



Politecnico
di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

PIC4SeR

Doctoral Dissertation

Doctoral Program in Electric, Electronic and Communication Engineering
(37th cycle)

Efficient and Robust Deep Learning for Robotics

**Bridging State-of-the-Art AI and Real-World
Constraints for Perception and Navigation**

By

Simone Angarano

Supervisor:
Prof. Marcello Chiaberge

Politecnico di Torino
2025

Declaration

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

Simone Angarano
2025

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

It is not enough that you should understand about applied science in order that your work may increase man's blessings. Concern for man himself and his fate must always form the chief interest of all technical endeavors; concern for the great unsolved problems of the organization of labor and the distribution of goods in order that the creations of our mind shall be a blessing and not a curse to mankind. Never forget this in the midst of your diagrams and equations.

Albert Einstein

*Address to students of CalTech
Pasadena, California (16 Feb 1931)*

Acknowledgements

The work presented in this thesis would not have been possible without the support, guidance, and encouragement of many people to whom I am deeply grateful.

First, I would like to thank Professor Marcello Chiaberge for the trust and respect he has shown me over the years. His support and confidence in my abilities have been fundamental in the development of this work and my growth as a researcher.

I am also deeply grateful for all the wonderful people I met at PIC4SeR who have accompanied me throughout these years. Their undeserved presence, collaboration, and friendship have helped me fully enjoy the most exciting aspects of research and not beat myself up during difficult moments. I cannot mention them all, but I especially want to thank:

- Vitto and Fra for mentoring me from day one and including me in their fun projects when I was a total newbie. I hope I absorbed at least some of their wisdom and passion;
- Mauro for writing with me a whopping 12 papers (plus endless review rounds) without ever giving up. He has been a constant reference point for me, and I hope to see him be the next Marcello soon;
- Gigi, Ale, Andre E., and Mauro for founding a band that still doesn't have a proper name (a.k.a. the PIC4Jam) but keeps rocking;
- Andre O., Umbi, Chiara, Fra M., Marco, Gianlu, Carlo, Brenno, Jack, Edo, Fra L.C., Ste C., Fede, Ariel, Xia, Ste T., Giorgio, Matti *et al.* for being the friendly faces I needed countless times (and for many drinks and grigliate).

I would also like to mention some of the colleagues and friends I met during my visiting period at UT Austin: Professor Atlas Wang, Scott, Neel, Hannah, Mansoor, Salim, Jorge, Gui, Felipe, Clint, Jen, Andrea, Giulia, and the kids. They have supported me and made me feel at home. Hook'em!

Speaking of home, I want to thank my parents, brothers, grandparents, and all my family in Pescara. There are no words to express the immense gratitude I have for all the constant, unconditional love I received from them despite the distance. They have made me who I am.

I also want to thank my friends in Torino, who have been my family for the last 10 years here and have taught me to dive headfirst into life and live it without holding back. They helped me become the man I am today (the names would be too many to list, but you know who you are).

Finally, I would like to thank my wife, Costanza, for being by my side throughout all of this and so much more. Through her love, she always sees more in me than I see in myself, and she takes every part of me seriously. I am deeply grateful for how much we have grown together, and I am profoundly happy to walk through whatever life brings with her.

Abstract

In recent years, the advancement of deep learning research has driven the creation of large models capable of performing complex tasks with high accuracy. However, it is crucial to efficiently apply these advancements in fields such as robotics, where computational constraints are crucial.

The integration of artificial intelligence in robotics has revolutionized the capabilities of autonomous systems across various domains, from domestic assistance to precision agriculture. However, deploying state-of-the-art deep learning models in robotic applications presents significant challenges. Computational constraints, strict latency requirements, and the need for robust performance in real-world scenarios create a pressing need for efficient yet powerful AI solutions that can be practically implemented on resource-limited robotic platforms.

This dissertation addresses these challenges head-on, developing innovative deep learning architectures and training methodologies that bridge the gap between cutting-edge advancements and the practical realities of service robotics. The key focus is enhancing efficiency while preserving the remarkable capabilities that deep learning has demonstrated, particularly in computer vision tasks essential for intelligent perception and decision-making in unstructured environments.

A primary contribution of this work is the adaptation of computationally intensive neural network architectures, such as Transformers, to make them viable for real-time robotic applications. Through careful architectural optimization, I demonstrate that state-of-the-art models can be practical for resource-constrained platforms without sacrificing high performance. This new direction is exemplified by the development of an efficient transformer for real-time human action recognition, surpassing state-of-the-art models (+2% accuracy) at a fraction of their computational cost (-50% parameters). By crafting efficient variants of

contemporary models, I have opened the door for the deployment of powerful deep learning capabilities on robotic systems.

In addition to these architectural innovations, this dissertation also advances the field of knowledge distillation, a technique for compressing large, complex models into efficient student networks. For instance, I introduce a standardized ensemble knowledge distillation approach that not only reduces model size but also significantly enhances generalization across diverse scenarios, a critical requirement for real-world robotic deployment. This methodology has had a particularly significant impact on semantic crop segmentation, where I demonstrate robust performance across various agricultural settings and species. By effectively transferring knowledge from high-capacity teachers to compact student models, I advanced the state of the art (+2% IoU), paving the way for the democratization of advanced deep learning capabilities in service robotics.

Beyond these contributions to efficient model design and training, this work presents a comprehensive deep learning-driven pipeline for autonomous navigation in crop fields. The system integrates satellite imagery processing, contrastive learning-based waypoint generation, and GPS-agnostic semantic segmentation-based local planning. This end-to-end solution showcases the practical application of the deep learning methodologies I developed, exhibiting robust performance across diverse environmental conditions. By seamlessly integrating various neural network-powered components, I have developed a robust robotic system capable of navigating complex agricultural environments with centimeter-level precision and a latency of a few milliseconds.

The techniques and findings presented in this dissertation have been extensively validated across multiple real-world applications. The demonstrated improvements over existing approaches highlight the significant impact of my work in bridging the gap between state-of-the-art deep learning and the practical constraints of service robotics.

Contents

Introduction	1
Research Motivation and Objectives	1
Main Contributions	4
How to Read this Dissertation	5
I Fundamentals	10
1 Machine Learning	14
1.1 Data	15
1.1.1 Encoding	15
1.1.2 Patterns	19
1.2 Pattern Recognition	21
1.2.1 Tasks	22
1.2.2 Performance Metrics	27
1.2.3 Experience	36
1.3 Models	39
1.3.1 Statistical Models	39
1.3.2 Generative and Discriminative Models	40
1.3.3 Optimization	41
1.3.4 Parametric Models	42

1.3.5	Non-parametric Models	47
2	Deep Learning	56
2.1	Neural Networks	56
2.1.1	Artificial Neuron	58
2.1.2	Multi-layer Perceptron	59
2.1.3	Universal Approximation	61
2.1.4	Activation Function	61
2.2	Optimization	64
2.2.1	Gradient Descent	65
2.2.2	Back-propagation	68
2.2.3	Stochastic Gradient Descent	70
2.2.4	Momentum	72
2.2.5	Adam	73
2.2.6	Optimization Issues	74
2.3	Convolutional Neural Networks	78
2.3.1	Limits of MLPs	79
2.3.2	Convolutional Layer	80
2.3.3	Pooling	85
2.3.4	Convolutional Model	87
2.3.5	Types of Convolution	89
2.4	Regularization and Scaling	91
2.4.1	Overfitting	91
2.4.2	Weight Regularization	93
2.4.3	Early Stopping	93
2.4.4	Data Augmentation	94
2.4.5	Dropout	95

2.4.6	Normalization Layers	96
2.4.7	Residual Connection	98
2.5	Transformers	100
2.5.1	Attention	100
2.5.2	Positional Embedding	105
2.5.3	Transformer Model	107
2.5.4	Additional Tokens	109
2.5.5	Computational Considerations	110
2.5.6	Multimodal Transformer	110
2.6	Advanced Learning Paradigms	111
2.6.1	Transfer Learning	112
2.6.2	Adversarial Training	115
2.6.3	Contrastive Learning	117
3	Efficient and Robust AI	125
3.1	Model Compression	127
3.1.1	Quantization	128
3.1.2	Pruning	129
3.1.3	Knowledge Distillation	131
3.2	Robustness	133
3.2.1	Generalization	133
3.2.2	Fault Tolerance	136
3.3	Hardware	139
3.4	Application Fields	140
3.4.1	Precision Agriculture	140
3.4.2	Home Assistance	142
3.4.3	Remote Sensing	143

II Making AI Efficient	150
4 A Self-Attention Model for Human Action Recognition	153
4.1 Dataset	156
4.2 Methodology	158
4.2.1 Architecture	158
4.3 Experiments	160
4.3.1 Experimental Setting	161
4.3.2 Results	163
4.3.3 Model Introspection	166
4.3.4 Latency	168
4.4 Conclusion	170
5 Super-Resolution at the Edge with Knowledge Distillation	176
5.1 Methodology	181
5.1.1 Network Architecture	182
5.1.2 Training Methodology	182
5.1.3 Knowledge Distillation	184
5.1.4 Model Interpolation	186
5.1.5 Quantization	187
5.2 Experiments	187
5.2.1 Experimental Setting	187
5.2.2 Real-time Performance	189
5.2.3 Super-Resolution Results	191
5.2.4 Ablation Study	195
5.2.5 Image Transmission for Mobile Robotics	196
5.3 Conclusion	199

6 Ultra-wideband Range Error Mitigation at the Edge	207
6.1 Dataset	210
6.2 Methodology	211
6.2.1 Architecture	212
6.2.2 Model Compression	215
6.3 Experiments	216
6.3.1 Experimental Setting	216
6.3.2 Results	218
6.4 Conclusion	224
 III Making AI Robust	 230
 7 Rediscovering The Role of Backbones in Domain Generalization	 233
7.1 The Back-to-Bones Benchmark	238
7.1.1 Backbones	241
7.1.2 Model Introspection	244
7.1.3 Domain Generalization Algorithms	246
7.2 Discussion	247
7.2.1 Are Transformers Better at Generalizing?	247
7.2.2 On Baseline Selection in Previous Works	248
7.3 Conclusion	249
 8 Domain Generalization with Knowledge Distillation	 259
8.1 Methodology	263
8.2 Experiments	265
8.2.1 Dataset	266
8.2.2 Experimental Setting	268

8.2.3 Domain Generalization Results	270
8.2.4 Ablation Study	271
8.3 Conclusion	272
9 An Edge AI Architecture based on Self-supervised Learning	278
9.1 Dataset	280
9.2 Methodology	281
9.2.1 Feature Extractor	283
9.2.2 Application Heads	285
9.2.3 Fault Robustness	286
9.3 Results	288
9.3.1 Cloud Segmentation	289
9.3.2 Flood Segmentation	290
9.3.3 Fire Segmentation	292
9.3.4 Discussion	293
9.3.5 Fault Robustness	295
9.4 Conclusion	298
IV Efficient and Robust AI for Precision Agriculture	301
10 A Pipeline for Autonomous Navigation in Crops	305
10.1 System Overview	308
10.2 Methodology	309
10.2.1 Waypoint Estimation	311
10.2.2 Global Path Planning	313
10.2.3 Segmentation Network	314
10.2.4 Segmentation-based Control	316

10.2.5 Local Path Planning Policy	318
10.2.6 An Alternative Approach: Reinforcement Learning	320
10.3 Experiments	325
10.3.1 Dataset	326
10.3.2 Training	328
10.3.3 Hardware	330
10.3.4 Results	330
10.4 Conclusion	333
Conclusion	339

Introduction

This first chapter serves the purpose of introducing the reader to the thesis work. It explains the primary motivations of the presented research and outlines the main contributions to the state of the art. Finally, the chapter ends with a “vademecum” on how the thesis is meant to be read and introduces some valuable tools to grasp its juiciest morsels.

Research Motivation and Objectives

This thesis work stems from Politecnico di Torino’s interdepartmental center for service robotics, PIC4SeR. The International Organization for Standardization defines a *service robot* as a robot in personal use or professional use that performs useful tasks for humans or equipment¹. These robots are designed to handle repetitive, hazardous, or inconvenient tasks for humans, and they can operate autonomously or semi-autonomously with human intervention. Artificial Intelligence (AI) is fundamental for service robotics because it enables robots to perform complex tasks autonomously, adapt to dynamic environments, and collaborate effectively with humans. Here are the key reasons why AI is indispensable for service robotics:

- **Autonomous Decision-Making:** AI equips service robots with the ability to process real-time data from sensors and make decisions autonomously. That includes navigating unfamiliar environments, avoiding obstacles, and manipulating objects with precision. For example, machine learning algorithms allow robots to learn from past experiences and improve their decision-making capabilities over time.

¹[ISO 8373](#)

- **Enhanced Interaction with Humans:** AI-powered service robots are designed to collaborate and interact with humans seamlessly. Natural language processing (NLP) enables robots to understand and respond to human speech, while computer vision helps them recognize faces, gestures, and emotions. These capabilities are essential for applications in customer service, healthcare assistance, and hospitality.
- **Real-Time Adaptability:** AI enables robots to adapt quickly to changing conditions or unexpected scenarios. For instance, retail robots can modify their navigation paths in crowded spaces using computer vision and edge computing technologies.
- **Precision and Efficiency:** AI enhances the precision of service robots by enabling them to perform tasks such as surgical assistance or quality-control inspections with high accuracy. Machine learning algorithms continually refine robotic operations, enhancing efficiency in industries such as manufacturing, logistics, and agriculture.

Reaching these goals and deploying autonomous agents in unstructured environments seemed easy when the first outbreak of AI occurred. Herbert Simon, a pioneer of AI, famously predicted in 1965: “machines will be capable, within twenty years, of doing any work a man can do” [1]. Researchers assumed that, having almost solved the “hard” problems, the “easy” issues of vision and commonsense reasoning would soon fall into place. But soon, a surprising reality emerged: as Moravec wrote in 1988, “it is comparatively easy to make computers exhibit adult-level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility” [2].

This paradox’s most widely shared explanation stems from evolutionary biology and computational challenges. First, skills like vision, motor control, and social interaction evolved over more than four million years in humans and are deeply hardwired into our biology. These “subconscious” abilities require minimal effort for humans but demand immense computational power for machines. Conversely, abstract reasoning (e.g., language) developed less than 100 thousand years ago in human evolution, making it easier to replicate using al-

gorithms. This consideration aligns with the two-system model proposed by cognitive psychologist Daniel Kahneman in “Thinking, fast and slow” [3].

In addition, there is a computational explanation for the paradox. Logical tasks (e.g., solving equations, playing chess) involve controlled, rule-based systems that computers execute efficiently. On the contrary, sensorimotor tasks (e.g., grasping objects, navigating cluttered spaces) require real-time processing of unpredictable physical environments through multiple sparse modalities (vision, audio, touch). Processing such a massive amount of information strains even advanced systems and hinders representation learning. These and other findings have led experts to conclude that designing systems based on how we think we think is not sustainable in the long run [4].

For this reason, robotics is one of the most interesting and vibrant AI research fields, building toward a future in which autonomous intelligent agents can robustly perceive the environment around them, plan optimal strategies, and act effectively to reach their goals. These robots will be able to support humans in work-related or daily life activities, and they will become safe and reliable “companions”.

Since 2017, PIC4SeR has pushed this research direction at multiple levels: from innovative mechanical architectures to control, localization, navigation, and perception. The research works contained in this thesis focus on perception systems and stem from PIC4SeR’s research agenda. In line with the vocation of Politecnico’s research centers, research ideas often emerge from the close collaboration with several industrial partners. For both small and big companies, PIC4SeR bridges the latest scientific discoveries and real-world problems that arise in various application fields such as space exploration, remote sensing, precision agriculture, and elderly care.

As we will see in more detail in the following chapters, intelligent robotics poses numerous challenges and constraints for AI systems, rendering off-the-shelf high-end solutions ineffective or infeasible. Although machine learning research has yielded outstanding results over the last decade, the gap between scientific benchmarks and deployment in real-world scenarios remains significant. The need for lightweight and efficient models in real-time applications has also grown, promoting the democratization of technologies that are often accessible only to a few large companies due to the high computational power required.

For this reason, my research has focused on making efficient and robust AI possible for a wide range of robotic applications. This thesis summarizes the research I conducted during my Ph.D., investigating whether deep learning models for robotics can be significantly improved in terms of robustness and computational efficiency through architectural design, training procedures, and model compression. With this work, I ultimately contribute towards bridging the gap between large-scale state-of-the-art models and real-time constraints posed by real-world robotic applications.

Main Contributions

The main contributions of the works reported in this thesis can be summarized as follows:

- I propose novel convolutional and transformer-based model architectures for efficient perception and control [5, 6, 7].
- I discover novel training and post-training methods to compress models with minimal performance loss [8, 9, 10].
- I introduce novel methods and evaluations for model generalization in the presence of data distribution shifts and robustness to hardware faults [11, 12, 13, 14].
- I apply such methods to real-world robotic tasks like human action recognition, semantic segmentation, super-resolution, remote sensing, localization, navigation, and control in key application fields [15, 16, 17, 18, 19, 20].
- I evaluate the obtained results in terms of both prediction accuracy and computational complexity.

The results reported in this thesis demonstrate that developing reliable and efficient AI is feasible and warrants further investigation.

All the works present in this thesis were peer-reviewed and published through several renowned Q1 international conferences and journals in the AI field. Besides PIC4SeR, some works have been developed with the contribution of

the Politecnico di Torino Interdepartmental Center SmartData@Polito, partially supported by the Italian government via the NG-UWB project (MIUR PRIN 2017), and by the Italian Space Agency in the framework of the Research Day “Giornate della Ricerca Spaziale” initiative (ASI contract N.2023-23-U.0).

How to Read this Dissertation

This thesis is divided into four main parts. Each part aggregates its contents according to the research problem it addresses or its application field.

Part I addresses the theoretical foundations necessary for correctly comprehending the research works presented in this thesis. Chapter 1 introduces the reader to the discipline of machine learning through data, patterns, statistical parametric and non-parametric models. Chapter 2 leaps into deep learning, introducing neural networks, optimization, regularization, convolutions, transformers, and advanced learning paradigms. Finally, Chapter 3 explores the subfield of efficient AI, encompassing model compression, robustness, and hardware devices, while outlining the main application fields considered in the thesis.

Part II groups three primary studies that propose novel architectures or training methods for efficient inference. Chapter 4 presents a fully self-attention-based model for pose-based human action recognition. Chapter 5 introduces an edge-AI GAN model for image super-resolution trained with knowledge distillation. Finally, Chapter 6 presents a low-power model to enhance indoor localization using ultra-wideband technology.

Part III contains three principal works that evaluate or improve model generalization to data distribution shifts and robustness to hardware faults. Chapter 7 investigates the domain generalization ability of recent convolutional and transformer-based architectures and the impact of several training methods. Chapter 8 proposes a novel domain generalization method based on the knowledge distilled from an ensemble of experts. Finally, Chapter 9 presents a multi-head model for onboard satellite image processing trained with self-supervised learning and optimized for fault tolerance.

Part IV focuses on the specific field of precision agriculture, putting together multiple AI-based solutions to perform autonomous tasks in unstructured environments. In particular, Chapter 10 presents a deep learning-driven algorithmic pipeline for autonomous navigation in row-based crops, seamlessly integrating global and local path planning.

Finally, a **Conclusion** chapter expresses some final considerations and outlines possible future directions stemming from the presented work.

Notes on Notation

Before embarking on our journey through the thesis content, a few considerations regarding style and notation are in order.

I will use **bold** notation for vectors, arrays, and matrices to distinguish them from scalars. Moreover, I will generally refer to them as *tensors*, following common ML terminology associated with the use of frameworks like PyTorch² and TensorFlow³. To express their dimensions, I will use the following notation:

$$\mathbf{X} \sim (b, w, h, 3)$$

\mathbf{X} is a 4-dimensional vector with dimensions b , h , w , and 3. Some dimensions can be pre-specified (e.g., 3), while others can be denoted by variables. Moreover, I will often use Pythonic slicing to index tensors, e.g.,

$$\mathbf{X}[0, a-5 : a+5, ::2, -1] \sim (10, \frac{h}{2})$$

Important definitions will be highlighted in **blue** boxes and side notes in **red** ones. Finally, key general concepts and discursive considerations will be accompanied by a charming little robot icon like the one in Fig. 0.1.

I think this is an appropriate place to acknowledge that the tone of this thesis may become too educational and even informal at times. I hope my reviewers will forgive me, but I was trying to write something that my friends and family

²pytorch.org

³tensorflow.org

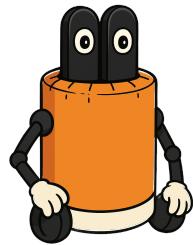


Fig. 0.1 The story of this little robot dates back to when three PhD students decided to create a baby agent (more info [here](#)).

might remotely enjoy reading (and I hope they do, too). With that being said, let's get started!

Bibliography

- [1] A. M. Zador, “A critique of pure learning and what artificial neural networks can learn from animal brains,” *Nature Communications*, vol. 10, no. 1, p. 3770, 2019, ISSN: 2041-1723.
- [2] H. Moravec, *Mind children: The future of robot and human intelligence*. Harvard University Press, 1988.
- [3] D. Kahneman, *Thinking, fast and slow*. Macmillan, 2011.
- [4] R. Sutton, Incomplete Ideas (blog), 2019.
- [5] V. Mazzia, S. Angarano, F. Salvetti, F. Angelini, and M. Chiaberge, “Action transformer: A self-attention model for short-time pose-based human action recognition,” *Pattern Recognition*, vol. 124, p. 108 487, 2022.
- [6] S. Angarano, V. Mazzia, F. Salvetti, G. Fantin, and M. Chiaberge, “Robust ultra-wideband range error mitigation with deep learning at the edge,” *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104 278, 2021.
- [7] F. Salvetti, S. Angarano, M. Martini, S. Cerrato, and M. Chiaberge, “Waypoint generation in row-based crops with deep learning and contrastive clustering,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part VI*, Springer, 2023, pp. 203–218.
- [8] S. Angarano, F. Salvetti, V. Mazzia, G. Fantin, D. Gandini, and M. Chiaberge, “Ultra-low-power range error mitigation for ultra-wideband precise localization,” in *Science and Information Conference*, Springer, 2022, pp. 814–824.
- [9] S. Angarano, F. Salvetti, M. Martini, and M. Chiaberge, “Generative adversarial super-resolution at the edge with knowledge distillation,” *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106 407, 2023.
- [10] A. Navone, M. Martini, S. Angarano, and M. Chiaberge, “Online learning of wheel odometry correction for mobile robots with attention-based neural network,” in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2023, pp. 1–6.

- [11] S. Angarano, M. Martini, F. Salvetti, V. Mazzia, and M. Chiaberge, “Back-to-bones: Rediscovering the role of backbones in domain generalization,” *Pattern Recognition*, vol. 156, p. 110 762, 2024.
- [12] S. Angarano, M. Martini, A. Navone, and M. Chiaberge, “Domain generalization for crop segmentation with standardized ensemble knowledge distillation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5450–5459.
- [13] E. Magli *et al.*, “A multi-service edge-ai architecture based on self-supervised learning,” in *Proc. of 75th International Astronautical Congress*, IAF, 2024, pp. 1–9.
- [14] M. Martini, S. Cerrato, F. Salvetti, S. Angarano, and M. Chiaberge, “Position-agnostic autonomous navigation in vineyards with deep reinforcement learning,” in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2022, pp. 477–484.
- [15] A. Navone, M. Martini, M. Ambrosio, A. Ostuni, S. Angarano, and M. Chiaberge, “Gps-free autonomous navigation in cluttered tree rows with deep semantic segmentation,” *Robotics and Autonomous Systems*, vol. 183, p. 104 854, 2025.
- [16] S. Cerrato *et al.*, “A deep learning driven algorithmic pipeline for autonomous navigation in row-based crops,” *IEEE Access*, 2024.
- [17] A. Navone, F. Romanelli, M. Ambrosio, M. Martini, S. Angarano, and M. Chiaberge, “Lavender autonomous navigation with semantic segmentation at the edge,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2023, pp. 280–291.
- [18] A. Navone, M. Martini, A. Ostuni, S. Angarano, and M. Chiaberge, “Autonomous navigation in rows of trees and high crops with deep semantic segmentation,” in *2023 European Conference on Mobile Robots (ECMR)*, IEEE, 2023, pp. 1–6.
- [19] L. Marchionna, G. Pugliese, M. Martini, S. Angarano, F. Salvetti, and M. Chiaberge, “Deep instance segmentation and visual servoing to play jenga with a cost-effective robotic system,” *Sensors*, vol. 23, no. 2, p. 752, 2023.
- [20] M. Martini, V. Mazzia, S. Angarano, D. Gandini, M. Chiaberge, *et al.*, “Local planners with deep reinforcement learning for indoor autonomous navigation,” in *2021 I-RIM Conference*, I-RIM, 2021, pp. 157–160.
- [21] S. Scardapane, *Alice’s Adventures in a Differentiable Wonderland–Volume I, A Tour of the Land*. arXiv: 2404.17625, 2024.
- [22] C. Canuto and A. Tabacco, *Mathematical Analysis I*. Springer, 2015.
- [23] C. Canuto and A. Tabacco, *Mathematical Analysis II*. Springer, 2015.
- [24] G. Strang, *Introduction to Linear Algebra*. Wellesley Cambridge, 2023.
- [25] S. M. Ross, *Introduction to probability models*. Academic press, 2014.

Part I

Fundamentals

The first part of our journey is devoted to providing the reader with a brief yet solid foundation in the essential elements that underpin this thesis. Even if excellent books already exist covering the topics you will find in the following chapters [21], it is worth doing it for two primary reasons:

- It makes this work standalone and accessible to any engineer or computer scientist without a background in machine learning.
- It allows me (the author) to present these topics the way I best understood them in a (tentatively) conversational and straightforward way, as if I were talking to friends who want to know what I have done in the past few years.

The presentation will only touch on essential topics to provide an overview of my work, avoiding excessive detail on theoretical or archaeological aspects. For this reason, a couple of prerequisites are demanded for the reader to grasp the described concepts correctly. The first is general knowledge of **calculus**, specifically differential and discrete calculus: functions, derivatives, and series theory will often be recalled. The second essential requirement is **linear algebra**, especially vector spaces, gradients, and operators. Finally, the third pillar will be **statistics**, particularly probability theory: distributions, random variables, and conditional probabilities will be used frequently. To the curious reader who wants to dive deeply into these tools, I suggest the books from which I learned them (including the calculus books written by my teacher) [22, 23, 24, 25].

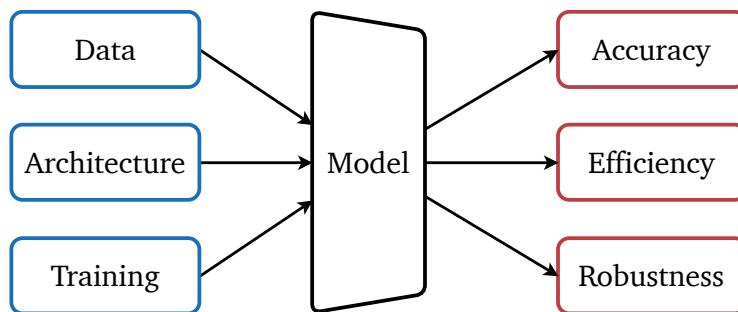


Fig. 0.2 The elements constituting a machine learning model (left) and its properties (right).

Coming to the content of the present part, let us look at the scheme in Fig. 0.2. In the following chapters, we will discover that machine learning models are essentially constituted of three elements: data, architecture, and training. **Data**

is the source of knowledge, and needs to be correctly encoded, curated, and processed to ensure optimal information representation. The **architecture** is the mathematical structure of the model, reflecting the assumptions we make about the task to be solved. Finally, **training** progressively updates the model to fit the data optimally. In the following chapters, we will also understand how to verify that the properties we wish the model to have (on the rightmost part of the figure) can be enforced through design choices and verified.

