733* (2020).

[90] Ishaan Gulrajani and David Lopez-Paz. "In Search of Lost Domain Generalization". In: *International Conference on Learning Representations*. 2021. URL: https://openreview.net/forum?id=lQdXeXDoWtI.

[91] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[92] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[93] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[94] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[95] Kaiming He et al. "Mask r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2961–2969.

[96] Kaiming He et al. "Momentum contrast for unsupervised visual representation learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 9729–9738.

[97] Dan Hendrycks and Kevin Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

[98] Felix Hill et al. "Environmental drivers of systematicity and generalization in a situated agent". In: *arXiv preprint arXiv:1910.00571* (2019).

[99] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[100] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. "Transforming auto-encoders". In: *International conference on artificial neural networks*. Springer. 2011, pp. 44–51.

[101] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. "Matrix capsules with EM routing". In: *International conference on learning representations*. 2018.

[102] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[103] Kjell Jørgen Hole and Subutai Ahmad. "A thousand brains: toward biologically constrained AI". In: *SN Applied Sciences* 3.8 (2021), pp. 1–14.

[104] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[105] Andrew Howard et al. "Searching for mobilenetv3". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1314–1324.

[106] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[107] Ping Hu et al. "Real-time semantic segmentation with fast attention". In: *IEEE Robotics and Automation Letters* 6.1 (2020), pp. 263–270.

[108] Gao Huang et al. "Snapshot ensembles: Train 1, get m for free". In: *arXiv preprint arXiv:1704.00109* (2017).

[109] Linjiang Huang et al. "Part-aligned pose-guided recurrent network for action recognition". In: *Pattern Recognition* 92 (2019), pp. 165–176. ISSN: 0031-3203.

[110] Wenkai Huang and Fobao Zhou. "DA-CapsNet: dual attention mechanism capsule network". In: *Scientific Reports* 10.1 (2020), pp. 1–13.

[111] Zeyi Huang et al. "Self-challenging improves cross-domain generalization". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 124–140.

[112] Itay Hubara et al. "Binarized neural networks". In: *Advances in neural information processing systems* 29 (2016).

[113] Alfredo R Huete. "A soil-adjusted vegetation index (SAVI)". In: *Remote sensing of environment* 25.3 (1988), pp. 295–309.

[114] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[115] Michal Irani and Shmuel Peleg. "Improving resolution by image registration". In: *CVGIP: Graphical models and image processing* 53.3 (1991), pp. 231–239.

[116] Benoit Jacob et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2704–2713.

[117] Jan Jelének, Veronika Kopačková, and Kateřina Fárová. "Post-earthquake landslide distribution assessment using sentinel-1 and-2 data: The example of the 2016 mw 7.8 earthquake in New Zealand". In: *Multidisciplinary Digital Publishing Institute Proceedings*. Vol. 2. 7. 2018, p. 361.

[118] Younghyun Jo et al. "Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3224–3232.

[119] Carl F Jordan. "Derivation of leaf-area index from quality of light on the forest floor". In: *Ecology* 50.4 (1969), pp. 663–666.

[120] Norman P Jouppi et al. "In-datacenter performance analysis of a tensor processing unit". In: *Proceedings of the 44th annual international symposium on computer architecture*. 2017, pp. 1–12.

[121] Armin Kappeler et al. "Video super-resolution with convolutional neural networks". In: *IEEE Transactions on Computational Imaging* 2.2 (2016), pp. 109–122.

[122] Fazle Karim et al. "Multivariate LSTM-FCNs for time series classification". In: *Neural Networks* 116 (2019), pp. 237–245. ISSN: 0893-6080.

[123] Arnon Karnieli et al. "Use of NDVI and land surface temperature for drought assessment: Merits and limitations". In: *Journal of climate* 23.3 (2010), pp. 618–633.

[124] Yoram J Kaufman and Claudia Sendra. "Algorithm for automatic atmospheric corrections to visible and near-IR satellite imagery". In: *International Journal of Remote Sensing* 9.8 (1988), pp. 1357–1381.

[125] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. "Generalization in deep learning". In: *arXiv preprint arXiv:1710.05468* (2017).