I will start from basic machine learning concepts (Chapter 1) and then build upon them to enter the world of deep learning (Chapter 2) (a.k.a., differentiable programming). After exploring some advanced concepts, I will introduce EdgeAI (Chapter 3), which addresses the need to bridge the state-of-the-art AI solutions and the constraints of embedded systems. I will also introduce key strategic application fields where my research has been applied.

Chapter 1

Machine Learning

The discipline of Machine Learning emerges at the intersection of data science and artificial intelligence: the former aims to analyze and understand phenomena using data, while the latter tries to create intelligent behaviors in computer systems. Machine Learning subverts the traditional paradigm of rule-based AI by exploiting recurrent patterns in data, enabling behaviors to be automatically defined by an iterative optimization process. Before describing the terms of this process, I will briefly discuss the importance of data, review its most common formats and encodings, and come to a definition of pattern recognition.

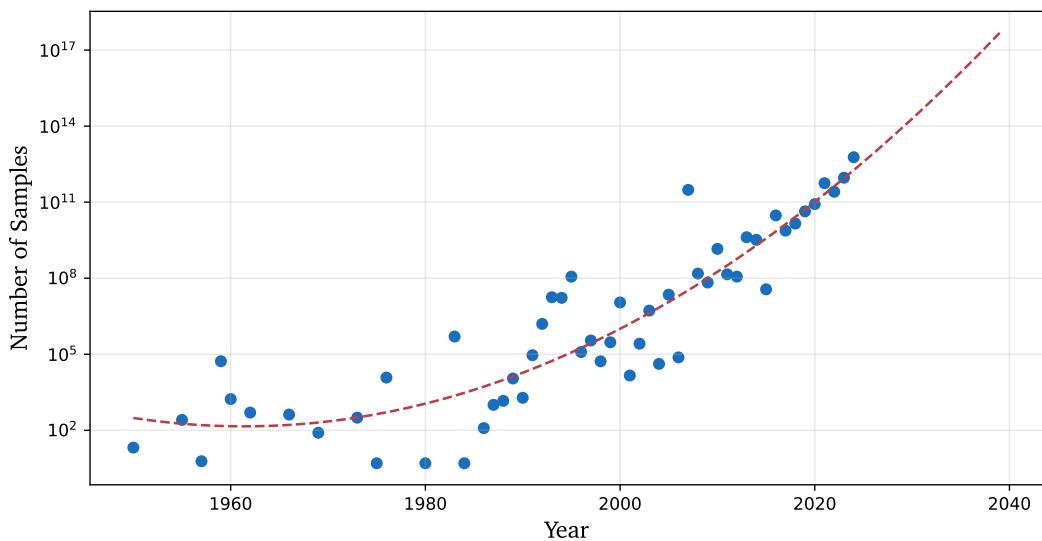


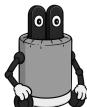
Fig. 1.1 Exponential growth of datapoints used to train AI systems and future trend [1].

1.1 Data

The production of signs to communicate meaning is a fundamental trait of humankind and has existed since its early evolution. A sign is created using a specific medium to suggest an interpretation, which is ultimately left to the receiver of the sign. Data can be considered a raw and objective representation of a phenomenon before any interpretation or analysis. It is constituted by a set of symbols following a specific encoding determined by the device supporting or storing it (a vinyl record, a book, or a person's hippocampus). I will focus on digital data, e.g., everything that can be stored in sequences of zeros and ones and processed by computers. We will see that the exponential growth of digital data production (and consumption) has been the first factor to make the AI revolution possible (Fig. 1.1).

Information is the sum of data and its interpretation, which makes the intrinsic content of meaning explicit and understandable. This interpretative step (which we also call decoding) should not be overlooked or considered naive. We will see that information is, in fact, hidden in the meanders of data, even for the simpler formats.

The information media humans perceive through senses and are familiar with (like images and sounds) are not straightforward for machines due to their analog nature and intrinsic high dimensionality.



To better understand this idea, I will now delve into the concept of encoding, which enables machines to process data from several sources.

1.1.1 Encoding

Encoding standards emerge from the need for shared and exactly reproducible processes to store data in computers. Computers are digital devices that require every piece of information to be coded in binary digits (or bits). Such a constraint is not problematic for discrete information media, such as mathematics and language. Everything that can be written on a paper sheet or put in a table can be easily converted into its digital representation through simple standards (e.g., the ASCII table [2] for letters and rounded binary representation for numbers).

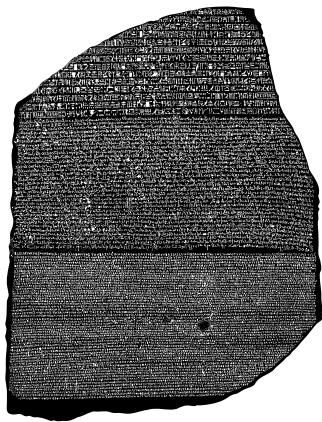


Fig. 1.2 The Rosetta stone. Being able to extract information and meaning from data is not straightforward if the receiver lacks the instruments to decipher the data encoding.

Other, more complex kinds of data (often referred to as non-tabular) require a deeper analysis.

Images can be encoded as matrices of pixels (Fig. 1.3). Pixels are the smallest addressable elements, each assigned a color. As colors are usually encoded according to the RGB standard, each pixel contains three values (called channels C) representing the intensities of red, green, and blue. Conventionally, these values are expressed as unsigned integers on eight bits (or a byte) and have values in the $[0, 255]$ range, allowing the creation of more than eight million different colors. Grayscale images are obtained by considering a single channel expressing the pixel's intensity, usually called *luminance*. *Resolution*, defined as the product of the number of columns in the matrix (called *width W*) and the number of rows (called *height H*), quantifies the level of detail of the image: the higher the resolution, the more pixels can describe the same region. In conclusion, we can view an image as a three-dimensional array of shape (W, H, C) , occupying $W \times H \times C$ bytes of memory.

Videos can be conceptualized as an ordered sequence of images. A video will then be processed as a four-dimensional array of shape (T, W, H, C) , where T is the number of time instants (or *frames*). Along with color discretization (through RGB encoding) and space discretization (through pixel grids), video encoding introduces time discretization. Indeed, the continuous flow of time must be



Fig. 1.3 A simple example of image encoding. The image has been downsampled and grayscaled for better clarity.

quantized according to a defined frame rate, which depends on the time Δt intercurring between successive video frames. The frame rate $1/\Delta t$ is measured in frames per second (*fps*) and quantifies the fluidity of the video stream.

Time Series are collections of measurements of one or more processes (e.g., stock prices). We can represent a time series as a matrix $X \sim (T, C)$, where T is the length of the time series, and $X_i \sim (C)$ are the C measurements taken at time T (e.g., sensors from an EEG scan). Time is discretized in the same manner as in videos. **Audio** signals are a specific type of time-series data. A sound is produced by continuous variations in air pressure, and we can measure its intensity over time by tracking the vibration generated by these variations. Volume, pitch, and timbre are nothing more than properties of this temporal sequence. To turn a sound into a digital signal, we measure its intensity at fixed time intervals. Each measurement is called a sample, and the number of samples per second (called the sample rate) is measured in Hertz (*Hz*). For instance, a typical sampling rate is 44.1 *kHz* (44,100 samples per second). This way, an audio waveform is converted into a one-dimensional array of T intensities. Another common practice in audio processing is generating spectrograms to extract information from the frequency-domain analysis of the signal Fig. 1.4. The spectrogram of a signal equals its Fourier transform over time. Suppose we plot time on the x-axis and frequency on the y-axis. In that case, the audio array is transformed into a two-dimensional array of shape (T, F) , where F is the number of discrete frequency values considered in the Fourier transform. Spectrograms are often used in deep learning to adapt models designed for images to audio, as a spectrogram can be viewed as a single-channel image.

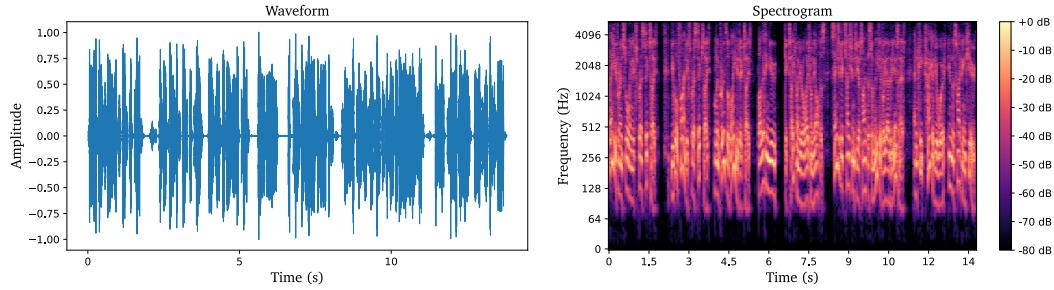


Fig. 1.4 Example of an audio waveform (left) and its corresponding spectrogram (right).

Text can be converted into a numerical sequence simply through *one-hot* encoding. Imagine we have a dictionary containing K words. Each word in this dictionary has a unique position given in alphabetical order. We can assign the i^{th} word a vector with K zeros and then set the i^{th} position to 1. This way, a sentence of n words is transformed into a $N \times K$ matrix with rows corresponding to the words. Besides being very inefficient computationally (the dictionary size K can be huge), this encoding does not consider semantic meaning. For this reason, *word embeddings* are commonly preferred. They are implicit representations of words across several hidden features or dimensions. These features capture patterns such as semantic relationships, contextual usage, or other language characteristics. This representation is not constructed by hand, but rather, it is inferred by a large corpus of text to give more similar representations to words the more they co-occur. For example, words like “king” and “queen” would naturally cluster together, while “apple” and “orange” might form another cluster, far away from unrelated words like “car” or “bus”. With this simple method, word embeddings create an efficient and nuanced representation of complex dependencies [3]. An alternative, more modern strategy involves splitting text sequences into sub-words, leveraging the so-called byte-pair encoding [4]. This approach reduces redundancies by avoiding using multiple entries for different forms of the same concept (e.g., play, plays, playing). Instead, a sentence like

I like playing drums.

is tokenized to keep subwords conveying meaning separated from those encapsulating modifiers (e.g., **-ing** and **-s**). Special dictionary entries can be assigned to

punctuation, “start-of-sentence”, and “end-of-sentence”. Modern tokenizers, like *tiktoken*¹ contain roughly 100k different subwords.

Non-sequential Data has no explicit spatial correlation between different elements, contrary to all modalities described so far. For example, a vinyl collection is a group of music records, each with its related properties: the name, the singer, its length, its genre, and so on. Although one could order them based on some criterion, they have no inherent sequential relationship. These data types are stored in tables or equivalent complex data structures, where no specific order is implied, and are also commonly referred to as *tabular data* (Fig. 1.5).

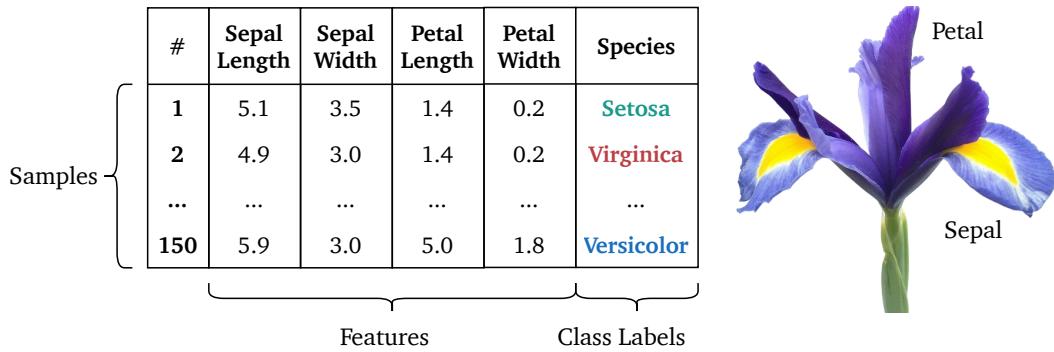


Fig. 1.5 An example of non-sequential data: the Iris dataset [5].

In conclusion, encoding allows machines to store and manipulate many different data types. However, we may want to take it a step further and make machines understand and extract information from data automatically. To do so, we must exploit some hidden regularities, called *patterns*.

1.1.2 Patterns

In 1985, physicist Satoshi Watanabe defined a pattern as “*the opposite of chaos; it is an entity, vaguely defined, that could be given a name*”. A pattern, first of all, is something that is not chaotic; it is a feature of reality that carries meaning or conveys a piece of information. We are confronted with patterns all the time in our daily lives. If we were walking down a street and noticed someone looking up, we would probably ignore them and keep walking. But if it were instead five

¹<https://github.com/openai/tiktoken>.

or six people looking in the same direction, we would probably give them more attention and look up ourselves. To our mind, a single occurrence of an event is less likely to be caused by some underlying meaning than multiple repetitions of the same event. Although correlation does not imply causation, we find ourselves acting based on observed patterns all the time. The emergence of patterns has also been fundamental for the development of physics. For example, Tycho Brahe's astronomical observations in the 16th century enabled Johannes Kepler to discover the empirical laws of planetary motion, which in turn helped Newton develop the foundation of mechanics. Similarly, the discovery of regularities in atomic spectra played a crucial role in the development and verification of quantum physics in the early twentieth century [6].

Within the scope of data encoding defined in the previous subsection, patterns refer to repeated numerical trends, regularities, and correlations in data samples. Take, for example, the popular digit classification dataset MNIST (Fig. 1.6).

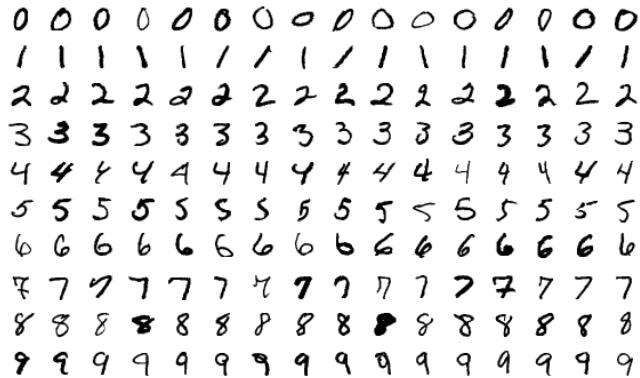


Fig. 1.6 The MNIST dataset [7].

We can immediately highlight some recurring features of instances of the same digit: for example, zeros all have a round and closed shape, while ones are straight and vertical. However, these patterns are found alongside other features that are not directly related to the represented number. These different patterns may depend on specific handwriting style: for example, some ways of writing a four may present the same patterns we find in sevens. These regularities, known as *spurious correlations*, pose a problem as they can lead machines to rely on misleading information. I will analyze this phenomenon more deeply in the following chapters.

Watanabe also notes that patterns can be given a name, which allows them to be recognized and associated with a specific meaning. Hence, the next step would be to enable computers to identify patterns in data and utilize them to extract semantic information. The discipline of *pattern recognition* serves this specific purpose.

1.2 Pattern Recognition

The field of pattern recognition aims to automatically discover regularities in data through the use of computer *algorithms* (or *models*). These regularities are then leveraged to take actions or solve tasks, such as categorizing the data. Continuing the previous section's example, we could be interested in designing a program that automatically detects patterns in handwritten digits and uses them to assign a class [0-9] to each one. One naive approach to tackling this could be to manually write a set of rules that the algorithm should follow to identify digits correctly. This approach, similar to standard code writing, is referred to as the *rule-based* approach and can be schematized as shown in Fig. 1.7.

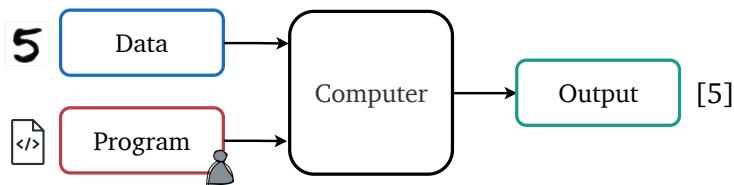


Fig. 1.7 The rule-based approach.

However, such an approach would lead to a proliferation of rules and exceptions, requiring considerable effort from the designer. Moreover, even a simple task like recognizing digits would yield poor results in practice. Instead, another approach involves collecting a large set of labeled examples and designing an algorithm that can process these examples and automatically adapt to them. We refer to this approach as *data-driven*, as rules are inferred from data without direct human intervention (Fig. 1.8).

This procedure implies an iterative process in which the internal functioning of the algorithm is progressively tuned to better fit the pool of provided examples (called *dataset*). The model's ability to recognize digits is quantified through a

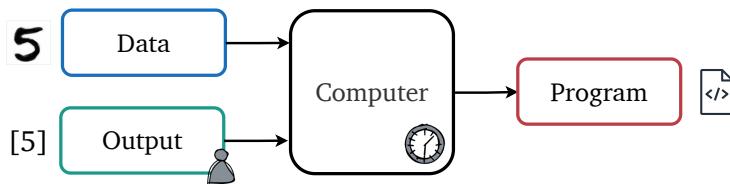


Fig. 1.8 The data-driven approach.

predefined metric, such as the ratio of correct predictions. The more data is fed to the model, the more the metric is expected to improve. This phenomenon resembles the human ability to learn progressively from experience, and thus it has been named **machine learning**. I can now introduce a formal definition coined by Mitchell [8]:

Definition 1.1 (Machine Learning)

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .

Now, before describing the mechanism that allows a machine-learning algorithm to improve over experience, I will explain the role of the three building blocks: T (the task), P (the performance measure), and E (the experience).

1.2.1 Tasks

Supervised Learning

Our toy example uses the MNIST dataset to create a machine learning model that recognizes handwritten digits. That is a **classification** task, as the model's goal is to predict the correct label (or class) for a given sample from a limited list of possible values (in this case, digits from 0 to 9). In other words, we could say that, in classification, y is a discrete or *categorical* value. Such algorithms are typically designed to assign a probability to each of the available classes, so that the class with the highest probability value is predicted. On the contrary, we talk about **regression** when y is not bound to a set of discrete classes but is a continuous value (e.g., a number). *Forecasting* is a typical regressive task: be it the temperature in a city or the price of a stock, the model aims at approximating

as closely as possible a continuous function $y = f(x)$ (where typically $y \in \mathbb{R}$) to predict future values in a time series. Due to their intrinsic differences, classification and regression imply different design choices and metrics. Nonetheless, all the examples we have considered so far lie in the category of supervised learning, as they benefit from the presence of labeled data samples to learn the joint probability distribution $p(x, y)$.

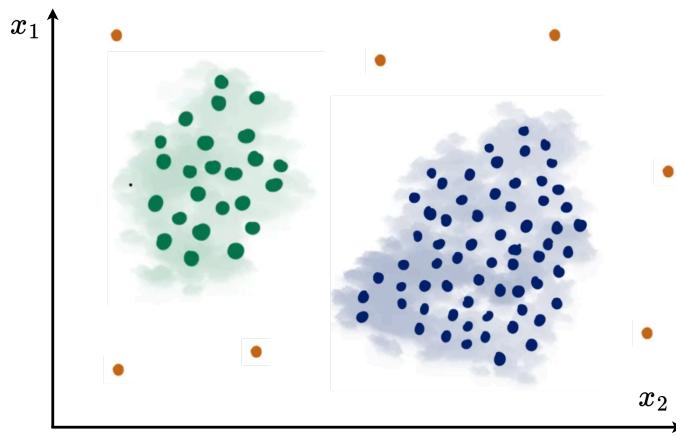


Fig. 1.9 Clustering algorithms group samples based on spatial arrangement and help detect anomalies in the dataset (also called outliers and depicted in orange).

Unsupervised Learning

When labels are unavailable, the task can be framed as an unsupervised learning problem. In this case, the dataset only contains unlabeled samples:

$$\mathcal{S}_n = \{x_i\}_{i=1}^n \quad (1.1)$$

In unsupervised learning, the objective is to extract patterns from the data to infer some hidden property of the sample distribution $p(x)$ without human labeling. The simplest form of unsupervised learning is **clustering**, which aims to group samples based on their arrangement in the data space. It stems from the intuition that samples in the same region share common features. The way we define the similarity between samples leads to different clustering approaches. The following sections will outline the main clustering algorithms and highlight their key characteristics. Clustering can be helpful in practical applications, such as profiling, which involves grouping users based on their usage of a specific service.

It can also be beneficial in anomaly detection, as samples lying in isolated portions of the data space may be highlighted by clustering (Fig. 1.9).



Fig. 1.10 MNIST visualization through t-SNE dimensionality reduction [9]. This visualization technique is explained in Section 1.3.5.

As we have already mentioned, however, for some data types, the number of dimensions increases rapidly and can pose critical problems. One of these is the difficulty of visualizing the diversity and similarities of high-dimensional samples on a 2D plot. **dimensionality reduction** algorithms provide a simple way of optimally shrinking the number of dimensions of data while maintaining the majority of its information. For example, Fig. 1.10 shows the result of applying dimensionality reduction to the MNIST dataset.

The reduction helps with data visualization, immediately showing which digits are more similar (e.g., nines and fours). Dimensionality reduction is also beneficial for algorithms that perform best on low-dimensional data. In those cases, it can be used as a preprocessing stage to ease learning. I will return to this concept when we address the *curse of dimensionality*.

Self-supervised Learning

Sometimes, the dataset we want to learn from is so extensive that it would be impossible to label each sample. The growing production of new data over the last decade has led to the collection of increasingly larger datasets, with the (correct) intuition that more data enables better machine learning systems. That is particularly true for broad and complex tasks such as image recognition and

natural language processing (NLP). In these cases, however, totally unsupervised learning does not yield sufficiently good results due to the absence of guidance from labels. A possible solution to the lack of supervision is to artificially create a pretext task that allows for the automated generation of labels through some manipulation. This paradigm is called self-supervised learning (SSL). For example, suppose we want a machine learning algorithm to learn to write English. Imagining a dataset suited for this task is difficult, let alone creating and labeling it. However, access to a large corpus of unlabeled text is relatively easy, so we can aim to predict how a small piece of text is likely to continue by masking out words from it [10]. Consider, for example, the following task:

A QUICK **[MASK]** FOX JUMPS OVER THE **[MASK]** DOG



A QUICK **BROWN** FOX JUMPS OVER THE **LAZY** DOG

In this case, text completion is a pretext task for natural language understanding. Some random sentence pieces are masked, and the model is asked to predict the missing portions.

Using sufficient data samples, this pretext task indirectly forces the algorithm to extract patterns from the structure of natural language, ultimately inferring grammar and syntax rules, as well as the semantic meaning of words. This knowledge can then be re-purposed to solve more specific problems, called *downstream tasks*, for which a labeled dataset is typically orders of magnitude smaller than the SSL dataset. In the last couple of years, the advancement in computational hardware and the realization that machine learning models can represent data surprisingly efficiently when pre-trained in a self-supervised way profoundly impacted the diffusion of these algorithms to the mass public².

Reinforcement Learning

Reinforcement Learning is based on the foundational stone of many cognitive theories, which suggest that humans, and especially children, learn from their interactions with the environment around them. A child progresses in learning a

²“ChatGPT Is The Fastest Growing App In The History Of Web Applications”, Forbes (2023).

skill by continually trying and making mistakes, be it walking, rolling spaghetti, or saying “mum”. Abstracting this process, we get a framework in which an *agent* is placed in an *environment* to learn how to solve a *task*. The *agent* chooses the action to perform based on its *state* and the state of the environment around it. The estimation of the state is achieved through some form of perception. After the action is taken, the environment responds, informing the agent that the state has changed and providing an evaluation of its action in the form of a numerical *reward*. Iterating this process, the agent, whose objective is to obtain higher and higher rewards, will learn the causal relation between action and reward from experience, just like a dog learns to fetch a stick (Fig. 1.11).

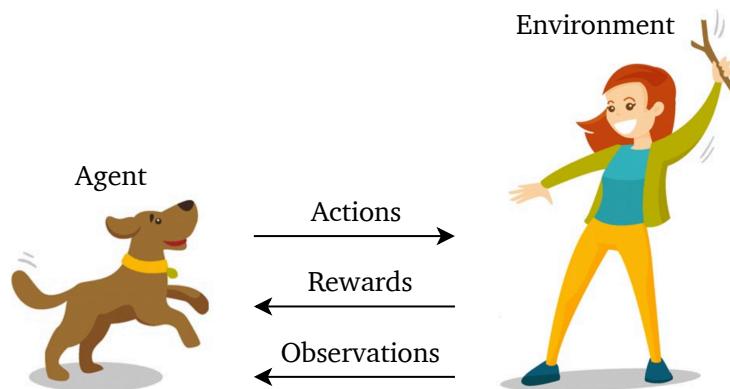


Fig. 1.11 Exemplification of the reinforcement learning paradigm³.

Hence, reinforcement learning maps states to actions to maximize a reward that progressively nudges it toward the optimal behavior. The learning happens through a closed-loop formalized as a *Markov decision process* [11]. Besides the agent and the environment, a reinforcement learning algorithm essentially consists of:

- a *policy* that selects the optimal action given the current state.
- a *value function* that maps each state to the total reward the agent is expected to gain in the future according to the current policy.

All these elements help the agent find an equilibrium between insisting on known behaviors (*exploitation*) and trying different strategies (*exploration*) to further

³mathworks.com

increase the reward. This long-term versus short-term reward trade-off has been applied successfully to various problems, including robot control [12], protein synthesis [13], and games [14].

Now that we have outlined the main tasks and learning paradigms, we can define the performance measures to evaluate the correctness of the process.

1.2.2 Performance Metrics

From the definition of machine learning I used at the beginning of this section, we saw that performance measures (or *metrics*) help verify that an algorithm improves its performance with experience. They give a concise representation (usually in the form of a scalar value) that is not used in an absolute way but rather as a relative index to compare different algorithms (or to the same algorithm at a different point in the learning process). We can say that:

Definition 1.2 (Metric)

Given a label y and a predicted value $\hat{y} = f(x)$ from a model f , a metric $m(y, \hat{y}) \in \mathbb{R}$ is a scalar function whose value correlates with the performance of the model.

The correlation can be positive if the metric expresses the “goodness” of the model or negative if it represents an error quantification, e.g., how much it deviates from the desired behavior.

Metrics are also helpful in finding bottlenecks, bugs, and undesired phenomena in the system. Indeed, in general, different measures are suited for different purposes, as typically a single value is not sufficient to capture all the nuances and pitfalls that the model may encounter. For this reason, many alternatives exist even for the same task. I will briefly address the fundamental ones and leave the others to the specific application scopes of the following chapters.

Classification

In classification, the easiest way to quantify a model's ability to predict the correct label is to compute the ratio between the number of correct predictions and the total. This metric is called **accuracy** and can be defined as:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Predictions}} \quad (1.2)$$

Accuracy is usually reported as a percentage, as it can only assume values in $[0,1]$. Although accuracy is the most used metric for both binary and multi-class classification, it does not capture aspects that are critical in some applications.

Accuracy is not exhaustive when classes are *unbalanced*, or some classes are more relevant than others. Metrics like **precision** and **recall** are more suited for such cases. First, we need to define *true positives* (TP) as the correctly classified positive samples and *false negatives* (FN) as the ones predicted as negatives. Similarly, we will refer to *false positives* (FP) as negative samples classified as positives and *true negatives* (TN) as the correctly predicted ones. Hence, we obtain Table 1.1.

Confusion Matrix		Predicted	
		Negative	Positive
Label	Negative	TN	FP
	Positive	FN	TP

Table 1.1 Confusion matrix for general binary classification.