[126] Michal Kawulok et al. "Deep learning for multiple-image super-resolution". In: *IEEE Geoscience and Remote Sensing Letters* (2019).

[127] Michal Kawulok et al. "Evolving imaging model for super-resolution reconstruction". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2018, pp. 284–285.

[128] Aleem Khaliq, Vittorio Mazzia, and Marcello Chiaberge. "Refining satellite imagery by using UAV imagery for vineyard environment: A CNN Based approach". In: *2019 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. 2019, pp. 25–29. DOI: 10.1109/MetroAgriFor.2019.8909276.

[129] Aleem Khaliq, Vittorio Mazzia, and Marcello Chiaberge. "Refining satellite imagery by using UAV imagery for vineyard environment: A CNN Based approach". In: *2019 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. 2019, pp. 25–29. DOI: 10.1109/MetroAgriFor.2019.8909276.

[130] Aleem Khaliq, Vittorio Mazzia, and Marcello Chiaberge. "Refining satellite imagery by using UAV imagery for vineyard environment: A CNN Based approach". In: *2019 IEEE International Workshop on Metrology for Agriculture and Forestry (MetroAgriFor)*. IEEE. 2019, pp. 25–29.

[131] Aleem Khaliq et al. "Comparison of satellite and UAV-based multispectral imagery for vineyard variability assessment". In: *Remote Sensing* 11.4 (2019), p. 436.

[132] Daehee Kim et al. “Selfreg: Self-supervised contrastive regularization for domain generalization”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 9619–9628.

[133] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[134] Gunter Klambauer et al. “Self-normalizing neural networks”. In: *Proceedings of the 31st international conference on neural information processing systems*. 2017, pp. 972–981.

[135] Kamran Kowsari et al. “Rmdl: Random multimodel deep learning for classification”. In: *Proceedings of the 2nd International Conference on Information System and Data Mining*. 2018, pp. 19–28.

[136] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[137] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[138] K Manikanda Kumaran and M Chinnadurai. “Cloud-based robotic system for crowd control in smart cities using hybrid intelligent generic algorithm”. In: *Journal of Ambient Intelligence and Humanized Computing* 11.12 (2020), pp. 6293–6306.

[139] Wei-Sheng Lai et al. “Fast and accurate image super-resolution with deep laplacian pyramid networks”. In: *IEEE transactions on pattern analysis and machine intelligence* 41.11 (2018), pp. 2599–2613.

[140] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann. lecun. com/exdb/mnist/* (1998).

[141] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[142] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage”. In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[143] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

[144] Christian Ledig et al. “Photo-realistic single image super-resolution using a generative adversarial network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4681–4690.

[145] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural networks* 6.6 (1993), pp. 861–867.

[146] Da Li et al. “Deeper, Broader and Artier Domain Generalization”. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[147] Fengfu Li, Bo Zhang, and Bin Liu. “Ternary weight networks”. In: *arXiv preprint arXiv:1605.04711* (2016).

[148] Hao Li et al. “Visualizing the Loss Landscape of Neural Nets”. In: ().

[149] Haoliang Li et al. “Domain generalization with adversarial feature learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 5400–5409.

[150] Wenbin Li and Matthieu Liewig. “A survey of AI accelerators for edge environment”. In: *World Conference on Information Systems and Technologies*. Springer. 2020, pp. 35–44.

[151] Bee Lim et al. “Enhanced deep residual networks for single image super-resolution”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2017, pp. 136–144.

[152] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network”. In: *arXiv preprint arXiv:1312.4400* (2013).

[153] Tsung-Yi Lin et al. “Microsoft coco: Common objects in context”. In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[154] Wei Liu et al. “Ssd: Single shot multibox detector”. In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[155] Ziyu Liu et al. “Disentangling and unifying graph convolutions for skeleton-based action recognition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 143–152.

[156] Mingsheng Long et al. “Deep transfer learning with joint adaptation networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 2208–2217.

[157] Antonio Loquercio et al. “Learning high-speed flight in the wild”. In: *Science Robotics* 6.59 (2021), eabg5810.

[158] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations*. 2019.

[159] Ilya Loshchilov and Frank Hutter. “Fixing weight decay regularization in adam”. In: (2018).

[160] David G Lowe. “Object recognition from local scale-invariant features”. In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[161] Diogo Luvizon, David Picard, and Hedi Tabia. "Multi-task deep learning for real-time 3D human pose estimation and action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* (2020).

[162] Angelos Mallios et al. "Underwater caves sonar data set". In: *The International Journal of Robotics Research* 36.12 (2017), pp. 1247–1251.

[163] Mmamokoma Grace Maponya, Adriaan Van Niekerk, and Zama Eric Mashimbye. "Pre-harvest classification of crop types using a Sentinel-2 time-series and machine learning". In: *Computers and Electronics in Agriculture* 169 (2020), p. 105164.

[164] Marcus Märtens et al. "Super-resolution of PROBA-V images using convolutional neural networks". In: *Astrodynamics* 3.4 (2019), pp. 387–402.

[165] Mauro Martini et al. "Domain-Adversarial Training of Self-Attention-Based Networks for Land Cover Classification Using Multi-Temporal Sentinel-2 Satellite Imagery". In: *Remote Sensing* 13.13 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13132564. URL: https://www.mdpi.com/2072-4292/13/13/2564.

[166] Jiri Matas, Charles Galambos, and Josef Kittler. "Robust detection of lines using the progressive probabilistic hough transform". In: *Computer vision and image understanding* 78.1 (2000), pp. 119–137.

[167] Vittorio Mazzia, Fred Daneshgaran, and Marina Mondin. "Use of Deep Learning for Automatic Detection of Cracks in Tunnels". In: *Progresses in Artificial Intelligence and Neural Systems*. Springer, 2021, pp. 91–101.

[168] Vittorio Mazzia, Aleem Khaliq, and Marcello Chiaberge. "Improvement in land cover and crop classification based on temporal features learning from Sentinel-2 data using recurrent-convolutional neural network (R-CNN)". In: *Applied Sciences* 10.1 (2020), p. 238.

[169] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. "Efficient-CapsNet: capsule network with self-attention routing". In: *Scientific Reports* 11 (2021).

[170] Vittorio Mazzia et al. "Action Transformer: A self-attention model for short-time pose-based human action recognition". In: *Pattern Recognition* 124 (2022), p. 108487.

[171] Vittorio Mazzia et al. "DeepWay: A Deep Learning waypoint estimator for global path generation". In: *Computers and Electronics in Agriculture* 184 (2021), p. 106091.

[172] Vittorio Mazzia et al. "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application". In: *IEEE Access* 8 (2020), pp. 9102–9114.

[173] Vittorio Mazzia et al. “UAV and machine learning based refinement of a satellite-driven vegetation index for precision agriculture”. In: *Sensors* 20.9 (2020), p. 2530.

[174] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[175] Diganta Misra. “Mish: A self regularized non-monotonic neural activation function”. In: *arXiv preprint arXiv:1908.08681* (2019).

[176] Mehryar Mohri and Afshin Rostamizadeh. “Rademacher complexity bounds for non-iid processes”. In: (2009).

[177] Andrea Bordone Molini et al. “DeepSUM: Deep neural network for Super-resolution of Unregistered Multitemporal images”. In: *IEEE Transactions on Geoscience and Remote Sensing* (2019).

[178] Andrea Bordone Molini et al. “DeepSUM++: Non-local Deep Neural Network for Super-Resolution of Unregistered Multitemporal Images”. In: *arXiv preprint arXiv:2001.06342* (2020).

[179] M Susan Moran, Yoshio Inoue, and EM Barnes. “Opportunities and limitations for image-based remote sensing in precision crop management”. In: *Remote sensing of Environment* 61.3 (1997), pp. 319–346.

[180] Krikamol Muandet, David Balduzzi, and Bernhard Schölkopf. “Domain generalization via invariant feature representation”. In: *International Conference on Machine Learning*. PMLR. 2013, pp. 10–18.