This table is called the *confusion matrix*, as it highlights the mutual relations between classes that the model may confuse. I can now define precision and recall:

$$\begin{aligned} \text{Precision} &= \frac{TP}{TP + FP} \\ \text{Recall} &= \frac{TP}{TP + FN} \end{aligned} \quad (1.3)$$

Notably, precision highlights false positives, for example, situations in which the model is triggered unnecessarily (false alarms). Recall, in spite, highlights false negatives, e.g., cases in which the model should be triggered and is not (misses). The *F1 score* can be used to unify the two metrics, assigning equal weight to each.

$$F1 = \frac{Precision + Recall}{2} \quad (1.4)$$

Regression

In regression tasks, the machine learning model predicts a continuous output value that, during the training phase, is compared to a label that can assume any real value. In this case, the more the output is numerically close to the label, the better the model will perform the task. For this reason, regression metrics are based on distance. The most straightforward distance formulation is the *absolute error*:

$$e = |\hat{y} - y| \quad (1.5)$$

where y is the label, \hat{y} is the prediction, $|\cdot|$ is the absolute value operator. Considering a set of n samples, we can compute the **mean absolute error** (MAE) as

$$MAE = \frac{1}{n} \sum_i^n |\hat{y} - y| \quad (1.6)$$

Similarly, an alternative solution is the **mean squared error** (MSE), defined as

$$MSE = \frac{1}{n} \sum_i^n \|\hat{y} - y\|_2^2 \quad (1.7)$$

where $\|\cdot\|_2$ is the Euclidean norm operator. MSE gives more weight to large errors than smaller ones, so it is more commonly used than MAE.

Cosine similarity is an important metric when we do not want to be biased by the magnitude of vectors. That is often the case for high-dimensional data, where Euclidean distance increases drastically with the number of dimensions. In such cases, the direction of vectors takes on more importance than their magnitude. For this reason, we define the cosine similarity of two tensors as the cosine of the angle between them. Using the rules of trigonometry, we can write

$$S_c(\mathbf{v}, \mathbf{w}) = \cos(\theta_{\mathbf{v}, \mathbf{w}}) = \frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{v}\| \cdot \|\mathbf{w}\|} \in [-1, 1] \quad (1.8)$$

Vectors with a small Euclidean distance from one another are located in the same region of a vector space. In contrast, vectors with a high cosine similarity are located in the same general direction from the origin. Fig. 1.12. In the particular case in which \mathbf{v} and \mathbf{w} are orthogonal, we have $S_c = 0$. Compared to *MSE*, one advantage of cosine similarity is its low complexity, especially for sparse vectors: only the non-zero coordinates need to be considered.

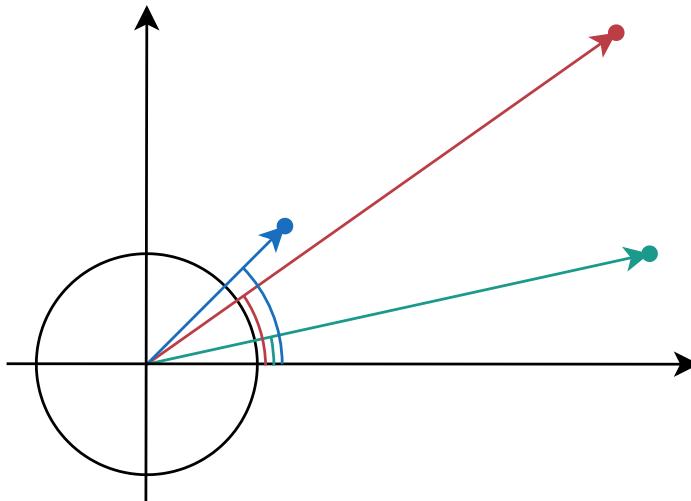


Fig. 1.12 While Euclidean distance is akin to measuring with rulers, cosine similarity is equivalent to measuring with goniometers. In this case, the **blue** sample is more similar to the **red** one than the **green** one⁴.

⁴baeldung.com

Dense Tasks

Several tasks, especially in computer vision, escape the simple distinction between classification and regression due to their complexity. For this reason, we hereby call “dense” the tasks that output multiple classification or regression values depending on the input content. For example, detection models output a variable list of bounding boxes depending on the number of objects in the image. For some of these tasks, models predict pixel-level values. It is the case of segmentation, super-resolution, depth estimation, and many others.

Intersection-over-Union (IoU) is a crucial metric for assessing segmentation and detection models. These computer vision tasks focus on recognizing where an object or class is located in the image. Detection predicts a bounding box around the object, while segmentation does it pixel-wise (Fig. 1.13). It is based on *Jaccard’s Index*, which quantifies how well a model can distinguish objects from their backgrounds in an image. It’s used in numerous computer vision applications, such as autonomous vehicles, security systems, and medical imaging.

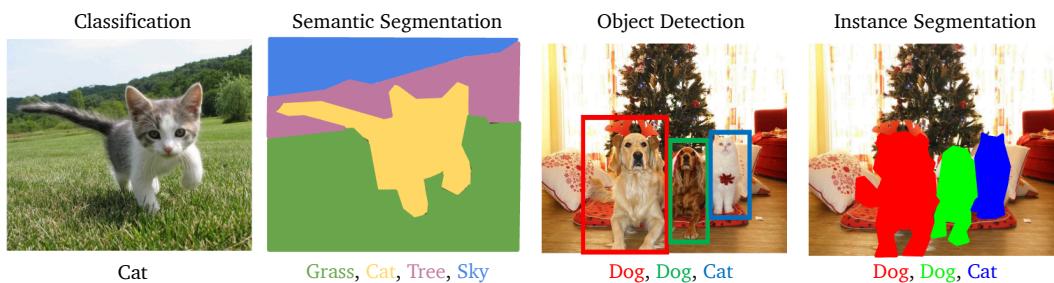


Fig. 1.13 Comparison between image classification, semantic segmentation, object detection, and instance segmentation. Note that classification has no spatial extent, and semantic segmentation does not recognize objects, only pixels. Instead, Object detection and instance segmentation detect multiple objects.

Given two bounding boxes (the actual label and the predicted one), IoU is the ratio of the intersection of the two boxes’ areas to their combined areas (Fig. 1.14). An IoU score of 1 indicates a perfect match between the projected box and the ground truth box, whereas a score of 0 means no overlapping between the boxes. The same logic applies to segmentation, while in this case, we have irregularly shaped areas to process.

I have defined some elementary metrics that help evaluate the performance of a machine learning algorithm after it has been trained. However, machine learning

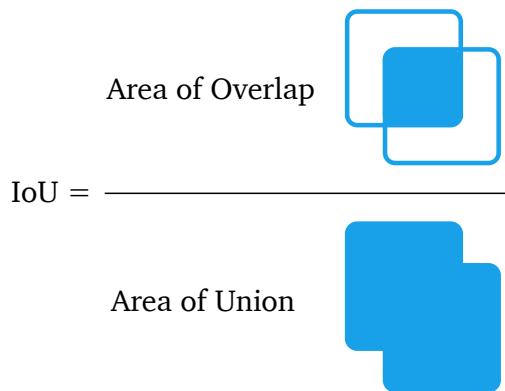


Fig. 1.14 Visual explanation of IoU.

models also actively use a specific set of metrics as a fundamental ingredient in their training process. These metrics are referred to as *loss functions* and require additional mathematical properties.

Loss Function

In the following subsection, I will dive into the training mechanism of machine learning algorithms and frame it as a statistical optimization process. This idea basically reduces learning a task to minimizing a predefined *loss function* that expresses how much the model deviates from the desired behavior. That implies two properties distinguishing a loss function l from a generic metric:

- l must be negatively correlated to the performance of the algorithm so that $l(y, \hat{y}_1) < l(y, \hat{y}_2)$ implies that the prediction \hat{y}_1 is better than the prediction \hat{y}_2 .
- l must be **differentiable** so that the gradient of l is always well-defined.

We will gain a better understanding of these constraints later. For now, let us define some fundamental loss functions for the most common tasks.

Classification The classification metrics we have seen so far all express how well the algorithm performs. Moreover, they are not differentiable as they come from simply counting correct predictions. If the algorithm predicts a probability value for each class, we can define a differentiable loss that quantifies the distance

between the predicted probability distribution and the ideal correct one. Using the concept of entropy derived from Shannon's information theory [15], we can measure the distance between two distributions using the *Kullback-Leibler divergence*:

$$\begin{aligned} D_{KL}(p, q) &= \sum_{c \in C} p(c) \cdot \log\left(\frac{p(c)}{q(c)}\right) \\ &= \sum_{c \in C} p(c) \cdot \log(p(c)) - p(c) \cdot \log(q(c)) \end{aligned} \quad (1.9)$$

where c is a class in the set C , p is the target probability distribution and q is the predicted one. Since $p(c)$ does not depend on the model but comes from the dataset, the first term can be removed in the scope of loss minimization, obtaining

$$D_{KL}(p, q) \simeq - \sum_{c \in C} p(c) \cdot \log(q(c)) = H(p, q) \quad (1.10)$$

We call $H(p, q)$ the **cross-entropy**. While other classification losses have been proposed [16] [17], cross-entropy is the most used.

Regression The metrics I defined for regression are already error measurements, satisfying the first mathematical condition of loss functions. However, while *squared error* is differentiable and can be directly used as a loss, *absolute error* is not. If we want to limit the loss value even for big errors, we can use functions with a linear asymptotic behavior that are differentiable around zero. Huber loss is an example, but many variants exist [18]. **Huber loss** (Fig. 1.15) is defined as a piecewise function:

$$L_\delta(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta \cdot (|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases} \quad (1.11)$$

where $a = y_p - y$ is the prediction error and δ is a tunable parameter. The higher δ , the steeper the asymptotes.

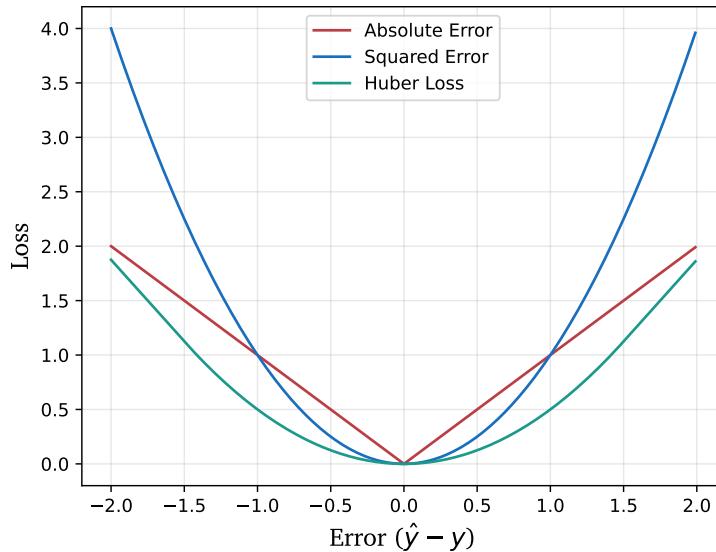


Fig. 1.15 Visualization of squared error (MSE), absolute error (MAE), and Huber Loss.

Validation

Since models become familiar with training samples during optimization, quantifying their performance using the same metrics is prone to error. For example, the model may minimize training loss by “cheating” and memorizing samples instead of extracting their underlying patterns (this pitfall, called *overfitting*, will be further detailed in Section 2.4.1). To distinguish “authentic” learning from loss hacking, we need a *validation* strategy.

The most straightforward approach to do so is to have access to another dataset, separate from the training dataset, which we call the *test set*, and use it to compare the ability of different models to solve a task through a specific metric. The test set, however, is meant for the final evaluation of a model and should be considered a real-world deployment scenario. After the model is tested, we cannot go back and modify it since doing so would invalidate its scope (that would be “cheating”, too). Instead, a common practice is to extract a test set proxy, called *validation set*, that we exclude from training and use solely for intermediate evaluations. This way, we always have a comparison term as we try different configurations and

hyperparameters. Validation can also be used to monitor training by evaluating the model at the end of each epoch.

Validation is crucial for comparing different models and identifying potential undesired phenomena in the optimization process. However, validation results highly depend on how the validation set is sampled from the training set. Especially for small datasets, validation data may be either too similar to training samples or too different. For instance, a significant dataset feature may be excluded from training, making it impossible for the model to learn. In such cases, **k-fold cross-validation** is often applied. It consists of splitting the dataset into k random parts and iteratively using one of them as a validation set. Once the training has been repeated k times, the metrics are averaged across the runs, leading to a more reliable estimate of the model's performance (Fig. 1.16).

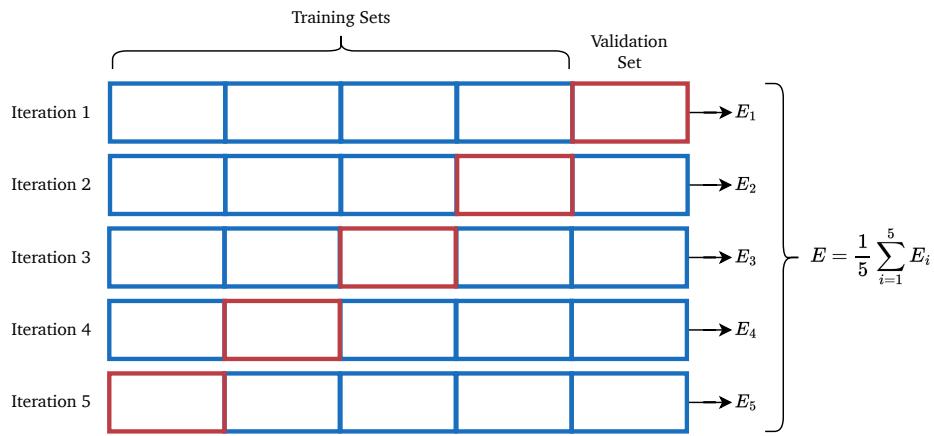


Fig. 1.16 Example of k -fold cross-validation for $k = 5$.

In general, validation is fundamental for optimal *model selection* since we cannot know precisely how well a predictive model will work in practice, i.e., the *true risk*. Validation estimates the true risk with a better approximation, called *empirical risk* [19]. The gap between training and validation risk estimates is typically referred to as *generalization error*, representing the model's ability to generalize to distributions different from the training dataset. In conclusion, I have defined how machine learning algorithms are evaluated by quantifying their ability to solve a task. Now, it is time to understand how experience can increase that ability.

1.2.3 Experience

Experience enables a machine learning algorithm to extract knowledge (e.g., relevant patterns) from data through a process known as training. During training, the algorithm progressively learns to approximate a desired behavior through some optimization. We will explore this concept further in the next section, but first, let us rigorously define how knowledge is provided to the model to enforce the desired behavior.

Dataset

In the very first section of this chapter, I introduced data as a raw representation of a phenomenon. I also stated that hard-coding a set of rules is highly time-consuming and scarcely effective for some pattern recognition tasks. In those same cases, however, it is relatively easy to gather examples of the desired behavior in the form of input-output couples. I will call the two elements of each couple *sample* and *label* or, in mathematical notation, x and y . The full dataset can then be formalized as

$$\mathcal{S}_n = (x_i, y_i)_{i=1}^n \quad (1.12)$$

From a statistical point of view, the samples are considered to be drawn from the same joint probability distribution $p(x, y)$. Intuitively, p is the distribution of all possible samples relevant to the task across all the possible variants. For example, in the case of handwritten digit classification, p will contain all the possible ways of writing any digit, varying line width, rotation, and calligraphy. Due to its definition, $p(x, y)$ is unknown and ultimately unknowable; however, it can be better approximated as n approaches infinity. Another requirement to be enforced on the dataset \mathcal{S}_n is that samples are independent, which means there is no hidden structure or correlation between any given pair of samples (x_i, x_j) . Combining the two assumptions, we conclude that the data samples are *independent* and *identically distributed*. In conclusion, we can already sense that learning from a dataset is an *inductive* process in which we aim at implicitly estimating a general target distribution $p(x, y)$ from which the dataset is sampled.

The better \mathcal{S}_n covers $p(x, y)$, the better our machine learning model will solve the task. I will delve deeper into this concept by addressing the *generalization* problem.

Data Space

At this point, I must introduce another important concept to help us visualize encoded data samples and wrap our heads around regularities and patterns: the *data space*. I defined a dataset as a set of n-dimensional arrays, where n depends on the type and shape of the data being represented. Drawing from our knowledge of *linear algebra*, we call the data space the n-dimensional vectorial space to which our data belongs. For example, the samples in the MNIST dataset are images with a resolution of 28×28 and a single channel. So, we can represent each sample in a vectorial space $\mathbb{R}^{(28,28)} = \mathbb{R}^{784}$. Once we plot all the samples in the data space, we can utilize geometric concepts such as distance and similarity metrics to highlight their correlations and extract relevant patterns. However, the number of dimensions skyrockets for specific data types (e.g., high-resolution images).

Preprocessing

As we will see in later sections, pattern recognition systems work best starting from correctly prepared data. For example, we generally want our data to be in adequate quantity, complete, coherent, and highly informative (avoiding irrelevant data or excessive noise). Moreover, statistical approaches are typically based on certain assumptions about the input samples, such as the *i.i.d.* property presented in the previous subsection. Several *preprocessing* techniques exist to polish and prepare datasets for machine learning models.

Selection is typically the first step, involving the choice of the relevant data portion to use. The choice is also based on the presence of corrupted or missing values and outliers, i.e., anomalies in the data distribution that may hinder the correct extraction of patterns.

Normalization is put in action after data selection to transform the statistical distribution of samples and align it with the chosen pattern recognition method. Most models require features to be in similar ranges to avoid one prevailing over the others. For this reason, a common practice is to apply **standardization**, i.e., rescale and center data distributions to resemble a Gaussian distribution:

$$\mathcal{G}(\mu = 0, \sigma^2 = 1) \quad (1.13)$$

where μ is the mean value of the distribution and σ^2 is its variance.

$$\hat{x} = \frac{x - \mu_x}{\sigma_x} \quad (1.14)$$

An alternative approach, called **min-max normalization**, is to rescale the data to be bounded between 0 and 1.

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \quad (1.15)$$

Other solutions exist that aim to reduce the negative impact of outliers. For example, the so-called **robust scaler**⁵ uses median and interquartile range (*IQR*) instead of mean and standard deviation:

$$\hat{x} = \frac{x - median(x)}{IQR(x)} = \frac{x - median(x)}{Q_{75}(x) - Q_{25}(x)} \quad (1.16)$$

where Q_{25} and Q_{75} determine the default low and high quantiles (different values can be used if needed).

Normalization enhances the numerical stability of algorithms based on data and accelerates them by utilizing smaller values. We will see its impact on machine learning in the following chapters. Now, we are ready to explore what a machine learning model is and the mechanism behind its learning process, known as *optimization*.

⁵[scikit-learn - RobustScaler](#)

1.3 Models

In this section, I will start by answering the question: “What is a model?”. We will discover that machine learning systems are statistical models that undergo some form of optimization to fit a dataset and create an implicit explanation of its distribution. ML models will be divided into parametric and non-parametric models, depending on whether their knowledge of the data is encoded into a set of internal tunable weights. Finally, I will briefly describe the main classes of models, highlighting their strengths and limits.

1.3.1 Statistical Models

A model is an entity that mimics a real system. It reflects its properties and behavior, allowing it to be used in place of the original system for a specific purpose. Depending on the purpose, the same system can be modeled in different ways, leading to different premises and outcomes Fig. 1.17.

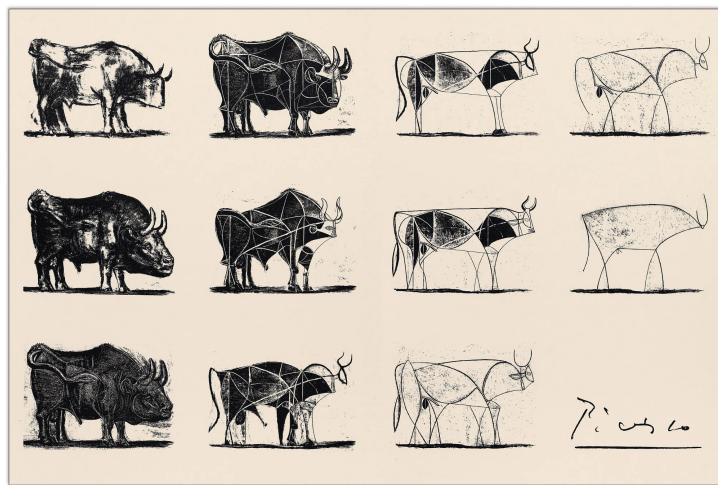


Fig. 1.17 Pablo Picasso, "Les 11 états successifs de la lithographie Le Taureau", 1945. "Picasso ended up where normally he should have started. It's true, but to achieve his pure and linear rendering of the bull, he had to pass through all of the intermediary stages. And when you stand before his eleventh bull, it's hard to imagine the work that went into it."

Following the data-driven approach described earlier, a model is designed not by hard-coding rules but by making the model automatically learn to emulate input-output relationships found in a dataset. This goal is achieved using a statistical

approach, i.e., maximizing the probability that the model's output coincides with the system's one through a process of *optimization*.

The model seeks the *optimal* explanation of the data; however, the space of possible solutions is infinite, and different solutions may be equally valid. Therefore, design choices regarding the model's structure enable the prioritization of one set of solutions over others. These choices enforce some a-priori criteria, called *inductive biases*, that restrict the space of solutions. For example, a fundamental criterion in modeling is *Occam's razor*, which states that “Entities are not to be multiplied without necessity” (The simplest explanation is usually the best). I will return to this concept when we cover *regularization*.

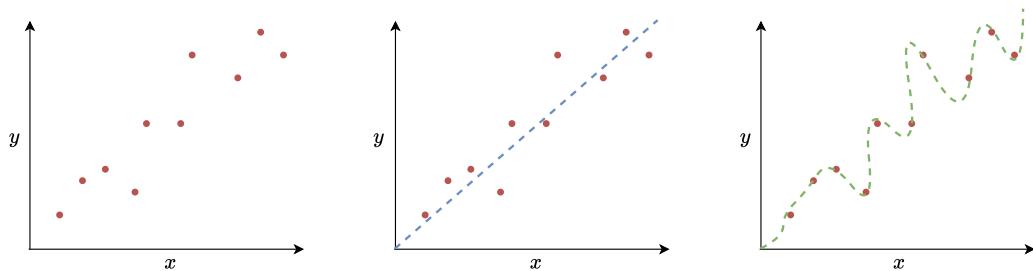


Fig. 1.18 Our inductive bias lets us prefer a simpler interpretation of reality even if it is not the best fitting for the data. Our experience of the world leads us to embrace the noise and variability of real phenomena.

By and large, when building a model, we inevitably choose a set of inductive biases. If we look at the leftmost plot in Fig. 1.18, our mind immediately prefers the linear interpretation, even if the rightmost one touches all the points. The primary reason is that linear relations frequently appear in the physical laws that govern nature. The first and simplest model we will study stems from this exact inductive bias and tries to give a linear representation of the data. However, first, we need to define the learning process rigorously, framing it as an optimization problem in which we aim to find the best model for the task at hand.

1.3.2 Generative and Discriminative Models

It is time I drew the line between two types of machine learning models: generative and discriminative. So far, we assumed our supervised pairs (x, y) come from some unknown probability distribution $p(x, y)$. By the product rule of

probability, we can decompose it equivalently as $p(y|x)p(x)$, or $p(x|y)p(y)$. Any model approximating $p(x)$ or $p(x|y)$ is called generative because you can use it to sample new input points. By contrast, a model that only approximates $p(y|x)$ is called discriminative. In simpler terms, discriminative models aim to separate data points into distinct classes and learn the boundaries using probability estimates and the principle of maximum likelihood. On the contrary, generative models model the actual data distribution rather than just decision boundaries. They learn what the data looks like, so they can be used to generate new data points.

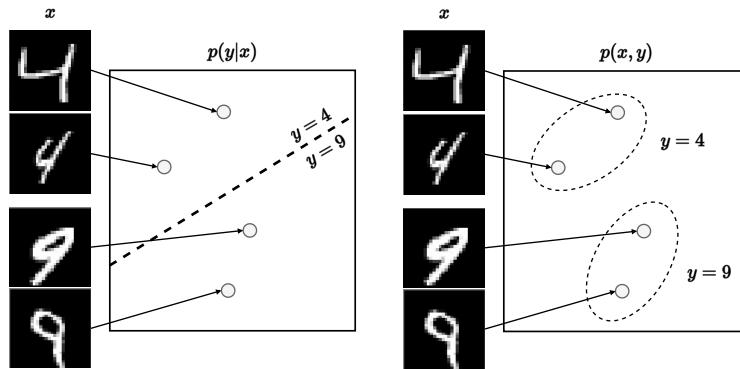


Fig. 1.19 A visual depiction of the key difference between discriminative (left) and generative (right) models.

As we can see in Fig. 1.19, a discriminative model tells you how likely a label is to apply to the instance, while a generative one tells you how likely a given example is. For example, models that predict the next word in a text sequence are typically generative because they can assign a probability to a sequence of words. Hence, generative models have to model deeper and are harder to train (they require more data).

1.3.3 Optimization

Optimization is the process of selecting the best element, subject to specific criteria, from a set of available alternatives. It is generally divided into discrete and continuous optimization, depending on the type of variables and functions involved. Machine learning can be framed as a continuous optimization problem if both its model and loss function are *differentiable*. In that case, we can formalize the search for the optimal model as follows:

Definition 1.3 (Continuous Optimization)

Given a dataset $\mathcal{S}_n = (x_i, y_i)$ and a loss function $l(y, \hat{y})$, a suitable optimization task to solve is the minimum average loss on the dataset achievable by any possible differentiable model f .

$$f^* = \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_i^n l(y_i, f(x_i)) \quad (1.17)$$

Where $f(x_i)$ is the prediction of the model for the i -th sample. For historical reasons, Eq. (1.17) is referred to as empirical risk minimization (ERM), where risk is used as a generic synonym for loss [19]. $f \in \mathcal{F}$ implies that f^* is not searched in the space of all possible models but rather in a subset defined by the design choices (leading to certain inductive biases).

In particular, for most of the approaches that we will study, we will be able to parametrize our model using a set of arrays w (called the model's parameters), and minimization is achieved by searching for the optimal value of these parameters via numerical optimization. Hence, we can train our model by solving the following optimization problem:

$$w^* = \arg \min_w \frac{1}{n} \sum_i^n l(y_i, f(x_i, w)) \quad (1.18)$$

Models of this type are called *parametric*, while models that do not fall under this scenario are called *non-parametric*.

1.3.4 Parametric Models

While the optimal value can be found analytically for simple cases, optimization is usually solved with numerical methods. For this reason, the loss function l should be differentiable and numerically stable. That is sometimes not straightforward, as there is an inherent tension between designing loss functions that encode our notion of performance and being useful for numerical optimization. I will revisit this concept later when we discuss the optimization mechanism. Now, let's start with our first machine learning model: the linear regressor.

Linear Regression

In the previous sections, I defined the essential elements of a supervised machine learning algorithm as the dataset \mathcal{S} , the model f , and the loss function l . In this section, I put all these elements together, considering the simplest possible case:

- $\mathbf{x} \in \mathbb{R}^c$ is the input sample. \mathbf{x} is an array of *features* (e.g., climate data measured in a specific place and time).
- $y \in \mathbb{R}$ is the label. In the unconstrained case, this is a regression task (e.g., predicting the measured temperature for the next day).
- \mathcal{S}_n is the dataset containing n (\mathbf{x}_i, y_i) pairs.
- $f : \mathbb{R}^c \rightarrow \mathbb{R}$ is parametrized as a *linear model* of parameters $\mathbf{w} \in \mathbb{R}^c$.
- $l_2(y_i, f(\mathbf{x}_i, \mathbf{w})) \in \mathbb{R}^+$ is the squared error loss.

Training this model consists of solving the optimization problem Eq. (1.18) and, hence, finding the optimal set of weights \mathbf{w}^* that minimizes l_2 . The linear model is defined as follows:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \quad (1.19)$$

The intuition is that the model assigns a fixed weight w_i to each input feature x_i and provides a prediction by summing all the effects for a given input \mathbf{x} . Whenever $\mathbf{x} = 0$, the model reverts to a default prediction b , called the *bias*.