[181] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[182] Preetum Nakkiran, Behnam Neyshabur, and Hanie Sedghi. “The Deep Bootstrap Framework: Good Online Learners are Good Offline Generalizers”. In: *arXiv preprint arXiv:2010.08127* (2020).

[183] Yurii Nesterov. “A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)”. In: *Doklady an ussr*. Vol. 269. 1983, pp. 543–547.

[184] Nanne van Noord and Eric Postma. “A learned representation of artist-specific colourisation”. In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2907–2915.

[185] Genevieve B Orr and Klaus-Robert Muller. *Neural networks: tricks of the trade*. Springer, 2003.

[186] Dirk Padfield. “Masked object registration in the Fourier domain”. In: *IEEE Transactions on image processing* 21.5 (2011), pp. 2706–2718.

[187] Joan-Cristian Padro et al. "Comparison of four UAV georeferencing methods for environmental monitoring purposes focusing on the combined use with airborne and satellite remote sensing platforms". In: *International journal of applied earth observation and geoinformation* 75 (2019), pp. 130–140.

[188] Luis Padua et al. "Vineyard variability analysis through UAV-based vigour maps to assess climate change impacts". In: *Agronomy* 9.10 (2019), p. 581.

[189] Gonzalo Pajares. "Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs)". In: *Photogrammetric Engineering & Remote Sensing* 81.4 (2015), pp. 281–330.

[190] George Papandreou et al. "Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 269–286.

[191] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[192] Xingchao Peng et al. "Moment Matching for Multi-Source Domain Adaptation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[193] Leo Pichon et al. "What relevant information can be identified by experts on unmanned aerial vehicles' visible images for precision viticulture?" In: *Precision Agriculture* 20.2 (2019), pp. 278–294.

[194] Francis J Pierce and Peter Nowak. "Aspects of precision agriculture". In: *Advances in agronomy* 67 (1999), pp. 1–85.

[195] Chiara Plizzari, Marco Cannici, and Matteo Matteucci. "Skeleton-based action recognition via spatial and temporal transformer networks". In: *Computer Vision and Image Understanding* 208 (2021), p. 103219.

[196] Daniela Poli and Thierry Toutin. "Review of developments in geometric modelling for high resolution satellite pushbroom sensors". In: *The Photogrammetric Record* 27.137 (2012), pp. 58–73.

[197] Antonio Polino, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization". In: *arXiv preprint arXiv:1802.05668* (2018).

[198] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *Ussr computational mathematics and mathematical physics* 4.5 (1964), pp. 1–17.

[199] Josiah Radcliffe, Julie Cox, and Duke M. Bulanon. "Machine vision for orchard navigation". In: *Computers in Industry* 98 (2018), pp. 165–171. ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2018.03.008.

[200] Panagiotis Radoglou-Grammatikis et al. "A compilation of UAV applications for precision agriculture". In: *Computer Networks* 172 (2020), p. 107148.

[201] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[202] Douglas A Reynolds. "Gaussian Mixture Models." In: *Encyclopedia of biometrics* 741 (2009), pp. 659–663.

[203] Fabio De Sousa Ribeiro, Georgios Leontidis, and Stefanos D Kollias. "Capsule Routing via Variational Bayes." In: *AAAI*. 2020, pp. 3749–3756.

[204] Arthur J Richardson and CL Wiegand. "Distinguishing vegetation from soil background information". In: *Photogrammetric engineering and remote sensing* 43.12 (1977), pp. 1541–1552.

[205] Rudolf Richter et al. "Correction of cirrus effects in Sentinel-2 type of imagery". In: *International journal of remote sensing* 32.10 (2011), pp. 2931–2941.

[206] Itsaso Rodriguez-Moreno et al. "Shedding light on people action recognition in social robotics by means of common spatial patterns". In: *Sensors* 20.8 (2020), p. 2436.

[207] Juan Jesús Roldán et al. "Robots in agriculture: State of art and practical experiences". In: *Service robots* (2018), pp. 67–90.

[208] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[209] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[210] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[211] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[212] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. "Dynamic routing between capsules". In: *arXiv preprint arXiv:1710.09829* (2017).

[213] Shiori Sagawa et al. "Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization". In: *arXiv preprint arXiv:1911.08731* (2019).

[214] Francesco Salvetti et al. “Multi-image super resolution of remotely sensed images using residual attention deep neural networks”. In: *Remote Sensing* 12.14 (2020), p. 2207.

[215] Julian Schrittwieser et al. “Mastering atari, go, chess and shogi by planning with a learned model”. In: *Nature* 588.7839 (2020), pp. 604–609.

[216] Ramprasaath R Selvaraju et al. “Grad-CAM: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 618–626.

[217] Soroosh Shahtalebi et al. “SAND-mask: An Enhanced Gradient Masking Strategy for the Discovery of Invariances in Domain Generalization”. In: *arXiv preprint arXiv:2106.02266* (2021).

[218] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[219] Shiv Shankar et al. “Generalizing Across Domains via Cross-Gradient Training”. In: *International Conference on Learning Representations*. 2018.

[220] Wenzhe Shi et al. “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1874–1883.

[221] Connor Shorten and Taghi M Khoshgoftaar. “A survey on image data augmentation for deep learning”. In: *Journal of Big Data* 6.1 (2019), pp. 1–48.

[222] Leslie N Smith. “A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay”. In: *arXiv preprint arXiv:1803.09820* (2018).

[223] Leslie N Smith. “Cyclical learning rates for training neural networks”. In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 464–472.

[224] Robert Sparrow and Mark Howard. “Robots in agriculture: prospects, impacts, ethics, and policy”. In: *Precision Agriculture* 22.3 (2021), pp. 818–833.

[225] Ancha Srinivasan. *Handbook of precision agriculture: principles and applications*. CRC press, 2006.

[226] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[227] Nikko Ström. "Phoneme probability estimation with dynamic sparsely connected artificial neural networks". In: *The Free Speech Journal* 5.1-41 (1997), p. 2.

[228] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and policy considerations for deep learning in NLP". In: *arXiv preprint arXiv:1906.02243* (2019).

[229] Baochen Sun and Kate Saenko. "Deep CORAL: Correlation alignment for deep domain adaptation". In: *European conference on computer vision*. Springer. 2016, pp. 443–450.

[230] Enrico Sutera. et al. "Indoor Point-to-Point Navigation with Deep Reinforcement Learning and Ultra-Wideband". In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,* INSTICC. SciTePress, 2021, pp. 38–47. ISBN: 978-989-758-484-8. DOI: 10.5220/0010202600380047.

[231] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[232] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[233] Mingxing Tan and Quoc Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 6105–6114.

[234] S. Tejaswi Digumarti et al. "An Approach for Semantic Segmentation of Tree-like Vegetation". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 1801–1807. DOI: 10.1109/ICRA.2019.8793576.

[235] Radu Timofte, Rasmus Rothe, and Luc Van Gool. "Seven ways to improve example-based single image super resolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1865–1873.

[236] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2017, pp. 23–30.

[237] Charles Toth and Grzegorz Jóźków. "Remote sensing platforms and sensors: A survey". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 115 (2016), pp. 22–36.

[238] Hugo Touvron et al. “Training data-efficient image transformers & distillation through attention”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 10347–10357.

[239] Jonathan Tremblay et al. “Training deep networks with synthetic data: Bridging the reality gap by domain randomization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 969–977.

[240] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. “Instance normalization: The missing ingredient for fast stylization”. In: *arXiv preprint arXiv:1607.08022* (2016).

[241] Leslie Valiant. *Probably Approximately Correct: NatureÕs Algorithms for Learning and Prospering in a Complex World*. Basic Books (AZ), 2013.

[242] Diego Valsesia and Enrico Magli. “A novel rate control algorithm for onboard predictive coding of multispectral and hyperspectral images”. In: *IEEE Transactions on Geoscience and Remote Sensing* 52.10 (2014), pp. 6341–6355.

[243] Laurens Van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE.” In: *Journal of machine learning research* 9.11 (2008).

[244] Rufin VanRullen and Christof Koch. “Is perception discrete or continuous?” In: *Trends in cognitive sciences* 7.5 (2003), pp. 207–213.