Geometrically, the model defines a line for $c = 1$, a plane for $c = 2$, and a hyperplane for $c > 2$. From a notational point of view, we can sometimes avoid writing a bias term by extending \mathbf{x} with a constant term 1:

$$f\left(\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}\right) = \mathbf{w}^\top \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{w}_{1:c}^\top \mathbf{x} + w_{c+1} \quad (1.20)$$

We can further stack \mathbf{x}_i and y_i along the dataset dimension and obtain a compact vectorized representation, obtaining:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \implies \mathbf{y} = f(\mathbf{X}) = \mathbf{X}\mathbf{w} \quad (1.21)$$

In this way, the optimization problem resolves to:

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \quad (1.22)$$

This *least-squares* problem is convex in the weights of the model, as can be understood informally by noting that the equations describe a paraboloid in the space of the weights (a quadratic function). The global minima are then derived by imposing the gradient of f with respect to \mathbf{w} to zero.

$$\nabla_{\mathbf{w}}(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0 \implies \mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0 \implies \mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y} \quad (1.23)$$

These are called the *normal equations*. They can be written in the form of a linear system of equations $\mathbf{A}\mathbf{w} = \mathbf{b}$, with $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$ and $\mathbf{b} = \mathbf{X}^\top\mathbf{y}$. The solutions of the system will then be:

$$\mathbf{w}^* = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\dagger\mathbf{y} \quad (1.24)$$

under the condition that $(\mathbf{X}^\top\mathbf{X})$ is invertible. \mathbf{X}^\dagger is called the pseudo-inverse of \mathbf{X} (as $\mathbf{X}^\dagger\mathbf{X} = \mathbf{I}$ and \mathbf{X} is rectangular).

That is the only case in which we can solve the optimization problem in closed form. More complex cases will be infeasible either due to non-linearities or large dimensionality, resulting in high computational costs. I will then define a numerical method called *reverse-mode automatic differentiation* (also known as *backpropagation*) to compute the gradient of the loss. The gradient will be used by an *optimizer* to iteratively modify \mathbf{w} to approximate the desired behavior (by minimizing the loss).

Logistic Regression

In classification, $y_i \in \{1, \dots, m\}$, where m defines the number of classes. This task encompasses a range of tasks in computer vision (e.g., image classification and semantic segmentation) and natural language processing (e.g., next-token prediction and sentiment analysis). We can tackle this problem with slight variations from the regression case. Numerically, it is difficult to design a differentiable program that predicts an integer number without invalidating its differentiability and causing gradients to disappear. It is, in practice, easier to stay in the continuous optimization domain by framing the problem as a regression problem in $[1, m]$ and then rounding the output to obtain the predicted class.

$$\hat{y} = \text{round}(f(\mathbf{x})) \quad (1.25)$$

However, this approach would encourage the model to assume that numerically close classes are semantically close. For example, an output value of 0.6 would imply that the sample is assigned class 1 but is also similar to samples from class 0. This inductive bias is not always true, so we seek a more general case where a class can correlate with any other class. So, we prefer using *one-hot encoding*.

$$y^{OH} \sim (m), \quad y_i^{OH} = \begin{cases} 1 & \text{if } y = i \\ 0 & \text{otherwise} \end{cases} \quad (1.26)$$

Similarly to what was said before about predicting a real value and deriving the corresponding integer class, in the one-hot case, we select the class by computing the max index in a prediction array.

$$\hat{y} = \max(\hat{\mathbf{y}}) = \max(f(\mathbf{x})) \quad (1.27)$$

In the particular case in which $\hat{\mathbf{y}}$ is a *probability simplex*, i.e.,

$$\hat{y}_i \geq 0 \quad \text{and} \quad \sum_i \hat{y}_i = 1 \quad (1.28)$$

we can consider $\hat{\mathbf{y}}$ as a probability distribution, from which we are sampling the main mode. The one-hot label is, hence, a particular case in which all the probability is concentrated in a class.

To adapt our formulation of the linear regressor to this task, we need to modify the model to predict an array of m values. The parameters becomes a matrix $\mathbf{W} \sim (m, c)$ so that

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (1.29)$$

To guarantee that the output is a probability simplex, we add a non-linear, parameter-free transformation to the model, called *softmax*, to ensure that the output is a probability simplex.

$$\text{softmax}(\mathbf{x}) \sim (m), \quad \text{softmax}(\mathbf{x})_i = \frac{\exp(x_i/\tau)}{\sum_j \exp(x_j/\tau)} \quad (1.30)$$

Softmax is a soft (differentiable) approximation of the *argmax* function. The parameter τ determines how “smooth” the approximation is. With $\tau = 0$, the function coincides with argmax, but a value of 1 is standardly used.

Applying softmax to the predictions, we are no longer in the jurisdiction of linear models, so it is time to talk about **logistic regression**. I defined the linear classifier model as

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{W}\mathbf{x} + \mathbf{b}) = \text{softmax}(\mathbf{h}) \quad (1.31)$$

where the pre-softmax outputs \mathbf{h} are called *logits*. Applying the cross-entropy loss (Section 1.2.2) to the one-hot case we obtain:

$$l_{CE}(y^{OH}, \hat{\mathbf{y}}) = -\sum_i y_i^{OH} \log(\hat{y}_i) \quad (1.32)$$

As only one value of y^{OH} will be non-zero, we can rewrite it as follows:

$$l_{CE}(y, \hat{\mathbf{y}}) = -\log(\hat{y}_y) \quad (1.33)$$

Using softmax, we enforce a relation between different class predictions such that, by maximizing the probability predicted for a class with cross-entropy, we implicitly minimize it for the others. In conclusion, we obtain the *logistic*

regression (LR) formulation for classification as

$$LR(\mathbf{W}, \mathbf{b}) = \frac{1}{n} \sum_i^n l_{CE}(y_i^{OH}, \text{softmax}(\mathbf{W}\mathbf{x}_i + \mathbf{b})) \quad (1.34)$$

Unlike the least squares problem, we can no longer compute a closed-form solution. In the next chapter, I will introduce a method to solve the problem with a numerical optimization approach called *gradient descent*.

1.3.5 Non-parametric Models

Before delving deep into more complex models, it is essential to do a brief overview of non-parametric models. Except for historical reasons, these models give us a better understanding of how much inductive biases (also called *priors*) affect the performance of a model (for the better or the worse). From what we have seen so far, a parametric model can predict a probability distribution of the data in a predefined number of parameters. For parametric models, model complexity is constant regardless of the input data. Non-parametric models, however, do not deal with any underlying probabilistic model, and their complexity can vary depending on the training data.

k-Nearest Neighbors

When I defined the data space, we saw that we could exploit geometric concepts to highlight relevant patterns. The *k-nearest neighbors* (KNN) classification algorithm leverages the intuition that samples lying close to each other in the data space share similar properties and are likely to belong to the same class. So, KNN classifies a sample based on the class of its closest samples, called neighbors. In the trivial case $K = 1$, that translates into copying the nearest class. However, a single neighbor is highly subject to noise and outliers in the data. Therefore, it is common practice to consider multiple neighbors ($K > 1$) and select the mode of their classes. Look, for example, at Fig. 1.20, where the training data has been plotted in red or blue according to the label.

In the first case, the closest neighbor is from the red class, so x_{test} is classified as red. In the second case, which considers $K = 3$ neighbors, two blue samples

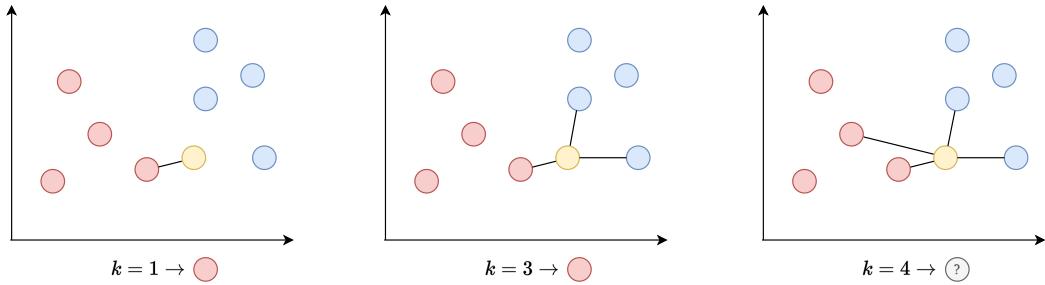


Fig. 1.20 KNN classification for different values of K . In the rightmost case, the outcome depends on how the distances are weighted, as there is no majority.

are added to the poll, and the result changes. Finally, for $K = 4$, there is no clear winner. In cases like this, the tie is broken by assigning a weight to each neighbor based on their distance from the sample (resulting, in the example, in classifying x_{test} as red). This approach is called weighted k-nearest neighbors (or WKNN). Moreover, KNN can also be used for regression by taking the average of the neighbor labels as the sample prediction.

Due to the need to compute the Euclidean distance (or other distance metrics) between the input sample and all samples in the training dataset, the computational cost of KNN increases rapidly with the number of samples and the data dimensionality.

Support Vector Machines

Support vector machines (SVM) stem from an inductive assumption similar to KNN. In this case, however, the model actively searches for an optimal separation boundary between classes. The boundary, which in the linear case is a hyperplane, maximizes the separation margin between the classes. In this way, the boundary is robust to noise and outliers that may confuse the model, meaning the solution is more general. That equals maximizing the distance between the separator and the nearest data point on each side. These data points are called **support vectors**. Mathematically, the separator is defined as the hyperplane

$$\mathbf{w}^\top \mathbf{x} + b = 0 \quad (1.35)$$

Hence, in the binary case, one of the classes lies in the hyperplane $\mathbf{w}^\top \mathbf{x} + b < 0$ and the other in the hyperplane $\mathbf{w}^\top \mathbf{x} + b > 0$. However, as stated before, we want to maximize the margin, and we can do so by solving the following optimization problem:

$$\mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i(\mathbf{w}^\top \mathbf{x} + b) \geq 1 \quad (1.36)$$

Here, the label y is conveniently coded to assume values $-1, 1$, such that the rightmost expression is valid for both classes. The relationship between the two equations is that the margin width is inversely proportional to $\|\mathbf{w}\|$. Indeed, the chosen **hard margin**, like in this case, will have a width of $\frac{2}{\|\mathbf{w}\|}$. Note that the solution to the problem is only affected by the samples on the edge of the margin, making it extremely efficient as processing the whole dataset is unnecessary (Fig. 1.21).

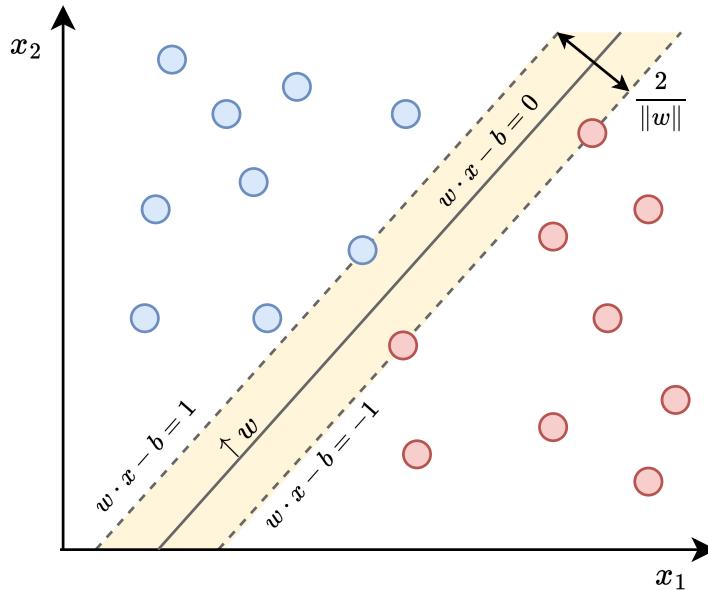


Fig. 1.21 Support vector machine with hard margin.

Of course, the hard margin formulation assumes the classes are linearly separable in the data space, meaning no sample crosses the optimal separator. In case some outliers lie on the other side of the hyperplane, it is necessary to use a different approach, called **soft margin**. It allows for the tolerance of outliers

via a parameter $C > 0$, which determines the trade-off between increasing the margin size and ensuring that the samples lie on the correct side of the margin. The optimization problem becomes

$$\begin{aligned} \mathbf{w}^*, b^* = \arg \min_{\mathbf{w}, b, \zeta} & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \zeta_i \\ \text{subject to } & y_i (\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \quad \forall i \in \{1, \dots, n\} \end{aligned} \quad (1.37)$$

In the case of non-linearly-separable classes, another variant of SVM can be applied using the so-called **kernel trick**. It transforms the data space into a feature space where classes are linearly separable. The transformation is performed through a kernel, a non-linear function that substitutes the dot product in the linear SVM:

$$k(\mathbf{a}, \mathbf{b}) = \langle \varphi(\mathbf{a}), \varphi(\mathbf{b}) \rangle \quad (1.38)$$

Typical kernel functions are polynomial, Gaussian radius basis function (RBF), and sigmoid. I invite the reader to explore the topic further by reading [20].

Dimensionality Reduction

As already stated, machine learning models (especially distance-based ones) tend to perform worse when the number of dimensions increases excessively. Indeed, samples become increasingly sparse as the number of dimensions increases due to the growing volume of the data space. As the number of dimensions increases, the distance between two points becomes more uniform or equidistant. This phenomenon occurs because the influence of any single dimension diminishes as the number of dimensions grows, leading to points being distributed more uniformly across the space Fig. 1.22. All these effects weaken the core assumption that same-class samples lie close to each other in the data space.

A solution to this problem, called the *curse of dimensionality*, is applying **dimensionality reduction** methods to make the data more compact while keeping

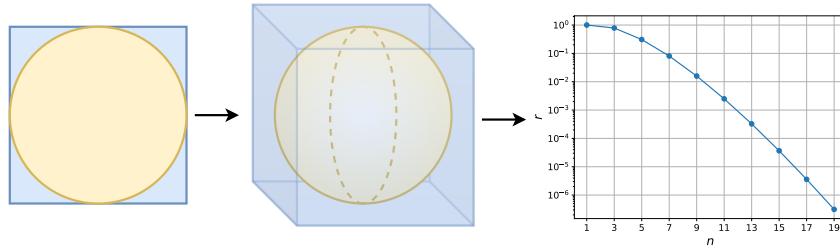


Fig. 1.22 The curse of dimensionality is given by the progressive sparsification of the data space as the dimensions increase. It can be visualized by considering the volume ratio between a hypersphere inscribed in a hypercube and the hypercube itself. As the number of dimensions grows, the ratio rapidly decays.

most of the information. The most straightforward approaches select a subset of the input variables and discard the others, while more effective ones project samples to a lower-dimensional space. The transformation can be either linear or non-linear, aiming to conserve most data informativeness through a specific strategy. For example, principal component analysis (PCA) performs linear mapping in such a way that the variance of the data in the low-dimensional representation is maximized (Fig. 1.23). In practice, the covariance matrix of the data is constructed:

$$\Sigma = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & \dots & Cov(X_1, X_n) \\ Cov(X_2, X_1) & Var(X_2) & \dots & Cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & Cov(X_n, X_2) & \dots & Var(X_n) \end{bmatrix} \quad (1.39)$$

where variance and covariance are defined as

$$\begin{aligned} Var(X) &= \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2, \\ Cov(X, Y) &= \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y}) \end{aligned} \quad (1.40)$$

and \bar{X} is the mean value of X . The matrix eigenvectors corresponding to the largest eigenvalues (the principal components) can now be used to reconstruct a significant fraction of the variance of the original data. Indeed, the eigenvectors

represent the directions in the data space corresponding to the maximum variance in the data. Data normalization is essential in this process, as unbalanced values can easily favor large-range, less informative components rather than small-range, more informative ones. For this reason, standardization along feature components is always applied before computing Σ . Finally, the matrix formed by the selected eigenvectors Σ_* is used to project the dataset X to the new low-dimensional space:

$$\tilde{X} = \Sigma_*^\top X^\top \sim (n, m) \text{ with } m < c \quad (1.41)$$

Where n is the number of samples and m is the new, smaller number of features.

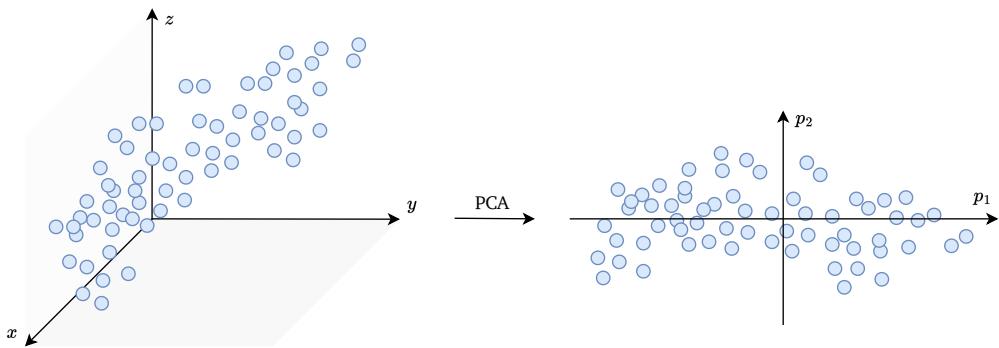


Fig. 1.23 Pricipal component analysis on 3D data.

The number of components is chosen by balancing it with the resulting variance. Variance is a good way to quantify the information contained in a dimension. Like SVM, the kernel trick can adapt PCA to non-linear transformations.

Dimensionality reduction methods are also instrumental in visualizing multi-dimensional data or vectors in a 2D/3D representation. t-SNE [9], for instance, was explicitly proposed for this scope. It performs a non-linear data projection, allowing nearby points to model similar objects. First, t-SNE computes the similarity between any pair of samples and then constructs a transformation to preserve it in the low-dimensional space. Unlike PCA, the size of clusters produced by t-SNE and the distance between them are not informative for learning purposes.

In conclusion, these methods can be considered a way to extract the most essential features from high-dimensional data through simple transformations. Hence,



Fig. 1.24 MNIST visualization through t-SNE.

they are rightfully considered machine learning models even if a specific task does not define their objective. We can say that these models ease the work of other models that use the transformed features for classification or regression tasks. This intuition of possibly decoupling *representation learning* from prediction is at the core of more complex models that have been historically referred to as *deep learning*.

Bibliography

- [1] Epoch AI, *Key trends and figures in machine learning*, Accessed: 2025-01-13, 2025.
- [2] V. G. Cerf, *ASCII format for network interchange*, RFC 20, 1969.
- [3] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” in *International Conference on Learning Representations*, 2013.
- [4] Y. Shibata *et al.*, “Byte pair encoding: A text compression scheme that accelerates pattern matching,” Technical Report DOI-TR-161, Department of Informatics, Kyushu University, Tech. Rep., 1999.
- [5] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.
- [6] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006, vol. 2, pp. 1122–1128.
- [7] J. Denker *et al.*, “Neural network recognizer for hand-written zip code digits,” *Advances in neural information processing systems*, vol. 1, 1988.
- [8] T. M. Mitchell, *Machine Learning*, 1st ed. USA: McGraw-Hill, Inc., 1997, ISBN: 0070428077.
- [9] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [10] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [11] R. Bellman, “A markovian decision process,” *Journal of mathematics and mechanics*, pp. 679–684, 1957.
- [12] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, “How to train your robot with deep reinforcement learning: Lessons we have learned,” *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.
- [13] J. Jumper *et al.*, “Highly accurate protein structure prediction with alphafold,” *nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [14] D. Silver *et al.*, “Mastering the game of go without human knowledge,” *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

- [15] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [16] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, “Are loss functions all the same?” *Neural computation*, vol. 16, no. 5, pp. 1063–1076, 2004.
- [17] T.-Y. Ross and G. Dollár, “Focal loss for dense object detection,” in *proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2980–2988.
- [18] S. Jadon, “A survey of loss functions for semantic segmentation,” in *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, 2020, pp. 1–7.
- [19] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.
- [20] R. Liao, *Support vector machines*. Springer, 2015.

Chapter 2

Deep Learning

As mentioned earlier, in standard programming, the designer assembles the proper building blocks (in this case, functions) to solve a problem deterministically. In this chapter, I show that we can do something similar for differentiable statistical models by composing a sequence of simple building blocks (in this case, layers). Layers can implement different linear and nonlinear functions, turning a machine learning model into a stack of simple operations. For historical reasons, these models have been referred to as *neural networks*. The term originates from the initial attempts to replicate the electrical phenomena between neurons within the brain. As the networks became increasingly complex with the addition of layers, the term *deep learning* was coined to highlight their ability to extract complex patterns hidden deep within the data. We will now embark on a quest to uncover the magic behind deep learning, beginning with its most straightforward applications and progressing to the latest advancements in the field. Get ready!

2.1 Neural Networks

Linear models are simple, but they carry a fundamental limitation: they cannot model nonlinear relationships across features. Considering the toy XOR dataset, where each feature can only take values in $\{0, 1\}$, the output is positive whenever only one of the two features is positive. Despite its simplicity, this is a non-linearly separable problem and cannot be solved with 100% accuracy by any linear model (Fig. 2.1).

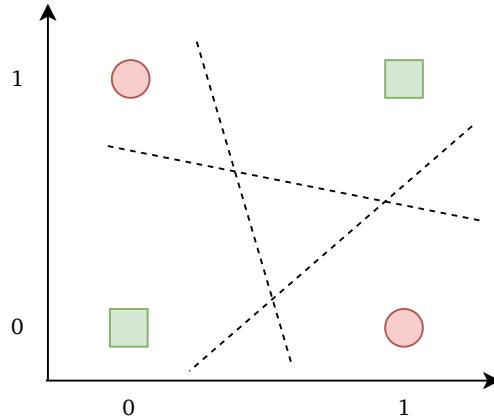


Fig. 2.1 No linear model can separate the XOR dataset perfectly, putting all squares (representing $y = 0$) on one side of the decision boundary and all circles ($y = 1$) on the other. Hence, the dataset is not linearly separable.

To be able to solve such tasks, we have to build more complex machine learning models. One idea is to compose multiple simple functions to model more elaborate patterns in data.

$$f(x) = (f_2 \circ f_1)(x) = f_2(f_1(x)) \quad (2.1)$$

This way, we could chain as many operations as we want, considering each will instantiate its set of parameters. If we take f_i as the linear models we saw in the previous chapter, i.e.,

$$\begin{aligned} \mathbf{h} &= f_1(\mathbf{x}) = \mathbf{W}_1 \mathbf{x} + \mathbf{b}_1 \\ y &= f_2(\mathbf{h}) = \mathbf{w}_2^\top \mathbf{h} + b_2 \end{aligned} \quad (2.2)$$

we obtain a classifier composed of two separate linear models. However, the resulting function can be rewritten as

$$y = (\mathbf{w}_2^\top \mathbf{W}_1) \mathbf{x} + (\mathbf{w}_2^\top \mathbf{b}_1 + b_2) \triangleq \mathbf{A} \mathbf{x} + c \quad (2.3)$$

which is, again, a linear model with more parameters. By stacking linear models, we obtain linear models equivalent to a single layer. We can introduce a nonlinearity between f_1 and f_2 to decouple the two models.

$$\begin{aligned}\mathbf{h} &= f_1(\mathbf{x}) = \phi(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\ y &= f_2(\mathbf{h}) = \mathbf{w}_2^\top \mathbf{h} + b_2\end{aligned}\tag{2.4}$$

where ϕ is a nonlinear function. That is the first and simplest formulation of an *artificial neural network* (ANN), which can be referred to as either a *fully connected* (FC) model or a *multi-layer perceptron* (MLP). The reason behind these names is historical, so I need to briefly recap the steps that led to such formulation to understand the intuition behind it and the consequent terminology.

2.1.1 Artificial Neuron

The term *neural network* has quite a long history. The theoretical basis for contemporary neural networks was independently proposed by Alexander Bain [1] and William James [2] in the late 19th century. Both posited that human thought emerged from interactions among large numbers of neurons inside the brain. In 1949, Donald Hebb [3] described *Hebbian learning*, the idea that neural networks can change and learn over time by strengthening a synapse every time a signal travels along it (Fig. 2.2).

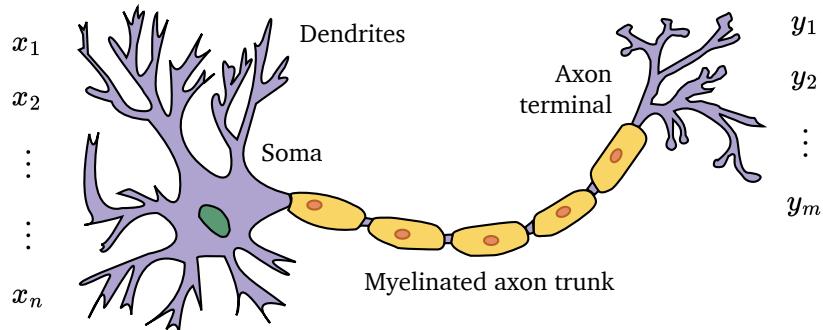


Fig. 2.2 A visual analogy between a real neuron in the pre-frontal cortex and the corresponding artificial model.

Artificial neural networks (ANNs) were initially used to model biological ones in the 1930s. However, starting with the invention of the perceptron by Warren McCulloch and Walter Pitts in 1943 [4], these networks have become increasingly used for machine learning applications. The formulation of the perceptron is

derived from the observation that neurons in the cortex process information by aggregating stimuli from other neurons. Specifically, a neuron “fires” (i.e., transmits a signal forward) only if the sum of input stimuli is higher than a threshold. For this reason, the perception was first implemented with the step activation function $\mathbb{1}$, resulting in a binary neuron (Fig. 2.3).

$$\mathbb{1}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (2.5)$$

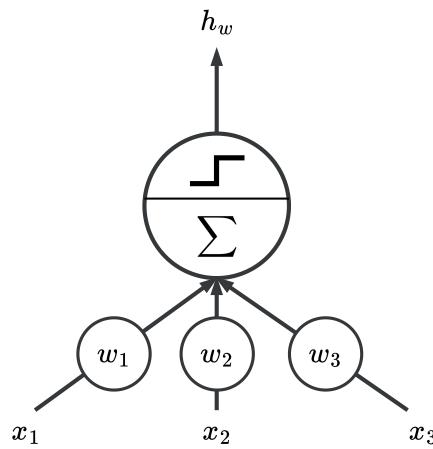


Fig. 2.3 The artificial neuron proposed by [4].

2.1.2 Multi-layer Perceptron

In general, each of the 8.6×10^{10} neurons in the cortex is connected to multiple input and output neurons, generating around 1.5×10^{14} connections called *synapses* [5]. To reproduce such density, the first ANN proposed by Frank Rosenblatt, the multi-layer perception (MLP), was designed as a composition of three *layers* of artificial neurons. In the example reported in Fig. 2.4, $x = [x_1, x_2, 1]$ is the *input layer*, f_2 is the *output layer* generating the output y , and f_1 is called the *hidden layer*. Each fully connected (FC) layer f_i can be written as:

$$FC(X) = \phi(XW + b) \quad (2.6)$$

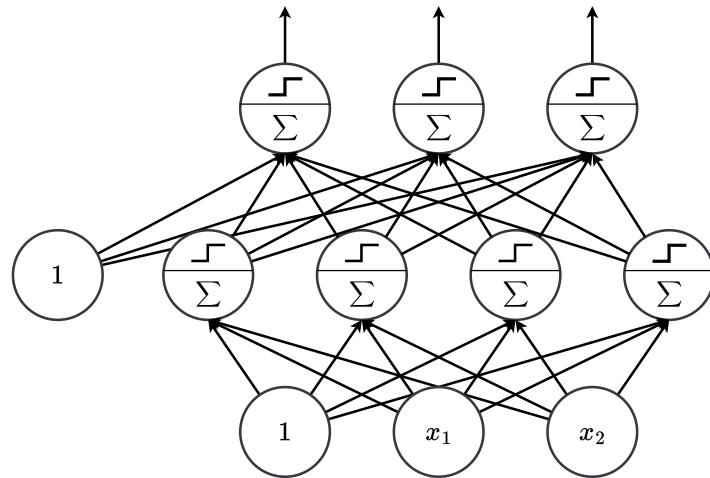


Fig. 2.4 A simple fully-connected layer with binary activation function and bias.

The parameters of the layer are the matrix $\mathbf{W}_i \sim (c_i, c_{i-1})$ and the bias vector $\mathbf{b} \sim (c_i)$, for a total of $c_i(c_{i-1}+1)$ parameters. c_i is the number of neurons at layers i (also referred to as layer width) and ϕ is the *activation function*. Following this architecture, a model can be defined by chaining together instances of such layers and obtaining a general MLP or FC model:

$$FC(x) = (FC_l \circ FC_{l-1} \circ \dots \circ FC_1)(x) \quad (2.7)$$

where l is the number of layers of the model.

In 1958, Rosenblatt himself implemented the first ANN in hardware [6], and since then, models have started becoming increasingly different from their biological counterparts. For example, we can replace the threshold function with another nonlinearity to obtain dense output values instead of binary responses. I will explore the possibilities in the choice of ϕ in the following sections. However, even this simple abstraction of neural information processing has a significant property worth explaining: the *universal approximation theorem*.

2.1.3 Universal Approximation

In the previous chapter, I stated that linear models can only model linearly separable data with zero error. From a theoretical perspective, we may ask ourselves what the significance of nonlinearity is, i.e., what the class of functions that can be approximated by adding hidden layers is. In 1989, George Cybenko first demonstrated that a single hidden layer is enough to have universal approximation capabilities [7]. In particular, he proved that:

Theorem 2.1 (Universal Approximation)

Given a continuous function $g : \mathbb{R}^d \rightarrow \mathbb{R}$, we can always find a model $f(x)$ in the form of an MLP with a single hidden layer and nonlinear activation functions, such that for any $\varepsilon > 0$:

$$|f(x) - g(x)| \leq \varepsilon, \quad \forall x \quad (2.8)$$

where the result holds over a compact domain. Stated differently, one-hidden-layer MLPs are “dense” in the space of continuous functions.

Of course, this theorem makes use of the fact that the width of the hidden layer of the model can grow without bounds. Hence, for any x for which the previous inequality does not hold, we can always add a new unit to reduce the approximation error. However, that is often infeasible in practice, as having an unbounded number of parameters does not pair well with any real problem. We will see that keeping the number of parameters fixed and adding more layers is preferable to increasing their width [8]. Given this interesting property of MLPs, I want to focus on the practical design of the models themselves, whose behavior can be more complex and challenging to control and design than these theorems suggest.

2.1.4 Activation Function

I close this section by briefly explaining possible choices for ϕ . Although activation functions were initially inspired by brain signals, any nonlinear continuous function can be adopted in theory. Nevertheless, mathematical properties and

empirical effectiveness have narrowed the choice to certain families of functions over the years.

One of the main reasons for abandoning the foremost threshold function for binary neurons was its non-differentiable nature. Indeed, using numerical optimization methods can lead to instability in critical points. A smooth approximation of $\mathbb{1}$ is the *sigmoid* function σ :

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.9)$$

which, by the way, corresponds to the *softmax* function in the binary class case, i.e., outputs a probability distribution.

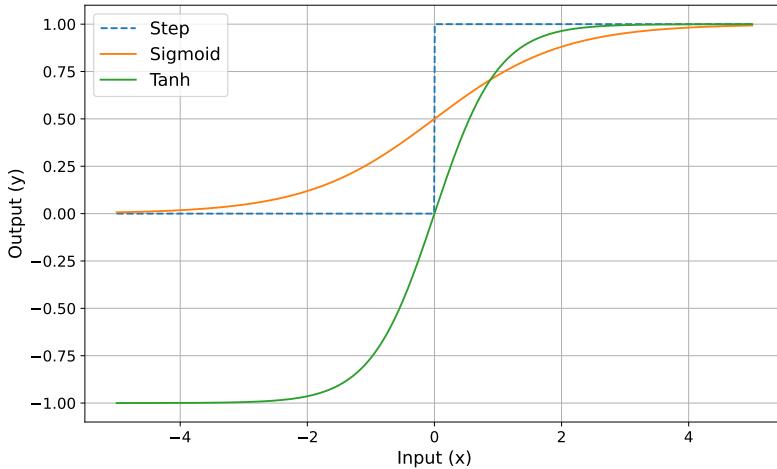


Fig. 2.5 Visual comparison of step, sigmoid, and tanh activation functions.

Since x is usually a multi-dimensional array, activation functions are applied independently to each element (and are therefore called *element-wise* operations). σ can be shifted and scaled to serve different purposes. A common differentiable variant is, for instance, the *hyperbolic tangent*, which maps values in the range $[-1, 1]$ (Fig. 2.5):

$$\tanh(x) = 2\sigma(x) - 1 \quad (2.10)$$

We can push this concept further and propose a parametric version of σ where the slope of the linear section is tunable during the learning process:

$$\sigma_a(x) = \sigma(ax) \quad (2.11)$$

This approach applies to all the activation functions I will present. Modern deep learning models predominantly use a different family of functions, the simplest and most popular of which is the *rectified linear unit* (ReLU) [9, 10].

$$\text{ReLU}(x) = \max(0, x) \quad (2.12)$$

As you can see, it is a pretty different formulation from the bio-inspired functions we studied earlier. ReLU essentially leaves all positive inputs unchanged and sets all negative ones to zero. In the next section, I will provide a more detailed analysis of why this function benefits learning, despite its non-differentiable point. For the moment, we can notice two main properties:

- It is straightforward and, hence, introduces negligible computational overhead.
- It is still nonlinear enough to avoid collapsing into a linear data representation.
- It generates sparse outputs, as all the negative inputs are set to zero.

To avoid losing too much information by zeroing all negative inputs, we can give a slight slope to the left-hand part of the function, thus obtaining a *LeakyReLU* [11]:

$$\text{LeakyReLU}(x) = \max(cx, x), \quad 0 < c < 1 \quad (2.13)$$

A typical value for c is 10^{-2} . If we wish to make c learnable, we obtain the general *parametric* ReLU (PReLU) [12]. Another direction of customization is solving the non-differentiable point $x = 0$. This procedure, however, complicates the formulation and inevitably adds some overhead. SoftPlus, ELU, and GELU fall under this category. Apart from some minor details in the negative quadrant, they are all relatively similar (Fig. 2.6).

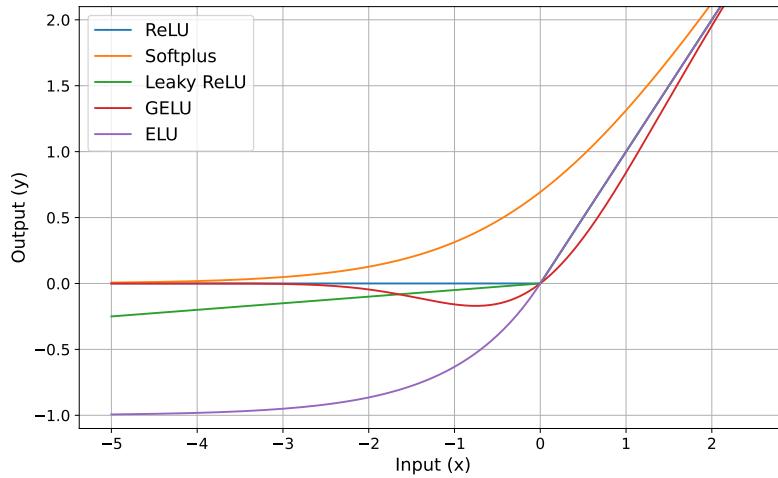


Fig. 2.6 Visual comparison of ReLU and four variants: LeakyReLU, Softplus, ELU, and GELU. LeakyReLU is shown with $\alpha = 0.05$ for better visualization.

In conclusion, in this section, we started our journey into deep learning by describing the architecture of the simplest neural networks. We just stacked multiple linear layers interleaved by basic nonlinear mappings to obtain fully connected models. We also understood that despite their simplicity, these networks can already approximate any continuous function. However, we still lack a method to solve the optimization problem associated with the task. In the next section, I will address the topic of numerical optimization, creating a framework generally used in all cases where an optimal solution cannot be computed in closed form. Buckle up!

2.2 Optimization

In this section, I describe the mathematical tools and computational methods that allow parametric models to learn how to solve a task by approximating a desired behavior. As we saw in the previous chapter, supervised learning achieves this goal by selecting a suitable loss function and solving the resulting optimization problem Eq. (1.18).

It involves finding the optimal set of weights that minimizes the loss function across the entire dataset. When the solution cannot be found analytically, we resort to numerical algorithms that iteratively approximate w^* starting from

an initial guess \mathbf{w}_0 . For every iteration of the algorithm, we take a step in the vectorial space \mathbb{R}^d , where d is the number of parameters of the model. A single step at time t can be decomposed into its magnitude and direction components (η_t and \mathbf{p}_t , respectively), obtaining:

$$\mathbf{w}_t = \mathbf{w}_{t-1} + \eta_t \mathbf{p}_t \quad (2.14)$$

We call η_t the *learning rate*, as it quantifies the rate of change of weights toward the optimal solution (Fig. 2.8); we call \mathbf{p}_t the *descent direction* if the new estimate of the weights reduces the loss value ($l_{t+1} < l_t$). If we ensure that we follow a descent direction step by step, the iterative algorithm will eventually converge to a minimum. But how do we ensure that? We must remember that our model is differentiable and we can exploit gradients.

2.2.1 Gradient Descent

In vector calculus, the gradient of a scalar function $y = f(\mathbf{x})$ is defined as the vector containing the partial derivatives of y with respect to each element of \mathbf{x} . In formulas:

$$\nabla f(\mathbf{x}) = \partial f(\mathbf{x}) = \begin{bmatrix} \partial_{x_1} f(\mathbf{x}) \\ \dots \\ \partial_{x_d} f(\mathbf{x}) \end{bmatrix} \quad (2.15)$$

Like in the simplest one-dimensional case, each partial derivative can be understood geometrically as the slope of the tangent passing through the point \mathbf{x} . This point of view is fundamental because the slope tells us how the function evolves in a close neighborhood: for a positive slope, the function increases to the right and decreases to the left (again, for a sufficiently small interval), while for a negative slope, the opposite is true. The gradient expresses this behavior on the directional axes of \mathbf{x} (Fig. 2.7).

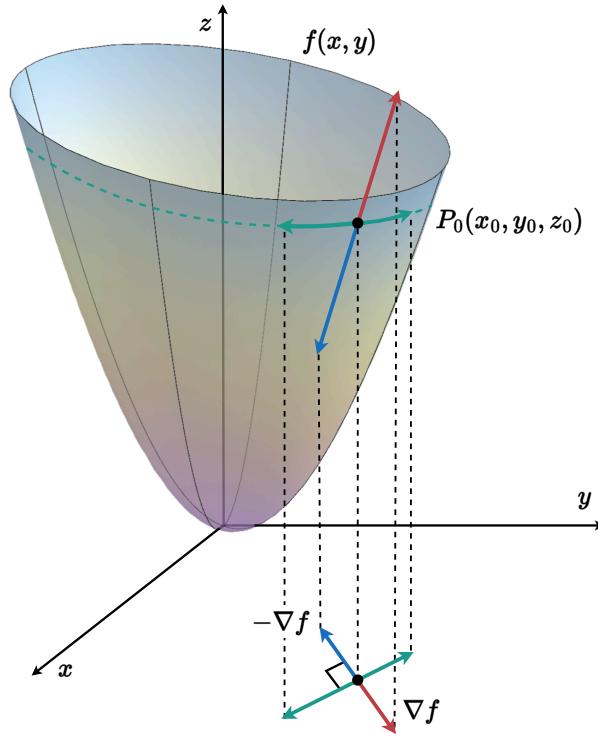


Fig. 2.7 Visualization of the gradient of a two-dimensional function f computed in P_0 . The **blue** arrows indicate the direction of the steepest decrease, while the opposite **red** ones indicate the direction of the steepest increase. The **green** arrows indicate the directions for which there is no change in the value of f .

To compute the derivative along a generic direction \mathbf{v} (where $\|\mathbf{v}\| = 1$), it is sufficient to operate the dot product of the gradient with \mathbf{v} (called the displacement vector).

$$D_{\mathbf{v}}f(\mathbf{x}) = \langle \nabla f(\mathbf{x}), \mathbf{v} \rangle = \sum_i \partial_{x_i} f(\mathbf{x}) \cdot v_i \quad (2.16)$$

which essentially projects the gradient in the desired direction. $D_{\mathbf{v}}$ is hence called the *directional derivative* operator. As we said, a descent direction goes toward lower loss, so using the concept of directional derivative, we can state that:

$$\mathbf{p} \text{ is a descent direction at step } t \Rightarrow D_{\mathbf{p}} f(\mathbf{x}_{t-1}) \leq 0 \quad (2.17)$$

Now, we can express the dot product in D_v using the cosine similarity notation

$$D_p f(\mathbf{x}_{t-1}) = \langle \nabla f(\mathbf{x}_{t-1}), \mathbf{p} \rangle = \|\nabla f(\mathbf{x}_{t-1})\| \|\mathbf{p}\| \cos(\alpha) \quad (2.18)$$

Where α is the angle between \mathbf{p} and $\nabla f(\mathbf{x}_{t-1})$. Since we chose $\|\mathbf{v}\| = 1$, Section 2.2.1 can be simplified as

$$\|\nabla f(\mathbf{x}_{t-1})\| \cos(\alpha) \leq 0 \quad (2.19)$$

As the norm of a vector is always ≥ 0 , the relation is satisfied only for values of α in the range $[\pi/2, 3\pi/2]$. In particular, the minimum is reached for $\alpha = \pi$, for which the descent direction becomes

$$\mathbf{p} = -\nabla f(\mathbf{x}_{t-1}) \quad (2.20)$$

Which represents the *steepest descent direction* for f at step t . Plugging this solution into Section 2.2, we obtain the *gradient descent* formulation:

Definition 2.1 (Gradient Descent)

Given a differentiable function $f(\mathbf{x})$, a starting point \mathbf{x}_0 , and a step size η , gradient descent (GD) proceeds as:

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \eta \nabla f(\mathbf{x}_{t-1}) \quad (2.21)$$

It can be proved that, for the class of *convex* functions¹, gradient descent always reaches an absolute minimum regardless of \mathbf{x}_0 . However, we cannot generally determine the convexity of f , which in deep learning is usually quite complex and almost undoubtedly non-convex. In these cases, gradient descent is only guaranteed to reach a stationary point (i.e., one where the gradient is zero).

¹In mathematics, a real-valued function is called *convex* if the line segment between any two distinct points on the graph of the function lies above or on the graph between the two points.

Nothing more can be said without recurring to higher-order derivatives. Hence, \mathbf{x}_0 has a significant influence on the optimization result. Empirically, GD converges to good local minima when initialized from a reasonable starting point. I will introduce a couple of initialization strategies in the following sections. But first, we need to understand how loss gradients are systematically and efficiently computed for each model parameter.

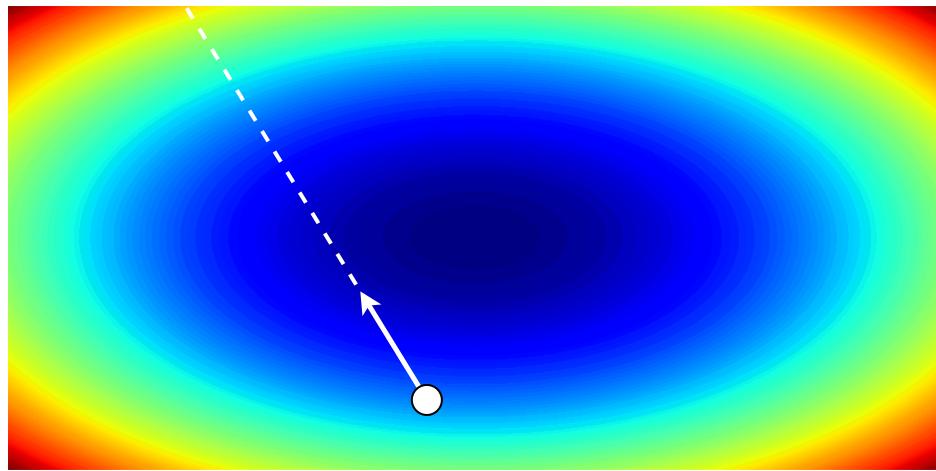


Fig. 2.8 Visualizing the effect of step size. We start at a particular spot and evaluate the gradient, which indicates the direction of the steepest decrease in the loss function (here represented by color). Small steps are likely to lead to consistent, yet slow, progress. Large steps can lead to better progress, but are more risky. Note that eventually, for a large step size, we will overshoot and make the loss worse. The step size (or *learning rate*) is a hyperparameter we must tune carefully.

2.2.2 Back-propagation

To set the problem, we assume we have at our disposal a set of *primitives*, i.e., functions for which we can compute gradients:

$$\mathbf{y} = f_i(\mathbf{x}, \mathbf{w}_i) \quad (2.22)$$

Each primitive represents an operation on an input vector $\mathbf{x} \sim (c_i)$, parameterized by the vector $\mathbf{w}_i \sim (p_i)$ (e.g., the weights of a linear projection), and giving as output another vector $\mathbf{y} \sim (c'_i)$. We, therefore, assume we can compute the partial derivatives with respect to the inputs \mathbf{x} and \mathbf{w}_i .

Consider a sequence of l primitive calls representing our model:

$$\begin{aligned}\mathbf{h}_1 &= f_1(\mathbf{x}, \mathbf{w}_1) \\ \mathbf{h}_2 &= f_2(\mathbf{h}_1, \mathbf{w}_2) \\ &\dots \\ y &= f_l(\mathbf{h}_{l-1}, \mathbf{w}_l)\end{aligned}\tag{2.23}$$

That is called an *evaluation trace* of the program. Roughly, the first $l-1$ operations represent the layers of a differentiable model, and operation l is the loss function (e.g., cross-entropy). Hence, the output of our program is always a scalar since we require it for numerical optimization. We can abbreviate the model with $F(\mathbf{x})$ and formalize the definition of *automatic differentiation*.

Definition 2.2 (Automatic Differentiation)

Given a program $F(\mathbf{x})$ composed of a sequence of differentiable primitives, automatic differentiation (AD) refers to the task of simultaneously and efficiently computing all individual input and weight gradients:

$$AD(F(\mathbf{x})) = \{\partial_{\mathbf{w}_i} y\}_{i=1}^l\tag{2.24}$$

There are two main approaches to AD, namely *forward* and *backward-mode differentiation*. I will address the latter, as it is computationally more efficient and almost always preferred by machine learning frameworks.

Reverse-mode automatic differentiation (R-AD) (also called *backpropagation*) was initially proposed in the 60s [13, 14] and then popularized 30 years later by [15]. It leverages the so-called *chain rule* (Fig. 2.9) that expresses the derivative of the composition of two differentiable functions:

$$h'(\mathbf{x}) = f'(g(\mathbf{x}))g'(\mathbf{x}).\tag{2.25}$$

It can be outlined as:

1. We start by executing the entire program to be differentiated, storing all intermediate outputs.

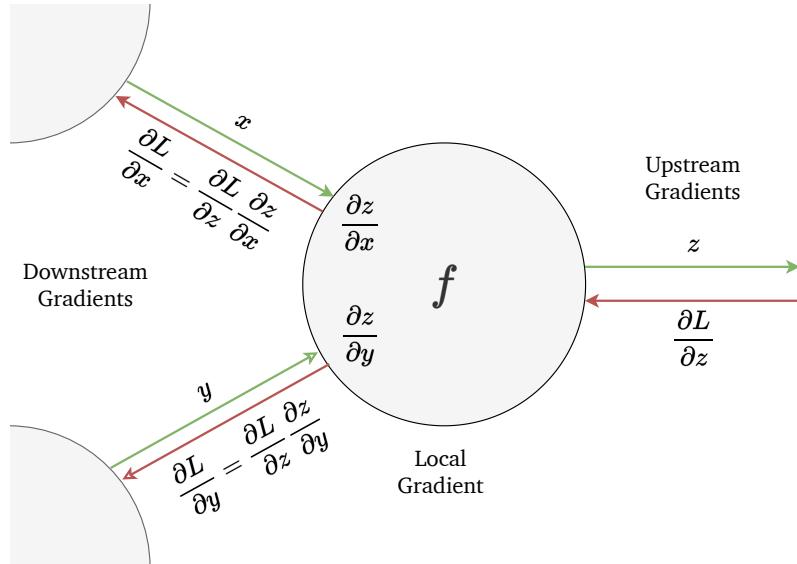


Fig. 2.9 The chain rule at the core of backpropagation.

2. We formulate the gradient of the loss by unrolling its definition with the chain rule:

$$\nabla_{w_i} y = [\partial_{h_{l-1}} \mathbf{h}_l] \dots [\partial_{h_i} \mathbf{h}_{i+1}] [\partial_{w_i} \mathbf{h}_i] \quad (2.26)$$

3. Moving in reverse order, i.e., for an index i ranging in $\{l, l-1, l-2, \dots, 1\}$, we compute the gradient with respect to the i -th layer.
4. As soon as the gradient for a parameter is computed, the corresponding weight is updated using Definition 2.1.

We then update our parameters in the opposite direction of the gradients, with the learning rate determining the magnitude of the update. In this way, gradient descent is guaranteed to converge to the global minimum for convex error surfaces and a local minimum for non-convex surfaces.

2.2.3 Stochastic Gradient Descent

In the original formulation, GD computes gradients using the loss for every sample in the dataset, then takes a step in the average direction of all the directions obtained. This approach is linear in the dimension of the dataset n , as it requires

processing all the samples for a single step. That is basically unfeasible for massive datasets with millions of samples. A faster and statistically advantageous alternative is evaluating each optimization step over a small *i.i.d.* subset of samples (called a *mini-batch*). Considering a mini-batch $\mathcal{B}_r \subset \mathcal{S}_n$, with $r \ll n$, we can compute an approximated loss by only considering the mini-batch as:

$$\tilde{L} = \frac{1}{r} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{B}_r} l(y_i, f(\mathbf{x}_i)) \approx \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{S}_n} l(y_i, f(\mathbf{x}_i)) \quad (2.27)$$

The borderline case, $r = 1$, involves updating weights based on a single sample, resulting in high variance. In general, lower dimensions r of the mini-batch result in faster iterations with higher gradient variance, whereas higher r values result in slower, more precise iterations. Memory is generally the biggest bottleneck for large models, and r can be selected to fill up the available hardware for each iteration. This strategy, called *stochastic gradient descent* (SGD), implies that a single training pass over the whole dataset is divided into n/r steps, each corresponding to a weight update. The entire pass over the dataset is called an *epoch*. Due to being averaged on a small number of samples, SGD introduces more noise (variance) compared to GD (Fig. 2.10).

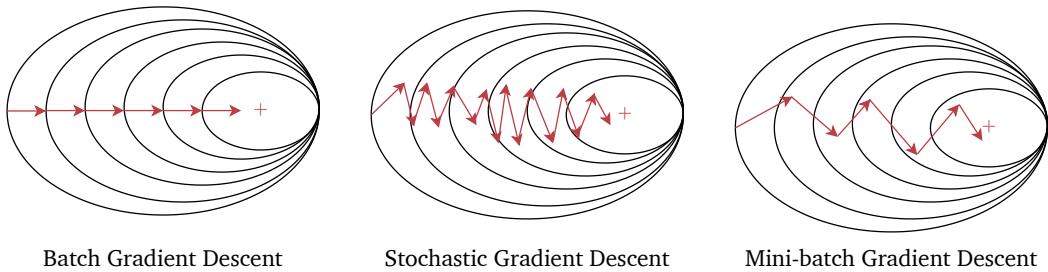


Fig. 2.10 Visual comparison between batch gradient descent, stochastic gradient descent, and mini-batch gradient descent.

Although noise can slow down convergence compared to smooth GD, it is sometimes beneficial to escape local minima and plateaus that may trap the optimizer. Choosing a proper learning rate η in this scenario can be difficult. A learning rate that is too small leads to slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge. For this reason, *learning rate scheduling* is a common practice for adjusting the step size during training [16]. Annealing the learning rate according to a pre-defined schedule increases the convergence rate

of SGD by reducing oscillations in late epochs. Moreover, several techniques have been proposed to accelerate the convergence of the optimization algorithm by selecting better descent directions. We will now briefly see the most relevant ones.

2.2.4 Momentum

One of the simplest yet most essential methods to improve convergence is *momentum*. SGD has trouble navigating ravines (i.e., areas where the surface curves much more steeply in one dimension than another), which are common around local optima. In these scenarios, SGD oscillates across the ravine's slopes, making hesitant progress toward the local optimum. It involves adding an integrative component to the optimization, preserving some direction from the previous gradient iteration.

$$\begin{aligned}\mathbf{g}_t &= -\eta_t \nabla f(\mathbf{x}_{t-1}) + \lambda \mathbf{g}_{t-1} \\ \mathbf{x}_t &= \mathbf{x}_{t-1} + \mathbf{g}_t\end{aligned}\tag{2.28}$$

where $\mathbf{g}_0 = \mathbf{0}$ and the coefficient λ determines how much the previous term is damped. Essentially, using momentum is like pushing a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity if there is air resistance). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same direction and reduces updates for dimensions whose gradients change direction. As a result, we gain faster convergence and reduced oscillation.

An improvement on this method was proposed by Nesterov in 1983 [17] and involves using a more accurate estimate of \mathbf{x}_{t-1} for gradient computation. We know that we will use our momentum term \mathbf{g}_t to update the parameters, allowing us to directly compute the gradient with respect to the approximate future weights.

$$\mathbf{g}_t = -\eta_t \nabla f(\mathbf{x}_{t-1} + \lambda \mathbf{g}_{t-1}) + \lambda \mathbf{g}_{t-1}\tag{2.29}$$

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \mathbf{g}_t\tag{2.30}$$

This anticipatory update prevents us from going too fast and increases responsiveness (Fig. 2.11).

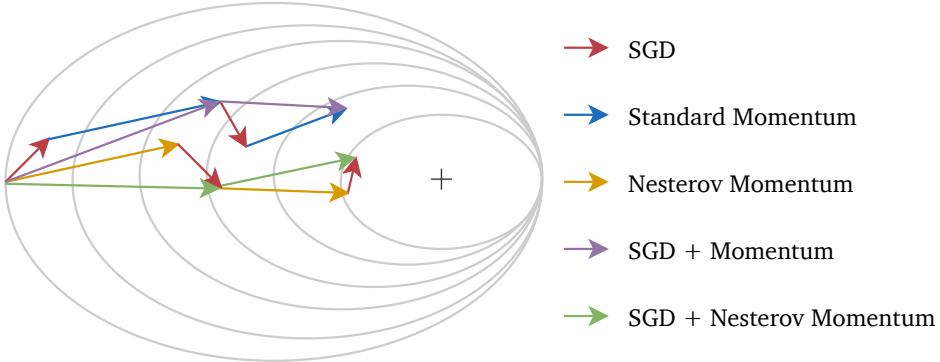


Fig. 2.11 Standard momentum first computes the current gradient (**red arrow**) and then takes a step in the direction of the updated accumulated gradient (**blue arrow**), resulting in the **violet arrow**. Nesterov's momentum first makes a big jump in the direction of the previously accumulated gradient (**yellow arrow**), measures the gradient, and then corrects (**red arrow**), which results in the complete update (**green arrow**).

2.2.5 Adam

Another common technique for efficient optimization is adapting the step size for different parameters depending on the magnitude of the gradient. It is the case, for example, of *Adam*, the adaptive moment estimator [18]. It stores an exponentially decaying average of past squared gradients, ν_t , and gradients, m_t . Whereas momentum can be likened to a ball rolling down a slope, Adam behaves like a heavy ball with friction, which tends to prefer flat minima in the error surface. We compute the decaying averages m_t and ν_t as follows:

$$\begin{aligned} \mathbf{m}_t &= \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t \\ \nu_t &= \beta_2 \nu_{t-1} + (1 - \beta_2) \mathbf{g}_t^2 \end{aligned} \tag{2.31}$$

m_t and ν_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients, respectively; hence, the method's name. As they are initialized as vectors of zeros, they are biased towards zero, especially during the initial steps and when the decay rates are low (i.e., β_1

and β_2 are close to 1). These biases are countered by computing bias-corrected moment estimates:

$$\begin{aligned}\hat{\mathbf{m}}_t &= \frac{\mathbf{m}_t}{1 - \beta_1}, \\ \hat{\mathbf{v}}_t &= \frac{\mathbf{v}_t}{1 - \beta_2}\end{aligned}\tag{2.32}$$

The update rule is further balanced by a correction factor ϵ , obtaining:

$$\mathbf{x}_{t-1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{v}_t} + \epsilon} \hat{\mathbf{m}}_t\tag{2.33}$$

One advantage of Adam is that it is relatively robust to the choice of its hyperparameters. One disadvantage of accelerated optimization algorithms is the increased storage requirements. For example, momentum requires storing the previous gradient iteration in memory, thereby doubling the space needed by the optimization algorithm. Other, more sophisticated optimizers have been proposed in recent years. I suggest that curious readers check out AMSGrad [19] and AdamW [20].