[245] Vladimir N Vapnik and A Ya Chervonenkis. “On the uniform convergence of relative frequencies of events to their probabilities”. In: *Measures of complexity*. Springer, 2015, pp. 11–30.

[246] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.

[247] Hemanth Venkateswara et al. “Deep Hashing Network for Unsupervised Domain Adaptation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[248] Ivan Vidović and Rudolf Scitovski. “Center-based clustering for line detection and application to crop rows detection”. In: *Computers and electronics in agriculture* 109 (2014), pp. 212–220.

[249] Matheus Alves Vieira et al. “Object based image analysis and data mining applied to a remotely sensed Landsat time-series to map sugarcane over large areas”. In: *Remote Sensing of Environment* 123 (2012), pp. 553–562.

[250] Li Wan et al. “Regularization of neural networks using dropconnect”. In: *International conference on machine learning*. 2013, pp. 1058–1066.

[251] Jun-Wei Wang et al. “Dynamic plume tracking by cooperative robots”. In: *IEEE/ASME Transactions on Mechatronics* 24.2 (2019), pp. 609–620.

[252] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity”. In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[253] Pete Warden and Daniel Situnayake. *TinyML*. O’Reilly Media, Incorporated, 2019.

[254] Ross Wightman, Hugo Touvron, and Hervé Jégou. “ResNet strikes back: An improved training procedure in timm”. In: *arXiv preprint arXiv:2110.00476* (2021).

[255] Sanghyun Woo et al. “Cbam: Convolutional block attention module”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[256] Yuxin Wu and Kaiming He. “Group normalization”. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[257] Edgar Xi, Selina Bing, and Yang Jin. “Capsule network performance on complex data”. In: *arXiv preprint arXiv:1712.03480* (2017).

[258] Bing Xu et al. “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv preprint arXiv:1505.00853* (2015).

[259] Huan Xu and Shie Mannor. “Robustness and generalization”. In: *Machine learning* 86.3 (2012), pp. 391–423.

[260] Jinru Xue and Baofeng Su. “Significant remote sensing vegetation indices: A review of developments and applications”. In: *Journal of sensors* 2017 (2017).

[261] Shen Yan et al. “Improve unsupervised domain adaptation with mixup training”. In: *arXiv preprint arXiv:2001.00677* (2020).

[262] Fuzhi Yang et al. “Learning Texture Transformer Network for Image Super-Resolution”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 5791–5800.

[263] Changqian Yu et al. “Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation”. In: *arXiv preprint arXiv:2004.02147* (2020).

[264] Jiahui Yu et al. “Wide activation for efficient and accurate image super-resolution”. In: *arXiv preprint arXiv:1808.08718* (2018).

[265] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[266] Chunhua Zhang and John M Kovacs. "The application of small unmanned aerial systems for precision agriculture: a review". In: *Precision agriculture* 13.6 (2012), pp. 693–712.

[267] GuoSheng Zhang et al. "Assessment of rice leaf blast severity using hyperspectral imaging during late vegetative growth". In: *Australasian Plant Pathology* 49.5 (2020), pp. 571–578.

[268] Marvin Mengxin Zhang et al. "Adaptive risk minimization: A meta-learning approach for tackling group shift". In: *International Conference on Learning Representations*. 2020.

[269] Yulun Zhang et al. "Image super-resolution using very deep residual channel attention networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 286–301.

[270] Zichao Zhang and Davide Scaramuzza. "Perception-aware receding horizon navigation for MAVs". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2534–2541.

[271] Kaiyang Zhou et al. "Domain generalization: A survey". In: *arXiv preprint arXiv:2103.02503* (2021).

[272] Michael Zhu and Suyog Gupta. "To prune, or not to prune: exploring the efficacy of pruning for model compression". In: *arXiv preprint arXiv:1710.01878* (2017).

[273] Jurgen Zoto et al. "Automatic path planning for unmanned ground vehicle using uav imagery". In: *International Conference on Robotics in Alpe-Adria Danube Region*. Springer. 2019, pp. 223–230.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.