2.2.6 Optimization Issues

In this subsection, I will explore some factors that hinder optimization. We will examine the challenges these problems present and the methods that can be used to tackle them. I will revisit these concepts later when we delve into a more comprehensive discussion of regularization techniques.

Vanishing and Exploding Gradient

Whenever we add a layer l to our model, we indirectly add a step to the gradient computation. In particular, for an activation function ϕ that boils down to just multiplying by the derivative of the function ϕ' computed for the input tensor \mathbf{x} :

$$\partial_{\mathbf{x}} h_l = h_l \cdot \phi'(\mathbf{x})$$

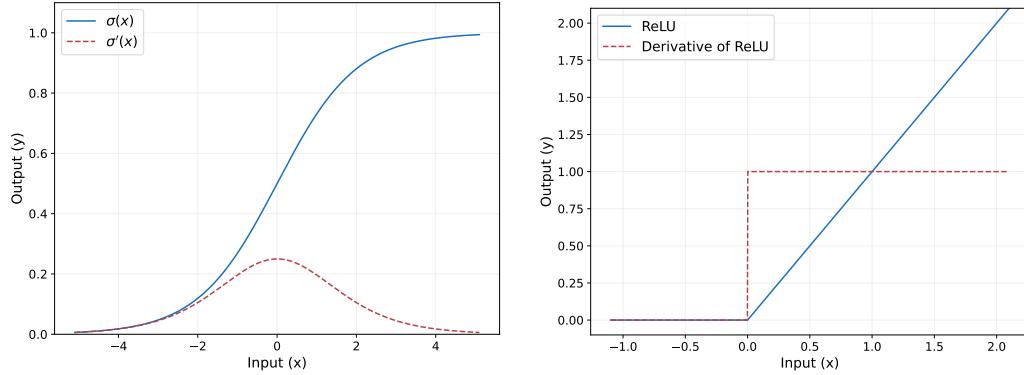


Fig. 2.12 (a) Plot of the sigmoid function (blue) and its derivative (red). (b) Plot of ReLU (blue) and its derivative (red).

which corresponds to scaling the gradient by a certain value depending on the type of activation. If $\phi'(\cdot) < 1 \forall x$ the gradient is progressively downscaled, while if $\phi'(\cdot) > 1 \forall x$ it is upscaled. The deeper the model, the more these effects (called *vanishing gradient* and *exploding gradient*, respectively) are impactful on training dynamics [21]. Indeed, in both cases, the optimizer's step size is highly unbalanced between early and deeper layers. In the case of vanishing gradients, the first layers of the model receive negligible updates because the gradient is close to 0.

For example, consider the sigmoid function $\sigma(\cdot)$. The image of $\sigma(\cdot)$ is bounded in $(0, 1)$, and its derivative can be computed as:

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \in (0, 0.25)$$

That means that the gradient backpropagating through $\sigma(\cdot)$ will, at best, be scaled by a factor of 4 (Fig. 2.12). Hence, it is highly subject to the vanishing gradient problem. On the contrary, an exponential function would easily manifest exploding gradients in deep networks due to its unbounded derivative. It is worth highlighting that vanishing and exploding gradients are problematic because computers use limited precision in their representation. They usually save real numbers as 32-bit floating-point values. Underflows and overflows can occur rapidly in this framework as the number of layers increases.

Choosing an activation function that never exhibits these problems is impossible. Indeed, only the identity I function satisfies $\phi'(\cdot) = 1 \forall x$. The best practice

is using a function similar to I but nonlinear enough to decouple linear layers and avoid collapsing. $ReLU$ is a good candidate for this scope since

$$\partial_x \text{ReLU}(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$

The gradient is either unscaled or zeroed, inducing computational sparsity (Fig. 2.12). The non-differentiable point 0 is never reached in practice by the optimizer when using non-zero *weight initialization*.

Weight Initialization

Gradient descent methods require a starting point in the space of possible weight values to begin the optimization process. Training deep models is a challenging task, and the choice of initialization has a significant influence on most algorithms. The initial point can determine whether the algorithm converges at all, with some initial points being so unstable that the algorithm encounters numerical difficulties and fails altogether [22]. Weight initialization is a procedure to set the optimal starting weights of a neural network.

Simply initializing all weights to zero poses problems for gradient vanishing and differentiability (especially when using $ReLU$) due to a phenomenon known as the *symmetry problem*. If all the network weights are initialized to zero, all the activations are zero, and correspondingly, so are the gradients. This symmetry is complex to break, as the network does not really learn anything. Another typical approach involves taking relatively small random values, for example, drawn from a Gaussian distribution $\mathcal{N}(0, \sigma^2)$. The variance, however, significantly affects both the outcome of the optimization procedure and the network's ability to generalize. Hence, more specific heuristics have been developed to estimate the optimal distribution scale.

In 2010, Glorot and Bengio [23] proposed to adopt an adequately scaled uniform distribution for initialization, called *Xavier initialization*. Its derivation is based on the assumption that the activations are differentiable; hence, it is invalid for $ReLU$. It is, though, the standard approach for sigmoid, tanh, and other similar activations (that are almost linear except for the saturation zones). Xavier initialization chooses the variance for the weight of a neuron w as the reciprocal

of the average number of connections entering and exiting from that neuron (n_{in} and n_{out}):

$$\sigma(w) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

This formulation arises from the need to mitigate vanishing and exploding gradients, where the weights are scaled based on the number of gradients flowing through a specific neuron.

If we release the assumption that the activation function is generally linear, we need a different initialization. In particular, *ReLU* has a non-zero mean and can have a dead gradient (outputting 0) for half its input space. He et al. [12] proposed to compensate for this unbalance by computing σ as:

$$\sigma(w) = \sqrt{\frac{2}{n_{in}}}$$

which proves beneficial for convergence and is, in practice, the default initialization in most frameworks.

Gradient Clipping

Another way to control gradient flow in backpropagation is *gradient clipping* [24]. It simply consists of using a threshold on the gradient and clipping values that exceed it (Fig. 2.13). There are two main approaches to gradient clipping, either focusing on the single elements of the gradient or the gradient norm [25].

In the following sections, and especially when we explore regularization, I will mention more methods that inherently control gradient flow. Indeed, the gradient magnitude is closely related to model complexity, and regularizing the learning process (i.e., biasing the model toward a more straightforward representation of the data) implicitly smooths the learning curve, leading to more regular gradients. As we will see, this approach is fundamental to scale models up to millions of parameters and beyond.

Now that we have uncovered neural networks' learning mechanism, we can explore more complex classes of models. The architectures I will introduce stem from different neuroscience observations and mimic some aspects of the human ability to process external stimuli.

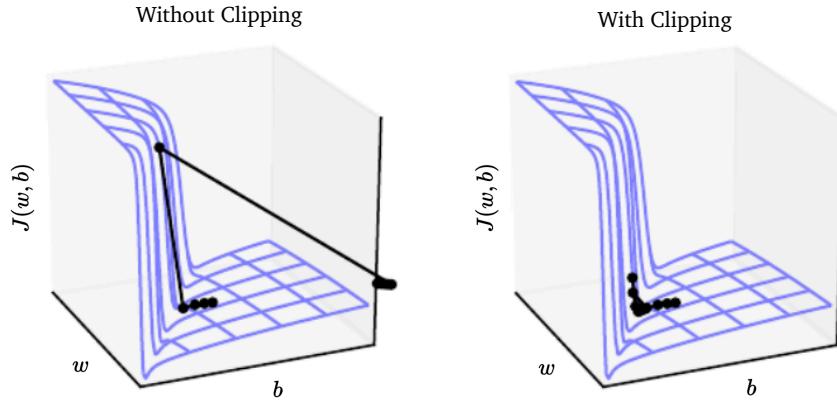


Fig. 2.13 Gradient clipping can make gradient descent perform more reasonably near steep cliffs. Gradient descent without gradient clipping overshoots the bottom of this small ravine, then receives a huge gradient from the cliff face. Instead, gradient descent with gradient clipping has a more moderate reaction to the cliff.

2.3 Convolutional Neural Networks

In the present section, I introduce the basic concepts behind convolution, the operation at the core of the computer vision revolution [26]. The development of convolutional layers in artificial neural networks is deeply rooted in our understanding of biological vision and the hierarchical organization of the visual cortex. Inspired by the pioneering work of neuroscientists like Hubel and Wiesel [27], who revealed how neurons in the early visual system detect edges and textures through progressive abstraction, convolutional neural networks (CNNs) mimic this biological blueprint.

In the brain, visual processing begins in areas like V1, where cells respond to simple stimuli (e.g., lines, orientations), and progresses through intermediate to higher cortical regions (inferotemporal cortex, V2 to V4) that integrate complex features like faces or objects (Fig. 2.14). Similarly, CNNs employ stacked convolutional layers to extract features in a hierarchical manner. Early layers capture basic patterns (edges, textures), while deeper layers synthesize semantic concepts (shapes, objects). However, the relationship is not a perfect mirror. While CNNs excel at replicating feedforward processing, biological vision incorporates feedback loops, parallel pathways, and dynamic attention mechanisms (features still underdeveloped in artificial models).

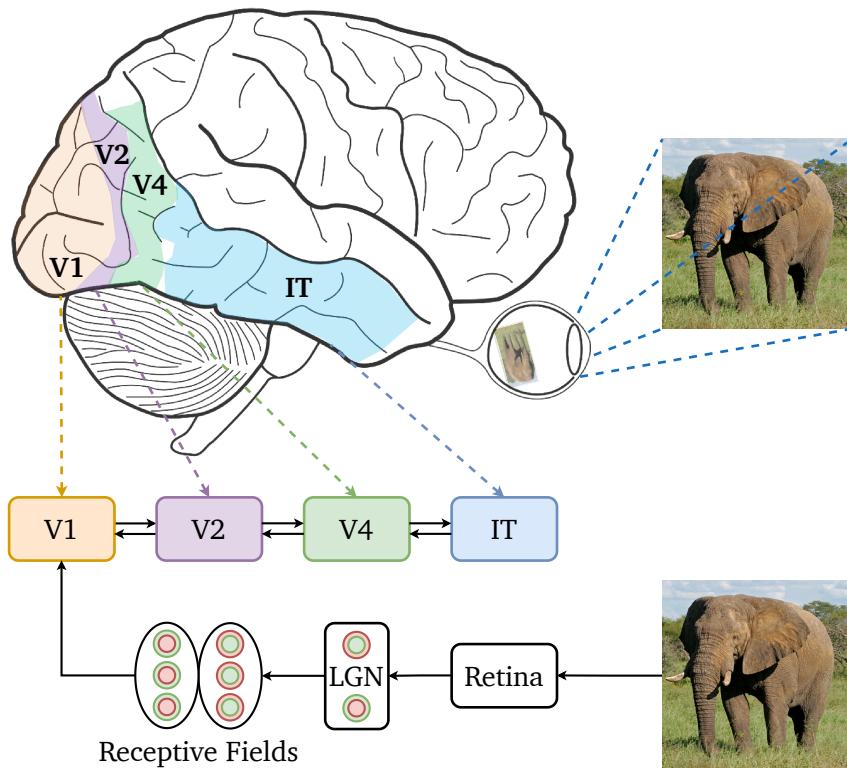


Fig. 2.14 Illustration of the areas associated with the primary visual cortex. Four Brodmann areas [28] are associated with the ventral visual stream. The figure also presents a block diagram illustrating several forward and backward projections between these areas. The visual signal passes through the retina and the lateral geniculate nucleus (LGN) before reaching V1's receptive fields.

Although designed for images, convolutions can be applied to any sequential data and have been progressively extended to many different media, such as audio and time series. I will start by defining the key properties of convolution and its differences from linear layers. Then, I will delve deep into some variants of convolutional layers and finally outline some best practices for building convolutional neural networks.

2.3.1 Limits of MLPs

Let's start with an example. We want to try to classify images from the MNIST handwritten digit dataset using an MLP. We already know MNIST contains 28×28 greyscale images spanning $n_c = 10$ classes. That implies input samples must be flattened, i.e., collapsed to a one-dimensional tensor of $n_i = 784$ elements. So,

the FC layer becomes:

$$\mathbf{h} = \phi(\mathbf{w} \cdot \text{flatten}(\mathbf{x})) \quad (2.34)$$

With a single hidden layer of $n_h = 500$ neuron units followed by the ReLU activation function, it is possible to achieve around 98% accuracy in a handful of epochs. The number of parameters in the model can be computed as:

$$(n_i + 1) n_h + (n_h + 1) n_c = 397,510 \quad (2.35)$$

where the added ones are due to the bias term. The high number of parameters is due to the characteristic of FC layers, where each neuron is connected with every other neuron in the adjacent layers. For example, if instead of 28×28 greyscale images, we use 224×224 RGB images (as in the ImageNet dataset [29]), the parameters become more than 75 million for a single-layer model.

This limitation makes it unfeasible to process high-dimensional inputs with MLPs. Moreover, images have an inherently sparse representation of reality, and MLPs are inefficient learners due to their redundancy. They are, instead, more suited for information-efficient representations, such as tabular data for handcrafted features. Another disadvantage of MLPs is that they disregard spatial information, as the output of every neuron is a weighted combination of all channels and pixels in the flattened input. We can obtain a more efficient solution by restricting computations through some *inductive biases* relative to sequential data. In this way, we limit the space of solutions and reduce parameter redundancy.

2.3.2 Convolutional Layer

Without loss of generality, I will now discuss the case of image data (2D convolutions), but the same logic can be applied to any sequential data. We already know an image can be described by a tensor $\mathbf{x} \sim (h, w, c)$, where h is the height of the image, w the width of the image, and c is the number of channels. However, these three dimensions are not identical since h and w represent a spatial arrangement of pixels, while the channels c do not have a specific ordering (in the sense that storing images in an RGB or a BGR format is only a matter of convention). I use

the same symbol (c) used for features in the tabular case because it plays a similar role in the design of the models. We can think of each pixel as being described by a generic set of c features, which are updated in parallel by the model's layers. To keep spatial consistency, we want layers to return a generic tensor $h \times w \times c'$ with an embedding of size c' for each of the $h \times w$ pixels. The two key ideas that constitute the inductive biases of convolution are *locality* and *translational equivariance*.

Locality

Locality stems from the fact that the spatial arrangement of pixels introduces a distance between the pixels. We define the distance between pixel (i, j) and (i', j') as the maximum distance across the two axes:

$$d((i, j), (i', j')) = \max(|i - i'|, |j - j'|) \quad (2.36)$$

Ideally, we can imagine that the influence of a pixel on another decreases with a factor inversely proportional to the distance between them. To take this idea to the extreme, we can assume that the influence is, in practice, zero for a distance higher than some threshold. To formalize this insight, I introduce the concept of *patch*.

Definition 2.3 (Patch)

Given an image \mathbf{x} , we define the patch $P_{i,j}^k$ as the sub-image centered at (i, j) and containing all pixels at distance $d \leq k$:

$$P_{i,j}^k = \mathbf{x}[i - k : i + k, j - k : j + k, :]$$

where d is defined as in Eq. (2.36).

Each patch is of shape (s, s, c) , where $s = 2k + 1$ since we consider k pixels in each direction along with the central pixel (Fig. 2.15). For reasons that will be clarified later on, we call s the *filter size* or *kernel size*.

If the output for a given pixel only depends on a patch of predetermined size, we say the layer is **local**.

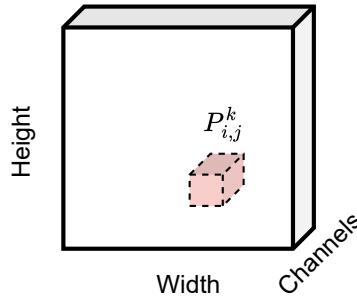


Fig. 2.15 Visualization of an image patch.

We can convert the layer defined in Eq. (2.34) into a local layer by setting to 0 all weights belonging to pixels outside the influence region (or *receptive field*) of each pixel:

$$\mathbf{h}_{i,j} = \phi(\mathbf{w}_{i,j} \cdot \text{flatten}(P_{i,j}^k)) \quad (2.37)$$

We obtained a *locally-connected* layer, where each output pixel has its own set of parameters $\mathbf{w}_{i,j} \sim (c', s, s, c)$. Compared to the original dense layer, we have reduced the number of parameters by a factor $\frac{s^2}{hw}$ (keep in mind that $s < \min(h, w)$ by construction). But what about the border output pixels? Their receptive field lies outside of the input image. For this reason, it is common practice to add the k rows and columns of zeros on each side of the input. This operation is called *zero-padding* (Fig. 2.16).

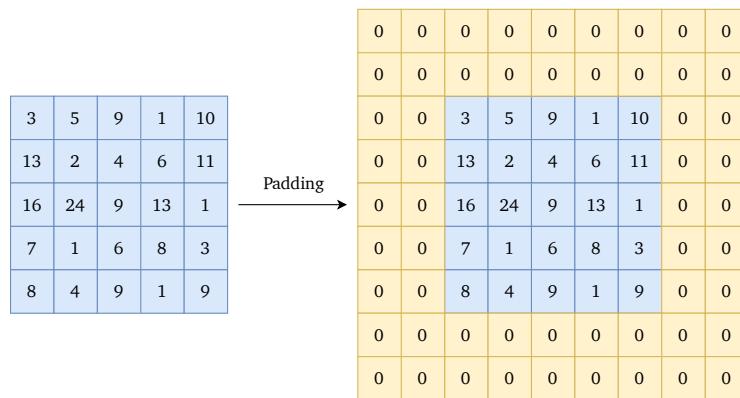


Fig. 2.16 Example of zero padding for $k = 2$.

Translational Equivariance

In the locally connected layer I just defined, two identical patches can yield different outputs based on their locations in the input tensor. That is because the two weight matrices $w_{i,j}$ and $w_{i',j'}$ differ. However, most of the time, we want our layer to transmit the same input pattern, regardless of where it appears specifically.

Definition 2.4 (Translational Equivariance)

We say that a layer $\mathbf{h} = f(\mathbf{x})$ is translation equivariant if translations of the inputs imply an equivalent translation of the output:

$$P_{i,j}^k = P_{i',j'}^k \text{ implies } f(P_{i,j}^k) = f(P_{i',j'}^k)$$

An example can be found in Fig. 2.17.

Equivariance and Invariance

Equivariance should not be confused with the concept of **invariance**. As depicted in Fig. 2.17, invariance refers to a situation where a change in the position of an object does not affect the nature of the output. Formally, we say that a function f is invariant to a class of functions \mathcal{G} if

$$f(g(x)) = f(x) \quad \forall g \in \mathcal{G}$$

Although they might sound contrasting, translation invariance and translation equivariance are not necessarily mutually exclusive. They can both occur in the same model, although under different contexts. Later in this section, we will see how translational invariance is enforced into convolutional models to learn specific tasks.

A simple way to achieve translation equivariance is given by *parameter sharing*, i.e., letting every position share the same set of weights:

$$\mathbf{h}_{i,j} = \phi(\mathbf{w} \cdot \text{flatten}(P_{i,j}^k)) \quad (2.38)$$

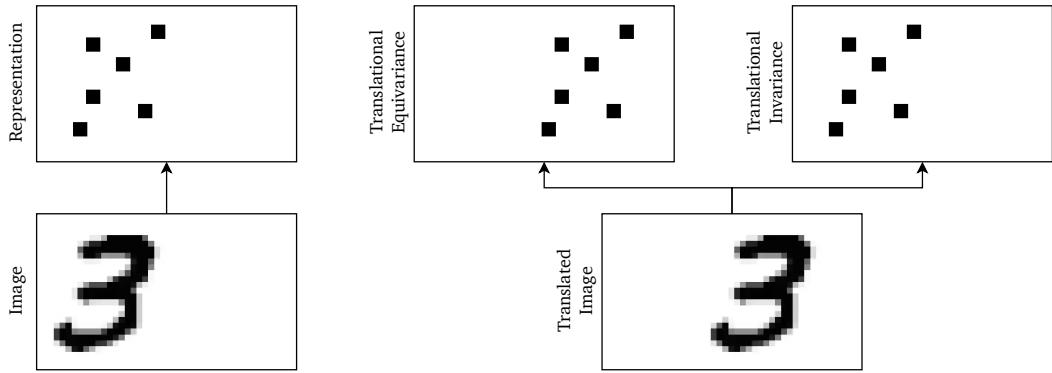


Fig. 2.17 Visual exemplification of the difference between translational invariance and equivariance.

This operator is called a convolutional layer, and it is incredibly parameter-efficient: we only have a single weight matrix $w \sim (c', s, s, c)$, which is independent of the resolution of the original image (once again, compare this with a layer which is only locally-connected with $h \times w \times s^2 \times c \times c'$ parameters for proof). We can write a variant with biases by adding c' additional parameters as a bias vector $b \sim (c')$, and obtain the complete convolutional layer formulation:

Definition 2.5 (Convolutional Layer)

Given an image $x \sim (h, w, c)$ and a kernel size $s = 2k + 1$, a convolutional layer $H = \text{Conv}(x)$ is defined element-wise by:

$$h_{i,j} = w \cdot \text{flatten}(P_{i,j}^k) + b \quad (2.39)$$

The trainable parameters are $w \sim (c', s, s, c)$ and $b \sim (c')$. The hyperparameters are k and c' (and eventually padding). The output has shape (h, w, c') if padding is applied and $(h - 2k, w - 2k, c')$ otherwise.

While other formulations of convolution are possible, I chose this one to clarify that a convolution remains a linear operation, albeit with a highly restricted weight matrix compared to a fully connected one.

From the point of view of signal processing, equation Eq. (2.39) corresponds to a filtering operation on the input signal x through a set of finite impulse response (FIR) filters [30], implemented via a discrete convolution. Each of the c' filters here corresponds to a slice of the weight matrix. In standard signal

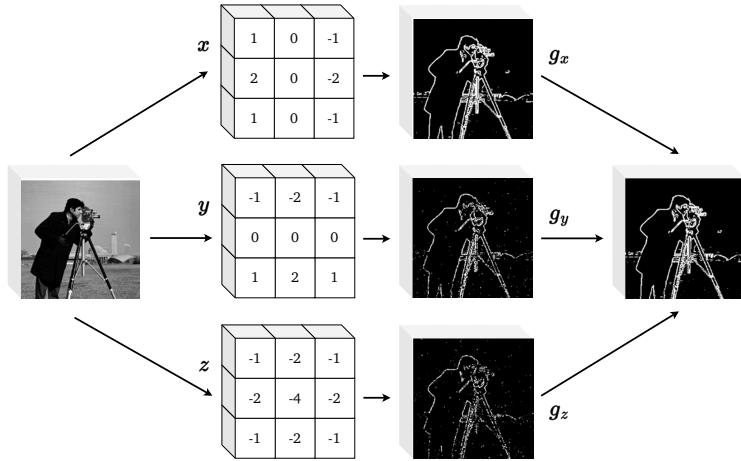


Fig. 2.18 Example of hand-crafted image filter: the Sobel operator.

processing, these filters can be manually designed to perform specific operations on the image, such as enhancing object contours (as shown in Fig. 2.18). In convolutional layers, instead, these filters can be randomly initialized and trained via gradient descent. Thanks to translational equivariance, the output maintains a certain spatial consistency, allowing it to be plotted. We call a slice $\mathbf{h}[:, :, i]$ of the output an activation map of the layer, representing how much the specific filter was “activated” on each input region. Now, it’s time to see how to stack multiple convolutional layers to build a model.

2.3.3 Pooling

Similarly to FC layers, simply stacking convolutional layers cannot extract complex patterns from data. Indeed, convolution is still a linear operation and suffers from the same collapse observed in Section 2.1. It can be proven that the concatenation of two convolutional layers is equivalent to a single layer with a wider kernel size (Fig. 2.19). The receptive field R of each output pixel grows linearly with the number of layers in the stack l :

$$R(i, j) = P_{i,j}^{lk}$$

Again, introducing nonlinearities between layers decouples the operations, al-

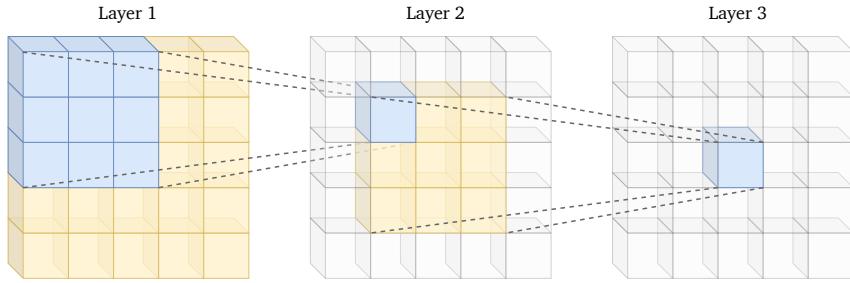


Fig. 2.19 The receptive field grows linearly with the number of layers. In this way, even if a single layer is limited in its receptive field by the kernel size, a sufficiently large stack of them results in a global receptive field.

lowing for the modeling of complex inputs.

$$f = \phi_l \circ Conv_l \circ \dots \circ \phi_1 \circ Conv_1$$

As seen in Fig. 2.19, the spatial dimension of the tensor remains constant while the number of channels varies depending on c' . However, reducing this dimensionality can be advantageous for certain tasks as we delve deeper into the model. That is the case, for example, of classification.

Let's consider an example: suppose we want to classify numbers, and our dataset contains instances in different areas of the frame, like in Fig. 2.17. If we aim to classify, we need one or more neurons to activate the number 3, regardless of its location in the image. The translation equivariance property of convolutional layers ensures that every feature present in the first image will be correspondingly visible in a translated region of the second image. Therefore, we need to progressively align these features to converge on a common spatial position, thereby achieving *invariance*. A simple way to accomplish this is *pooling*, which involves reducing the spatial dimensions and retaining only the highest input value. The decision to only forward the maximum value stems from the observation that patterns more relevant to the task at hand trigger higher activations in the model's hidden layers.

$$\mathbf{h}(\mathbf{x}) = \max_{i,j}(\mathbf{x})$$

This operation is called *global max pooling*, as the spatial dimensions are collapsed at once. An alternative version, *global average pooling* (GAP), is obtained by aver-

aging spatial features instead of selecting the maximum value. In practice, global pooling leads to a sudden bottleneck that loses too much spatial information. Instead, it is common to progressively reduce spatial dimensions through *max pooling*, which applies pooling on patches of size s :

Definition 2.6 (Max-pooling Layer)

Given a tensor $\mathbf{x} \sim (h \times w \times c)$, a max-pooling layer of kernel size k and stride s is defined element-wise as:

$$\text{MaxPool}(\mathbf{x})_{i,j,c} = \max(\mathbf{x}[i \cdot s : i \cdot s + k - 1, j \cdot s : j \cdot s + k - 1, c])$$

Note that the operation is performed independently for each channel. An example is depicted in Fig. 2.20.

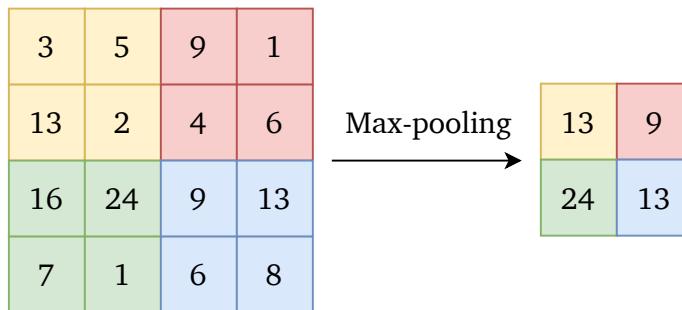


Fig. 2.20 Example of max-pooling with stride $s = 2$.

2.3.4 Convolutional Model

I can now formulate a basic convolutional block by combining a convolution, an activation function, and a pooling operation.

$$\text{ConvBlock}(\mathbf{x}) = (\text{MaxPool} \circ \phi \circ \text{Conv})(\mathbf{x})$$

This abstraction highly simplifies the design of complex CNNs. A simple image classification model could resemble the one in Fig. 2.21². After progressively shrinking spatial dimensions, a global average pooling (GAP) reduces the tensor

²Indeed, the first deep learning breakthrough is attributed to a model very similar to the reported one [26], the main difference being the use of flattening instead of GAP.

of shape (h, w, c) to a 1D tensor of shape (c) . The final FC layer guarantees the model outputs a value for each class.

This design exhibits several interesting properties. First, we can use the same training procedure described in the previous sections. For example, we can learn to classify MNIST samples by just applying *softmax* to the model’s output and back-propagating the cross-entropy loss. Second, the GAP operation outputs a fixed-shape tensor regardless of the image shape. This design choice allows feeding images with different resolutions without architectural changes. Moreover, from a design perspective, we can decouple the task to solve into two distinct goals:

- **Feature extraction:** projecting the high-dimensional input into a latent, lower-dimensional space. This space constitutes the internal representation of the data (called *embedding*) that the model implicitly builds by progressively learning the task. The feature space exhibits some interesting geometric properties, which I will detail in Section 2.6.1. The part of the model devoted to feature extraction is usually referred to as *backbone* or *encoder* and generally constitutes its most complex and computationally intensive part.
- **Prediction:** exploiting the learned embedding representation to solve a specific task. The part of the model devoted to prediction is usually referred to as *head* or *decoder* and is typically a straightforward function (i.e., an MLP). If the backbone is well-optimized, even a single linear layer is sufficient.

The terminology encoder/decoder originates from standard signal processing and denotes, among other things, the circuits that operate conversions between one-hot encoded digital signals and more compact representations [31]. The parallelism is quite fitting, as backbones effectively project high-dimensional and sparse data to a compact representation (the latent space), and heads reproject them back to a human-understandable one. Notably, models that output data in the same representation space as the input (like, for example, *autoencoders* [32]) assume an hourglass-like shape. I will elaborate on this concept in Section 2.6.1, demonstrating that a suitable backbone can be applied beyond its original purpose for which it was trained.

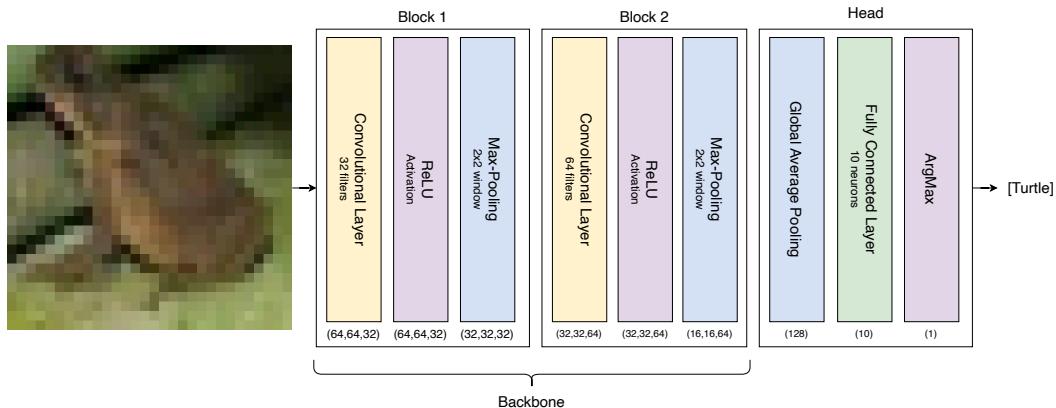


Fig. 2.21 A simple convolutional neural network (CNN) for image classification.

Note about batch dimension

As we have seen in the definition of mini-batch SGD, stacks of samples are processed in parallel during training. It means that, in general, models process inputs of shape:

$$\mathbf{x} \sim (b, [s_i], c)$$

where b is the batch size, $[s_i]$ is the set of spatial and temporal dimensions (e.g., for videos we have height, width, and the number of frames), and c is the number of channels. The ordering choice to put c last is conventional (some ML frameworks, like PyTorch, do the opposite). I will sometimes omit the batch dimension for simplicity.

2.3.5 Types of Convolution

So far, I have generically defined convolutional layers based on three primary hyperparameters: the kernel size k , the number of output channels c' , and the padding mode. It is worth noting that this definition can assume certain configurations that are particularly beneficial to learning under specific conditions. For example, choosing $k = 0$, we obtain the so-called 1×1 convolution. It essentially updates each pixel based solely on the corresponding input pixel.

$$\text{Conv}_{1 \times 1}(\mathbf{x})_{i,j,z} = \sum_{t=1}^c \mathbf{w}_{z,t} \mathbf{x}_{i,j,t}$$

where $z \in [0, c']$ is any of the channel outputs and $\mathbf{W} \sim (c, c')$. This operation is useful when modifying the channel dimension while keeping the spatial relations unchanged. That is equivalent to applying a dense layer to each pixel independently.

A dual version of the convolutional layer consists of recombining spatial information without mixing channels:

$$\text{Conv}_D(\mathbf{x})_{i,j,c} = \sum_{i'=1}^{2k+1} \sum_{j'=1}^{2k+1} \mathbf{w}_{i',j',c} \mathbf{x}_{i'+i-k-1, j'+j-k-1, c}$$

where $\mathbf{w} \sim (s, s, c)$ and each output channel only depends on the corresponding input channel. We call this operation *depthwise convolution* (Conv_D). If we process groups of channels together instead of single channels, we obtain *groupwise convolution* (Conv_G).

Alternating a $\text{Conv}_{1 \times 1}$ and a Conv_D in a model, we decouple channel and spatial processing, obtaining a *depthwise separable convolution*. The great advantage of this approach is that the number of parameters is reduced from $(s \times s \times c \times c')$ to $(s \times s \times c + c \times c')$. For this reason, depthwise separable convolutions are highly used for low-power models.

Note that all I have said about convolutions on images (generally referred to as 2D convolutions) can be extended to 1D data (e.g., audio) and 3D data (e.g., video). The only difference is in the shape of the kernel and the dimensions along which it is convolved (Fig. 2.22).

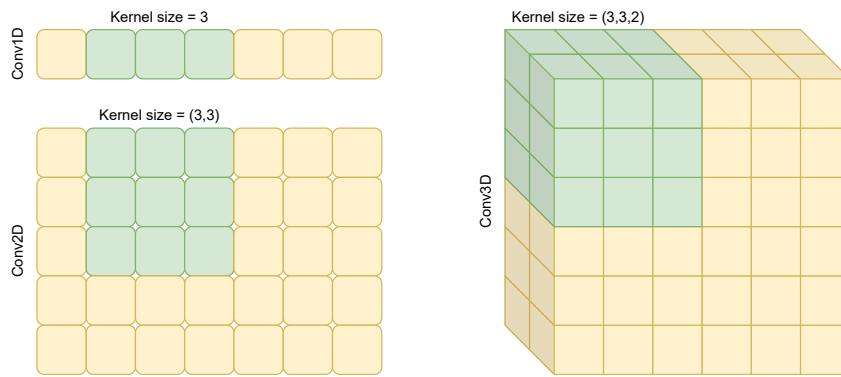


Fig. 2.22 Comparison between 1D, 2D, and 3D convolutional kernels. For simplicity, I only represent a single channel.

Now that we understand how CNNs work, we can move toward the modern deep learning era, where scale is the key to solving increasingly complex tasks.

2.4 Regularization and Scaling

Following the deep learning revolution, pioneered by open-ended competitions such as the *ImageNet Large Scale Visual Recognition Challenge* [33], model design has increasingly relied on a minimal set of guiding principles and components. In contrast, the focus of research has shifted to scaling them up to their limits. Starting with AlexNet’s eight layers and 60 million parameters [26], a wide range of techniques has been developed to stabilize the training of very large models.

The power of scaling is embodied in the relatively recent concept of neural scaling laws, which have been instrumental in driving massive investments in artificial intelligence. For practically any task, increasing data, compute power, and model size results in a predictable increase in accuracy. The tremendous power of combining simple, general-purpose tools with exponentially increased computational power was called *the bitter lesson of AI* by Richard S. Sutton in 2019 [34].

However, scaling up is non-trivial, as it encounters several problems, including slow optimization, gradient issues, and numerical instability. In this section, I delve into the issue that arises when attempting to solve a task with higher precision by training larger models. Then, I explore techniques that have enabled the exploitation of exponentially growing computational power and data over the last decade to train increasingly larger models. Some were well-known before the advent of deep learning, while others were discovered in the last decade.

2.4.1 Overfitting

As already described in Section 1.2.2, every machine learning model aims to minimize the expected risk given by its training dataset. However, our real goal is to reduce the *empirical error* on some future or unknown dataset, of which the training set is only a proxy. We assume that these two datasets have a similar distribution, but many factors can create shifts that alter this assumption

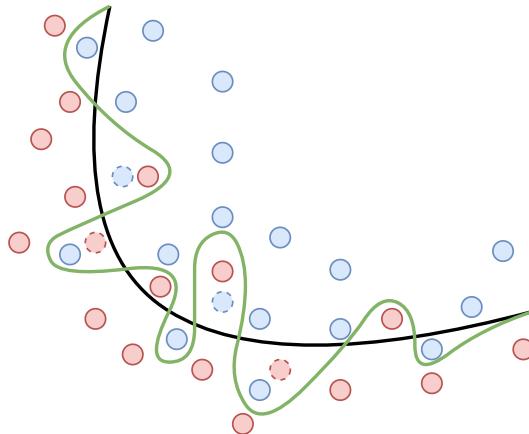


Fig. 2.23 A simple representation of data overfitting in the case of a binary classifier. The green line represents an overfitted model, and the black line represents a simpler model. While the green line best follows the training data, it is too dependent on it and is likely to have a higher generalization error (illustrated by dashed dots) than the black line. We already observed a similar example in Section 1.3.1.

(I will revisit this in Section 3.2.1). This discrepancy leads to an unavoidable *generalization error*.

When we increase the model size and, consequently, the number of parameters, we give it the capability of representing more complex correlations (recall the example in Section 1.3.1). This further minimizes the expected error but increases the likelihood that subtle patterns may not be detected in the test dataset. When that happens, it means that the model is “memorizing” the training data, or *overfitting*. The more a model minimizes the expected error, the more closely it will correspond to the training data and may fail to fit additional data reliably. In practice, overfitting is apparent when the dynamics of training and validation errors diverge, i.e., when the validation loss steadily increases, even as the training loss continues to decrease. We say the model is too complex for the task or over-parametrized (Fig. 2.23). Recent discoveries in training massive models have observed the disappearance of overfitting when training datasets are so immense as to empirically bridge the gap with testing scenarios³.

When we try solving a task with a large model, we often frame an ill-posed problem. A high number of parameters leads to a very complex loss landscape, which in turn brings optimization pitfalls and numerical instability. In this context,

³Ben Recht - "Thou Shall Not Overfit" (2025)

scaling is a thoughtful balance between a larger representational space and a simplified model that guarantees optimal solutions are reachable. *Regularization* plays the critical role of converting or steering the answer to an optimization problem to a simpler one. Moreover, it counters overfitting, hindering the model from directly memorizing the data (and its associated noise) instead of the patterns behind it. I will now explore several regularization techniques that serve both these purposes.

2.4.2 Weight Regularization

Regularizing weights is one of the easiest ways to discourage overly complicated representations. Applying *Occam’s razor*, we want to bias the model by adding a loss component, which is a function of the magnitude of its weights:

$$L_{reg} = L(\mathbf{w}, \mathbf{x}) + \lambda R(\mathbf{w})$$

The regularization term is weighted by the hyperparameter $\lambda \geq 0$, ensuring that the model weights do not assume huge values. Usually, R is a norm function: we talk about Ridge (l_2) regularization when $R(\mathbf{w}) = \|\mathbf{w}\|_2^2 = \sum_i w_i^2$ and Lasso (l_1) regression when $R(\mathbf{w}) = |\mathbf{w}|_1 = \sum_i |w_i|$. In general, the two can be combined to create the so-called *elastic-net* regularization [35]. Favoring solutions with a lower weight magnitude corresponds to “less abrupt” changes in the output for a small deviation in the input and, for standard SGD, coincides with decaying weight values as training proceeds. However, that does not hold for more sophisticated optimizers like Adam, in which the gradient is rescaled. To address this issue, [20] proposed *decoupled weight decay*, which improves optimization stability and reduces the magnitude of weights independently.

2.4.3 Early Stopping

We have already seen that overfitting causes an increase in the generalization gap as the model begins to learn spurious correlations in the training data that are absent in the test set. In the training dynamics, the point at which overfitting begins can be identified by tracking training and validation metrics until the latter start diverging. That is a sign that the optimization has gone too far, and

stopping is the best course of action. *Early stopping*, with a reasonable *patience* window to account for oscillations, avoids useless computation in these cases. It is particularly advantageous on small datasets in which overfitting regimes are typical.

2.4.4 Data Augmentation

In general, the most effective way to improve model performance is to increase the dataset size. Indeed, more samples imply a broader coverage of data distribution, a smaller generalization gap, and smoother optimization. However, labeled data is seldom easy to obtain without a discrete investment of time and money (though recent advancements in synthetic data generation have alleviated this problem [36]).

A common practice to virtually increase the number of available samples for a task is *data augmentation*, which involves applying functions that transform the data without altering its semantics. The functions belong to a set of classes defined according to the task at hand, which can be combined to achieve a specific goal. Data augmentation is always associated with a random component to maximize the transformations of the same image. Consider, for example, augmenting data by adding Gaussian noise to the input:

$$\mathbf{x}' = \mathbf{x} + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim N(0, \sigma^2)$$

This approach can generate a deliberate number of new samples in the neighborhood of \mathbf{x} . The more σ grows, the more samples deviate from the real data; therefore, the amount of shift we choose is paramount to ensure that data augmentation is beneficial for learning.

For the specific case of images, several transformations exist that heavily modify the data without altering its semantic content. For example, brightness and contrast changes, zoom, and rotation are parametric functions $t(\cdot)$ where the parameter is a random variable changing at every function call. Rotation, for example, can be parametrized by a uniform random angle in the range $[-20^\circ, 20^\circ]$. More sophisticated transformations involve multiple samples. MixUp [37], for example, interpolates a couple of samples using a convex combination. CutOut [38] and CutMix [39], instead, use patch-wise cut and paste (Fig. 2.24).

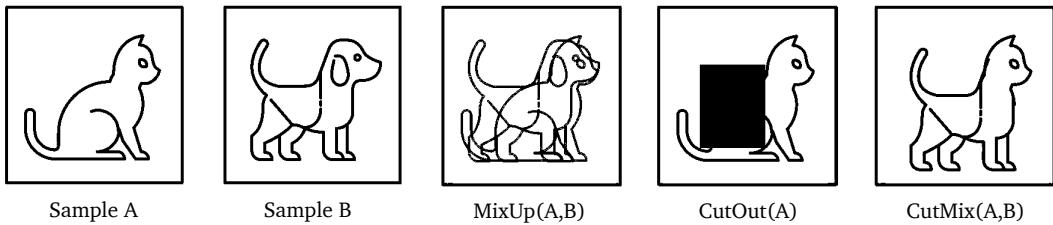


Fig. 2.24 Example of sophisticated data augmentation transformations.

Data augmentation is applied after loading the samples in memory so that the storage footprint of the dataset is unaltered. Choosing which transformations to use and how to compose them is critical and should be carefully considered depending on the task at hand. Some automated frameworks optimize data augmentation search, like *RandAugment* [40]. Additionally, data augmentation can prevent overfitting by avoiding the repetition of the same input multiple times.

2.4.5 Dropout

Contrary to previous methods, *dropout* [41] is not a training algorithm or data loading component. Instead, it can be viewed as an additional layer in the model, serving a similar goal to data augmentation. Indeed, dropout forces the model to learn in a more complicated setup by randomly masking out some neurons during training.

Definition 2.7 (Dropout layer)

Given a batch $x \sim (n, c)$ of internal activations, a dropout layer first samples a binary mask $m \sim (n, c)$, where each element is drawn from a Bernoulli distribution of probability p

$$M_{i,j} = \text{Bern}(p)$$

Hence, each element of m is 1 with probability p and 0 with probability $(1 - p)$. The dropout layer partially masks the input:

$$\text{Dropout}(x) = m \cdot x$$

The layer is not trainable and has only one hyperparameter, p .

For every mini-batch, a random set of neurons will be cut. This approach reduces the model's dependence on any specific feature and leads to redundant, and hence, robust layers.

2.4.6 Normalization Layers

As we have seen in Section 1.2.3, ensuring all features share the same range and statistics benefits optimization and numerical stability. Normalization layers translate this idea to intermediate features. Since internal activations change after each training step, so do their statistics.

Batch Normalization (BN) [42] tracks them by updating running channel-wise mean μ and variance σ^2 estimates with each mini-batch.

$$\begin{aligned}\mu_j &= \frac{1}{n} \sum_i x_{i,j} \\ \sigma_j^2 &= \frac{1}{n} \sum_i (x_{i,j} - \mu_j)^2\end{aligned}\tag{2.40}$$

At each forward step, the layer normalizes inputs to have zero mean and unit variance:

$$x' = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\tag{2.41}$$

where ϵ is a small positive term added to avoid dividing by zero. Since the Gaussian distribution is just a convention, we can instead give the model the ability to choose the optimal μ and σ^2 at the cost of additional training parameters. We obtain the full formulation of the *batch normalization* layer:

Definition 2.8 (Batch Normalization)

Given an input $x \sim (n, c)$, a batch normalization layer applies the normalization

$$BN(x) = \alpha \left(\frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta$$

where $\alpha \sim (c)$ and $\beta \sim (c)$ are trainable parameter arrays and μ and σ^2 are computed according to Eq. (2.40). At inference time, μ and σ^2 are frozen using a global exponential moving average to guarantee deterministic outputs.

BN is usually placed between linear and non-linear layers to intuitively rescale processed outputs before activation functions. For instance, this trick better exploits ReLU's negative quadrant. Subsequent studies have found that using *BN* makes it possible to train with larger learning rates, thanks to its more stable gradients [43]. For higher-dimensional data, *BN* is always applied channel-wise, supposing that features follow the same distribution along spatial dimensions.

BN also has some drawbacks. For example, the estimated variance can be quite large for small mini-batches, which can hinder training dynamics. Moreover, replacing μ and σ^2 with their static versions after training creates a slight mismatch in predictions that may compromise the model's accuracy. Some variants have been proposed to solve these issues by modifying the normalized input axis.

Layer Normalization (LN) [44], for example, computes the mean and variance for each input independently, i.e., along its rows.

$$\mu_i = \frac{1}{c} \sum_j x_{j,i} \quad (2.42)$$

$$\sigma_i^2 = \frac{1}{c} \sum_j (x_{j,i} - \mu_i)^2 \quad (2.43)$$

For high-dimensional inputs, such as images and videos, one can consider all spatial locations together or treat them separately (see the different colors denoting the two variants in Fig. 2.25).

Group Normalization (GN) [45], restricts the layer normalization operation to a subset of channels. Its main merit is that it allows for tiny batch sizes without the variance divergence problem we found in batch normalization. A particular case is **Instance Normalization** (IN) [46], in which each channel is given its independent group. All the normalization layers are schematized in Fig. 2.25.

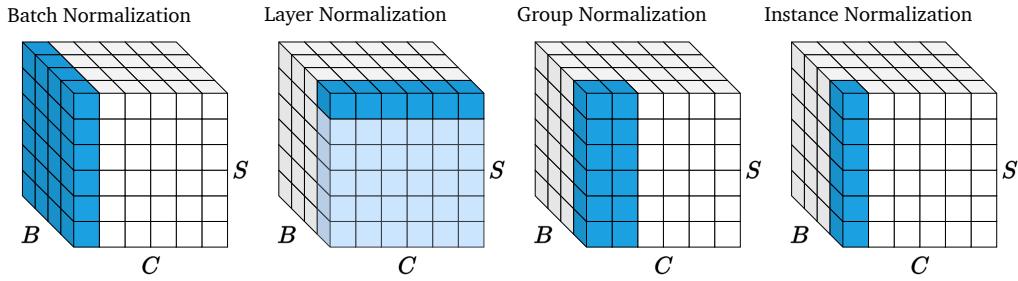


Fig. 2.25 Graphical depiction of normalization layer variants: Batch Normalization (BN), Layer Normalization (LN), Group Normalization (GN), and Instance Normalization (IN).

2.4.7 Residual Connection

So far, all the regularization methods we have seen permit increased model complexity by adding layers that extract deeper patterns from data. However, an upper bound was observed, after which the training error of models tended to recede (Fig. 2.26).

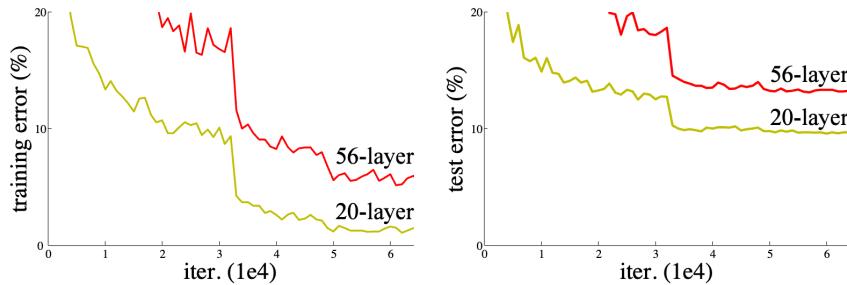


Fig. 2.26 Training error (left) and test error (right) on the CIFAR-10 dataset [47] with 20-layer and 56-layer networks. The deeper network has a higher training error, and thus a higher test error [48].

This phenomenon seems counterintuitive since a deeper network should at least maintain the modeling ability of a shallower one. Considering, for example, a two-layer model that is a submodule of a deeper one:

$$\begin{aligned} g_1(\mathbf{x}) &= (f_3 \circ f_1)(\mathbf{x}) \\ g_2(\mathbf{x}) &= (f_3 \circ f_2 \circ f_1)(\mathbf{x}) \end{aligned} \tag{2.44}$$

Intuitively, g_2 should not perform worse than g_1 since, in the worst-case scenario, f_2 can always approximate the identity function and collapse to the simpler $g_2 \approx g_1$. However, layers do not naturally lean toward the identity function as

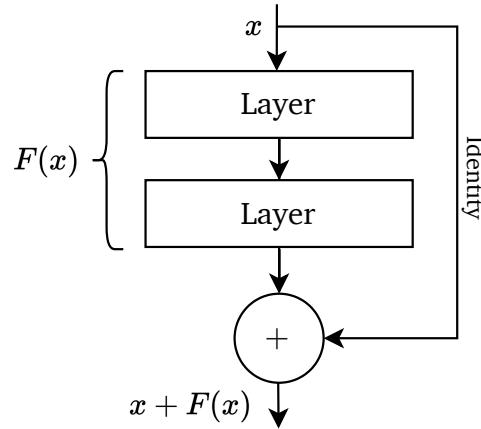


Fig. 2.27 General structure of a residual block.

their initial weights are centered around zero. We can improve that by explicitly biasing the layer towards such simplification and rewriting it as a *residual* (or *skip*) *connection*:

$$f_2(x) = x + F(x) \quad (2.45)$$

This way, the model learns the optimal deviation from the identity function, rather than the deviation from zero. This simple trick unlocks the possibility of increasing the model depth to hundreds of layers without losing representation ability. These models are typically referred to as *residual networks* (ResNets). The benefit of a residual block is also important during backpropagation, as the unmodified back-propagated gradient helps mitigate gradient instabilities. Moreover, it works well with batch normalization, which biases the layers towards the identity function in early training steps. A typical residual convolutional block (represented in Fig. 2.27) is the following:

$$h = (BN \circ Conv2D \circ ReLU \circ BN \circ Conv2D)(x) + x$$

Many alternatives and reworks of ResNets have been proposed in subsequent years. Most notably, *ResNeXt* [49] achieves even better scalability through the use of *bottleneck* layers with a lower number of filters at the end of residual blocks. We will see that the same structure has been used for the later introduction of transformer blocks (Section 2.5).

2.5 Transformers

As the previous section demonstrates, convolutional models serve as strong baselines and have achieved unprecedented performance on many tasks, particularly for images and time series where local relations prevail. Still, they are limited in handling very long sequences or non-local dependencies between elements. Indeed, sequences like text and time series can be very long. That is also the case for high-resolution images, where we may need to correlate small visual details that are far apart from each other. In this case, the locality of convolutional layers can be a drawback because we need a linearly increasing number of layers to process larger and larger receptive fields. We can, hence, say that CNNs have a *scaling limit*. This chapter demonstrates that alternative models can be designed to address this problem.

2.5.1 Attention

The first breakthrough away from the convolutional paradigm occurred in 2016 with the popularization of the *transformer* [50], an architecture for natural language processing that efficiently handles long-range correlations. In particular, it had a decisive advantage in scaling when trained on massive datasets, incentivizing people to adapt it to all data types and tasks. The excellent generalization ability of transformers makes them, to this date, the state-of-the-art approach in many fields [51].

The core element of the transformer is *attention* (hence the name of the proposing paper “*Attention is all you need*” [50]). But what is attention? Let’s start with an example and consider these two sentences:

The book is on the table.

The book my brother is looking for is on the table.

As detailed in Section 1.1.1, we need to tokenize the two sentences and compute their embeddings to feed them into a machine learning model. Suppose we obtain the tensors $\mathbf{x}_1 \sim (n_1, e)$ and $\mathbf{x}_2 \sim (n_2, e)$, where n_i is the number of tokens in sentence i and e is the embedding dimension. From a semantic point of view, the relationship between *table* and *book* is the same for both sentences. However,

their distance in the sequence varies and can generally become arbitrarily large. We refer to these as *long-range* dependencies. Moreover, our brain recognizes the same relationship between the two words, disregarding their offset and focusing on the words themselves (*input-conditional* dependencies).

If we apply a 1D convolution to \mathbf{x}_1 and \mathbf{x}_2 , the weight matrix only depends on the offset between tokens and goes as far as the receptive field allows. Modeling long-range dependencies with convolutions necessitates an increase in the kernel size of the layer, resulting in a linear increase in the number of parameters. Moreover, even with larger receptive fields, the weight given to a token depends only on its offset from the center of the kernel.

To solve this problem, we need to apply weights based on the content of the tokens rather than their offset. We can formulate it in the following way:

$$\mathbf{h}_i = \sum_{j=1}^n g(\mathbf{x}_i, \mathbf{x}_j) \mathbf{x}_j \quad (2.46)$$

where the summation on all tokens indicates a global receptive field and $g(\cdot)$ is the function that computes the input-conditional weight for $(\mathbf{x}_i, \mathbf{x}_j)$. Layers of this class are generally called *non-local* (Fig. 2.28).

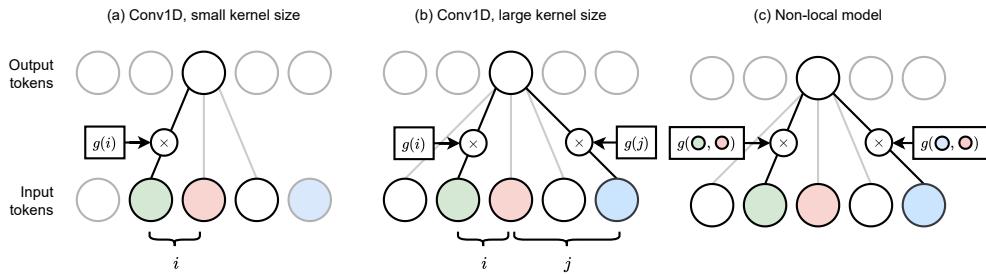


Fig. 2.28 A comparison between 1D convolutions with different receptive fields and a generic non-local network.

The multi-head attention (MHA) layer proposed by [50] is a non-local layer with scalar weights. In particular, MHA measures the similarity between tokens using their dot product:

$$g(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \in \mathbb{R} \quad (2.47)$$

Assuming the input tokens have variance σ^2 , this value becomes σ^4 for their dot product. For this reason, the result is scaled by the square root of the embedding size e .

$$g(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\sqrt{e}} \mathbf{x}_i \cdot \mathbf{x}_j \in \mathbb{R} \quad (2.48)$$

Furthermore, since we are processing a variable number of tokens, it is also helpful to normalize the temporal dimension. A common choice is using softmax, obtaining:

$$\mathbf{h}_i = \sum_{j=1}^n \text{softmax}(g(\mathbf{x}_i, \mathbf{x}_j)) \mathbf{x}_j \quad (2.49)$$

We call $g(\cdot)$ the *attention scoring* function and the normalized outputs the *attention scores*. The intuition behind this name is that each token has limited attention to divide among other tokens. Due to the softmax function, increasing attention toward a specific token will inevitably reduce it for others.

So far, g does not have trainable parameters because it is a simple dot product operation. A trainable projection is added to the input before applying g to give the MHA layer proper representation capability. The input is projected three times through linear layers, producing key, value, and query tokens.

$$\begin{aligned} \text{Key tokens: } \mathbf{k}_i &= \mathbf{W}_k \mathbf{x}_i \sim (n, n_k) \\ \text{Value tokens: } \mathbf{v}_i &= \mathbf{W}_v \mathbf{x}_i \sim (n, n_v) \\ \text{Query tokens: } \mathbf{q}_i &= \mathbf{W}_q \mathbf{x}_i \sim (n, n_k) \end{aligned} \quad (2.50)$$

The weight matrices $\mathbf{W}_k \sim (n_k, e)$, $\mathbf{W}_v \sim (n_v, e)$, $\mathbf{W}_q \sim (n_k, e)$ are learnable. Hence, we obtain $3n$ tokens that we can plug in Eq. (2.49), obtaining the *self-attention* (SA) layer:

$$\mathbf{h}_i = \sum_{j=1}^n \text{softmax}(g(\mathbf{q}_i, \mathbf{k}_j)) \mathbf{v}_j \quad (2.51)$$

We compute the updated representation of token i by comparing its query to all possible keys, and we use the normalized weights to combine the corresponding

value tokens. Note that the dimensionality of keys and queries must be identical, while the dimensionality of the values can be different.

The advantage of the dot product is that we can stack keys, values, and queries and compactly process all the tokens at once:

$$\begin{bmatrix} k \\ v \\ q \end{bmatrix} = \mathbf{x} \begin{bmatrix} W_k \\ W_v \\ W_q \end{bmatrix} \quad (2.52)$$

The SA layer can then be written as:

$$\text{SA}(\mathbf{x}) = \text{softmax}\left(\frac{\mathbf{q} \cdot \mathbf{k}}{\sqrt{n_k}}\right)\mathbf{v} = \text{softmax}\left(\frac{\mathbf{x}W_q \cdot \mathbf{x}W_k}{\sqrt{n_k}}\right)\mathbf{x}W_v \quad (2.53)$$

Since the input has shape (n, e) , the SA layer has $2ke + ve$ parameters independent of the sequence length n . The involved operations are summarized in Fig. 2.29.

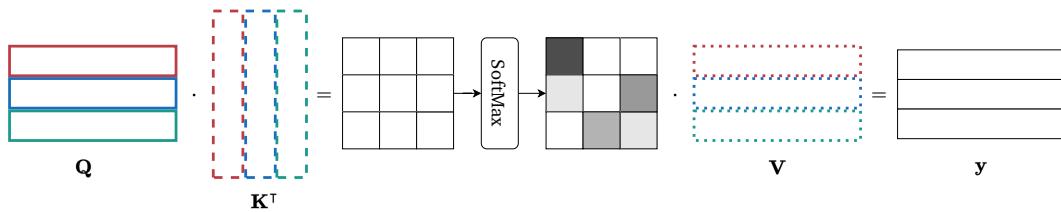


Fig. 2.29 Basic schematization of the self-attention layer.

Note about nomenclature

The names “keys”, “values”, and “queries” originate from databases. The most straightforward data structures are collections of key-value pairs. For example, Python dictionaries are sets of $(key,value)$ pairs, where each *key* is a univocal ID to retrieve the corresponding *value*. Hence, to retrieve the *value* we need to *query* the dictionary for the exact *key*:

$$o = v \Leftrightarrow q = k \quad (2.54)$$

For example:

```
d = dict()
d["Simone"] = 2
d["Simone"]      # Returns 2
d["Smone "]     # Returns an error
```

Once we define a similarity measure over a (q,k) pair, we can consider the SA layer as a soft database in which each token is updated using a weight combination of all other tokens based on their similarity. As we use *softmax*, the most present value v on the output will be the one maximizing (q,k) similarity.

The SA layer allows the modeling of pairwise correlations across tokens with high flexibility. However, sometimes, we can have multiple overlapping relationships to model. Let’s think again of our example Section 2.5.1. The relationship between *book* and *table* is different from the one between *book* and *brother*. In general, we want to have the power to decouple different dependencies to ease model learning. A straightforward way is to use multiple parallel SA layers and then aggregate their outputs. We instantiate h layers, which we refer to as **heads**, and assign each one its own set of parameters and projections. The *multi-head attention* (MHA) layer stacks the outputs of the heads and projects them to the desired embedding space:

$$\text{MHA}(\mathbf{x}) = [\text{SA}_1(\mathbf{x}) \parallel \dots \parallel \text{SA}_h(\mathbf{x})] \mathbf{w}_o \quad (2.55)$$

where $w_o \sim (o, hv)$ are the weights of the linear output projector. Each SA layer returns a tensor of shape (n, v) , so the output has shape (n, o) (o can be chosen to have the desired output dimensionality).

2.5.2 Positional Embedding

We need an additional component to formulate a complete transformer model. Imagine applying a permutation to the input of an MHA layer. Interestingly, we can prove that the only effect of this operation is to rearrange the layer's outputs equivalently, i.e.,

$$\text{MHA}(p(\mathbf{x})) = p(\text{MHA}(\mathbf{x})) \quad (2.56)$$

where p is the permutation operator. Hence, the operation is permutation equivariant for the entire input.

Translational equivariance was a desirable property for a convolutional layer, but permutation equivariance is undesirable (at least for text) because it discards the valuable ordering of the input sequence. Indeed, the SA and MHA layers are set functions, not sequence functions, and do not inherently value token ordering. However, we know spatiality is paramount in most data modalities (e.g., word order in text).

Instead of modifying the layer by adding a new operation, transformers introduce *positional embedding*. This auxiliary component depends only on the position of tokens in the sequence. Specifically, absolute positional embeddings depend on the absolute token position, while relative positional embeddings depend on the distance between two tokens.

If $\mathbf{x} \sim (n, e)$ represents the content of the input tokens, defining a positional encoding $\mathbf{s} \sim (m, e)$ means representing the absolute position of each token for a fixed maximum number of tokens m . The absolute positional encoding is summed to \mathbf{x} obtaining:

$$\mathbf{x}' = \mathbf{x} + \mathbf{s}[:n] \quad (2.57)$$

In this way, we break permutational equivariance and incorporate positional information into each token.

The most straightforward strategy is to consider s as part of the model's parameters and train it. This approach works well in practice where n is relatively stable in the dataset (e.g., in computer vision). Otherwise, other deterministic approaches, such as an increasing scalar proportional to the absolute position, can be used. However, a single scalar value is not very effective in practice. An alternative may be to use binary encoding, generating a tensor for each position in the following way:

0:	0	0	0	0	8:	1	0	0	0
1:	0	0	0	1	9:	1	0	0	1
2:	0	0	1	0	10:	1	0	1	0
3:	0	0	1	1	11:	1	0	1	1
4:	0	1	0	0	12:	1	1	0	0
5:	0	1	0	1	13:	1	1	0	1
6:	0	1	1	0	14:	1	1	1	0
7:	0	1	1	1	15:	1	1	1	1

We can observe that the bits change value at different rates, similar to a series of square waves of varying frequencies. Since we use floating-point representation for our computations, we can replace this integer representation with its continuous counterpart: a sinusoidal function. We can encode each position i through a set of sine and cosine functions with frequencies ω that increase continuously:

$$\mathbf{s}_i = [\sin(\omega_1 i), \cos(\omega_1 i), \dots, \sin(\omega_{e/2} i), \cos(\omega_{e/2} i)], \quad (2.58)$$

In this way, each absolute position is assigned an embedding $\mathbf{s}_i \sim (e)$ through the tensor ω spanning increasing frequencies. We can, hence, represent many absolute positions efficiently (Fig. 2.30). Moreover, this representation allows the model to easily learn to attend by relative positions since for any fixed offset k , \mathbf{s}_{i+k} can be represented as a linear function of \mathbf{s}_i ⁴.

We force the dimensionality of \mathbf{s} to match that of \mathbf{x} to allow simple addition. That is preferable to concatenation, as it lowers the number of parameters and computations. For ω , a geometric progression is commonly used, i.e., $\omega_i = \frac{1}{10000^{i/e}}$ so that $\omega_0 = 1$ and $\omega_e = 10^{-4}$. This approach, known as *sinusoidal positional encoding*, was proposed in [50] and is currently the most popular. Another popular recent alternative is the *rotary position embedding* [52].

⁴A detailed proof is available in [this blog post](#).

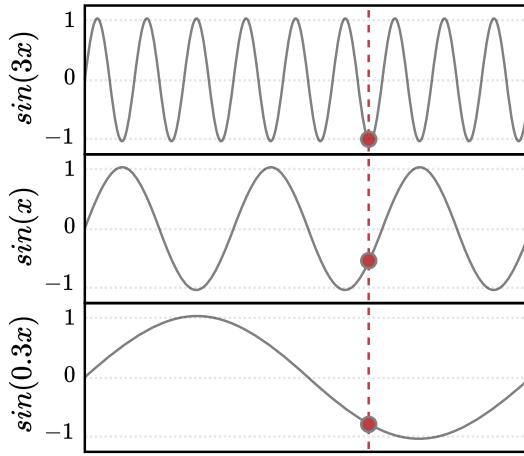


Fig. 2.30 Given a token position x , Positional Embedding returns an array of values taken from sinusoids at increasing frequencies.

2.5.3 Transformer Model

To build a complete model optimally from MHA layers, we must follow specific shared design patterns. First, we must interleave attention layers with nonlinearities (such as softmax) to prevent collapsing. Second, MHA layers should be alternated with separate MLP blocks operating independently on each token. This procedure resembles depth-wise separable convolutions (Section 2.3.5), as it separately processes spatial information (the tokens) and channels. In particular, a so-called *bottleneck architecture* is usually chosen for the MLP. It consists of two FC layers of the form:

$$MLP(\mathbf{x}) = \mathbf{w}_2 \phi(\mathbf{w}_1 \mathbf{x}) \quad (2.59)$$

where $\mathbf{x} \sim (e)$ is a single token, $\mathbf{w}_1 \sim (p, e)$, and $\mathbf{w}_2 \sim (e, p)$ are linear weight matrices. The hyperparameter p is chosen as a multiple of e (usually $3e$ or $4e$) so that the first layer projects each token to a higher dimension and the second reprojects it back to the original shape.

Finally, two additional regularization strategies are needed to train a transformer block efficiently: residual connections and layer normalization. Depending on where *LN* is applied, we obtain two different configurations, schematized in Fig. 2.31. The post-normalization version is the original configuration proposed

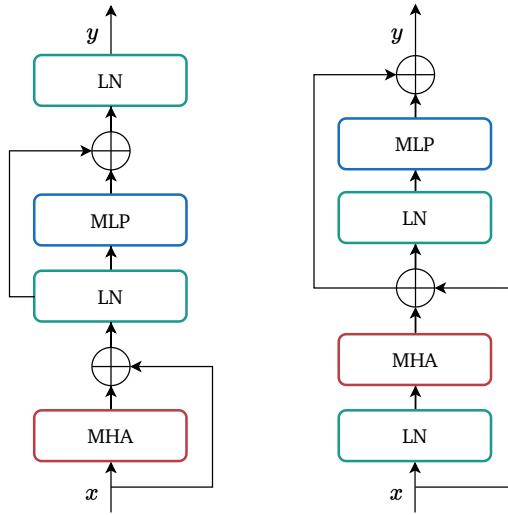


Fig. 2.31 Two versions of the transformer block: post-normalization (left) and pre-normalization (right).

in [50], while the pre-normalization alternative was later found to be more stable and faster to train [53].

Now that I have defined the structure of a transformer block, I can present the full transformer model architecture (Fig. 2.32):

1. Input Tokenization and embedding $\rightarrow \mathbf{x} \sim (n, e)$.
2. Positional embedding $\rightarrow \mathbf{s} \sim (n, e)$.
3. Transformer blocks $\rightarrow \mathbf{h} \sim (n, e)$.
4. Task-specific head $\rightarrow y$.

Note that the transformer block does not alter the input shape, as it preserves the same token sequence, utilizing residual connections and avoiding pooling operations. To implement our task-specific head, we could apply some pooling to the final token sequence and use a linear layer to get the desired output shape (similar to what we do for CNNs). However, the structure and properties of transformers enable more nuanced approaches to optimize information transfer between the backbone and the head.

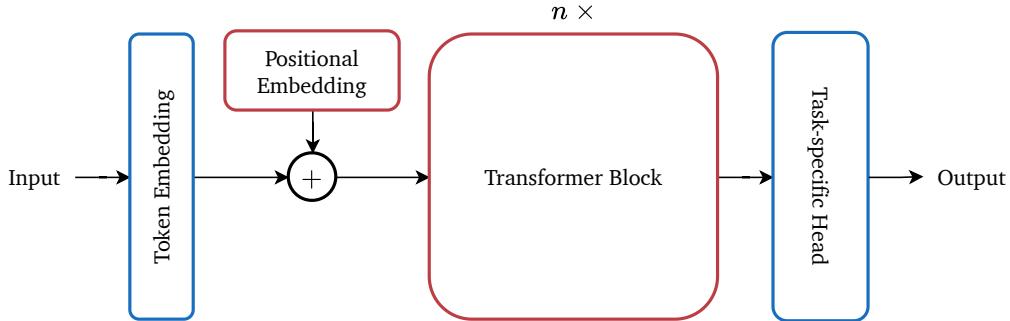


Fig. 2.32 Complete structure of the transformer model.

2.5.4 Additional Tokens

For example, imagine we want to use a transformer for image classification. Instead of pooling from all the output tokens, we can create an additional ad-hoc token c and concatenate it to the embedded input.

$$\mathbf{x}' = \begin{bmatrix} \mathbf{x} \\ c \end{bmatrix} \quad (2.60)$$

We can then design the final pooling operation to keep the c token and discard the others. This choice may sound counterintuitive if we overlook the power of training. Indeed, pooling implicitly forces the model to learn to store the most relevant information for the classification task compactly within c . We call c the *classification token* [54], and usually initialize it as a trainable tensor. Hence, the classification head will be

$$y = \text{softmax}(\text{MLP}(\mathbf{h}_{n+1})) \quad (2.61)$$

This token shows the deep learning power of transformers in routing information across tokens. On the same note, we could add more tokens without explicitly using them for prediction. Instead, they could serve as additional “memory storages” (called *registers* [55]) for information throughout the inference process.

2.5.5 Computational Considerations

So far, we have seen the potential of transformer models thanks to the MHA operation. However, this design paradigm comes with a cost. Indeed, every token must attend to all the others, making this computation more expensive than convolution. Regarding computational complexity in software, two key points of view must be considered: memory and time.

If we look at Eq. (2.53), we can see that the linear operation inside *softmax* scales as $O(n^2k)$ as we compute a dot product for each pair of tokens. This quadratic growth can be problematic for very long sequences, while convolutions scale linearly with sequence length. Several approaches have been proposed to alleviate this problem, the most notable being *online softmax* [56], *KV cache* [57], *flash attention* [58], and *latent attention* [59].

Moreover, some approaches have been proposed to keep the patch-wise philosophy of transformers but use MLPs [60] and convolutions [61] instead of MHA, respectively. Just like transformers, these models separate spatial and channel information, maintaining the data shape constant throughout the blocks. The primary advantage of these alternatives is their higher computational efficiency, which makes them a viable option in resource-constrained scenarios.

2.5.6 Multimodal Transformer

Although initially proposed for natural language processing, the transformer architecture was soon adapted to other data modalities. Most notably, Dosovitskiy *et al.* [54] were the first to propose a tokenization strategy for images. As transformers scale quadratically with the sequence length, treating each pixel as a token is computationally infeasible. Instead, the authors proposed dividing the input image into small square patches (e.g., 16×16 pixels) and embedding them as sequence tokens (Fig. 2.33). Specifically, each patch is flattened and projected to a defined embedding size through a convolutional layer. The resulting **vision transformer** (ViT) uses trainable positional embedding (as the position must be encoded considering two dimensions here) and a class token to learn image classification.

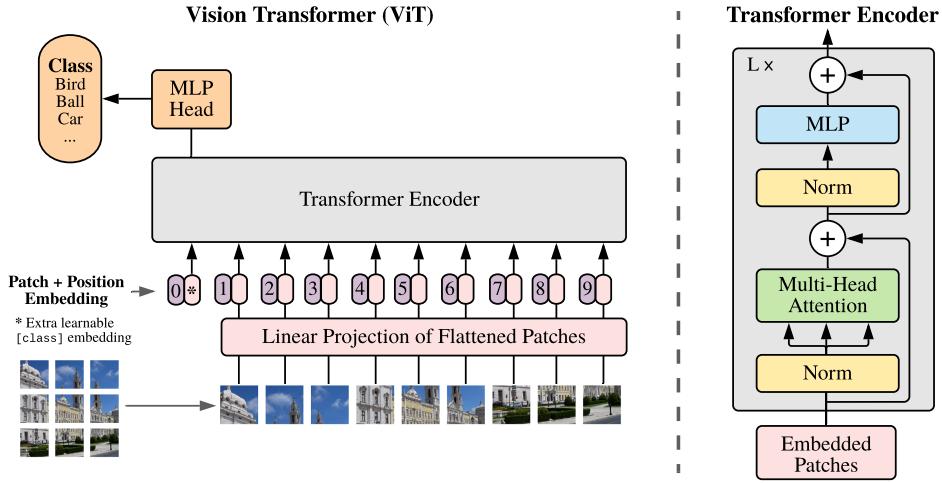


Fig. 2.33 Scheme of the Vision Transformer (ViT) architecture [54].

This approach, later extended to other modalities such as audio, video, graphs, and time series, allows for the complete decoupling of the data modality from the architecture. It is also the basis for *multimodal* variants, which can take different data types as input (or provide them as output). Essentially, each modality is tokenized with its appropriate method, and the resulting tokens are concatenated into a single sequence [51]. Output tokens are generated using an autoregressive approach, meaning the model progressively predicts (applying a regression) based on its previous outputs (Fig. 2.34).

2.6 Advanced Learning Paradigms

In the previous sections, I focused most of my attention on the incredible innovations of advanced deep learning architectures, such as residual convolutions and transformers. However, the possibility of building large-scale models and the availability of massive data have also favored the introduction of new learning paradigms. In this section, I provide a brief overview of the main approaches and highlight their impact on the field.

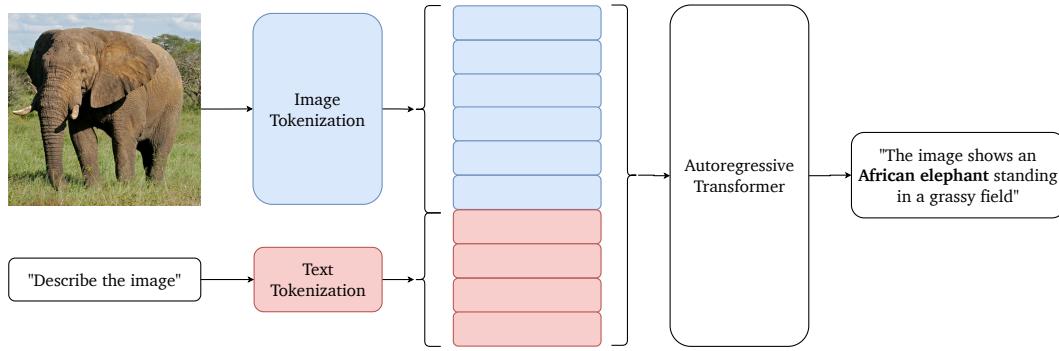


Fig. 2.34 Example of a multimodal transformer architecture processing text and images simultaneously to output a textual output.

2.6.1 Transfer Learning

In Section 2.3.4, I have anticipated that in deep learning models, tasks are practically divided into two sub-tasks, namely feature extraction (operated by the *backbone/encoder*) and prediction (operated by the *head*). The decoupling assumption implies that the representation learned by the backbone is not bound to a specific task but is only dependent on the input dataset. Indeed, training deep models on large-scale datasets (e.g., ImageNet for images) has been shown to have beneficial geometrical properties in that semantically similar objects are projected (embedded) into close points in these representations. The internal states of this model can be plotted as points in a high-dimensional space to visualize this result (Fig. 2.35).

However, this ability is only accessible if the model is trained on an extensive dataset that covers sufficiently diverse and generic cases. On the contrary, training very large models on smaller datasets from scratch is practically impossible. For example, imagine we want to train a model to classify images containing two different types of fruit: cherries and peaches. We aim to achieve high-accuracy performance, but we lack sufficient supervised samples to train a large model (e.g., a ResNet). Now, assume the same model has already been optimized to classify some other images that we presume are sufficiently generic, e.g., ImageNet [29]. We refer to that as a *pre-trained* model. Since the *cherry* and *peach* classes are absent in ImageNet, we cannot use the pre-trained model directly for our scope. One solution is to initialize the encoder using the pre-trained model and *fine-tuning* the model by optimizing the task-specific head. In this way, we exploit

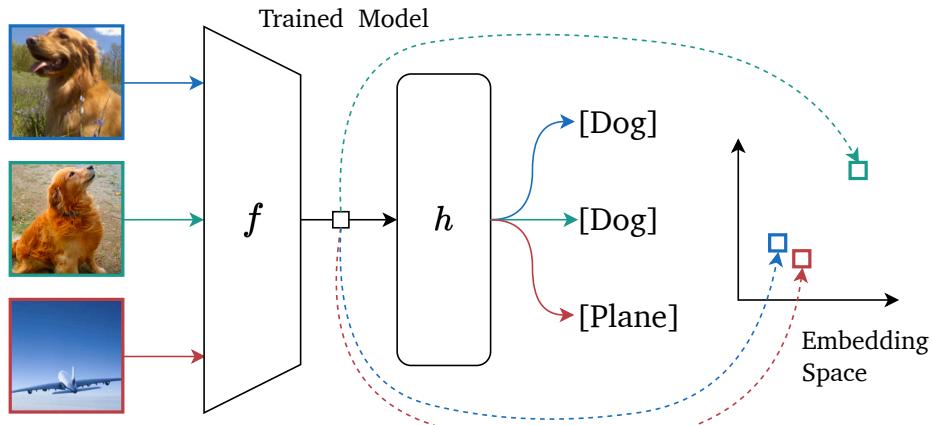


Fig. 2.35 The internal representations of trained models show some geometric properties: semantically similar images are projected to close points in the latent space. Transforming data from a non-metric space (original input images) to a metric space (bottom right) is called embedding the data.

the general representation ability of a well-optimized model for a downstream task. Indeed, even if ImageNet does not contain cherries and peaches, it presents other fruit (e.g., apples and oranges) among its 1000 different classes. We can suppose, then, that the pattern recognition ability necessary to model such a variety of objects can be beneficial in classifying new ones. Since the number of parameters to be trained is drastically reduced, a tiny supervised dataset is sufficient to optimize the head properly. Using a pre-trained backbone optimized on large-scale datasets in a different context is standard practice and confirms the decoupling assumption of deep models.

In general, this paradigm can be extended even further to cases where the downstream task is substantially different. Following our previous example, suppose we now want to classify images and locate fruits within the images at a pixel level. That translates into a **transfer learning** problem from *classification* to *semantic segmentation*, which requires a differently designed head to be performed but exploits the same general-purpose encoding knowledge (Fig. 2.36).

We have discovered that well-optimized feature extractors are both class-agnostic and task-agnostic. This finding makes pre-trained encoders a valuable, general-purpose asset that we can leverage for multiple tasks simultaneously. By appending various heads, we only need to run the input data through the backbone once to obtain all the outputs in a *multi-task* fashion. In the next

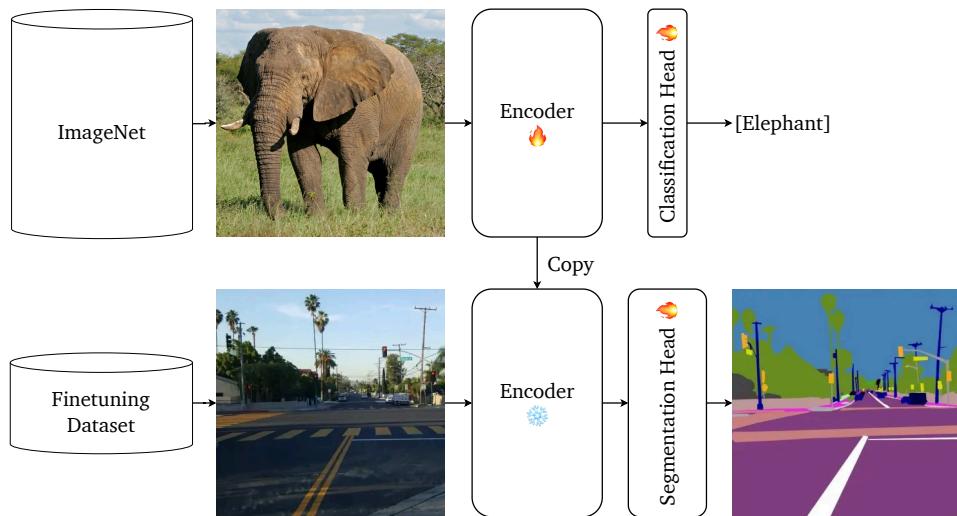


Fig. 2.36 Transfer learning from image classification to semantic segmentation.

chapter, we will see how the representation capability of large models can also be indirectly exploited to bootstrap the training of smaller models through *knowledge distillation*.

In the last couple of years, the advances of deep learning and computational hardware have unlocked the possibility of training very large models that aim at having a universal encoding capability for a particular data modality (text, images, etc.), the so-called **foundation models** [62]. Moreover, the shift in how we interact with these models has pushed the concept of transfer learning to an entirely new stage. Many tasks can now be solved by simply prompting (i.e., interacting with text or visual instructions) a pre-trained model, with the model's internals remaining a complete black box.

 From a high-level perspective, this is similar to a shift from having to program your libraries in a programming language towards relying on open-source or commercial software whose source code is inaccessible. This analogy aligns well with Yann LeCun's definition of differentiable programming⁵.

In the simplest case, they can be used out-of-the-box for a new task, such as answering a query: in this case, we say they are used in a *zero-shot* fashion. It is also possible to provide a small number of examples of a new task as an input prompt, in which case we refer to it as *few-shot* prompting.

⁵Yann LeCun - "Deep Learning est mort. Vive Differentiable Programming!".

2.6.2 Adversarial Training

Adversarial training is a learning framework that stems from game theory and has been applied to various fields (most notably, *generative models*, image denoising, and *domain generalization*). It consists of two models contesting each other in a zero-sum game, where one's gain is the other's loss.

The paradigm was first proposed by Goodfellow *et al.* [63] to generate new images based on the training data statistics. The core idea of a *Generative Adversarial Network* (GAN) is based on the evolutionary arms race between the *generator* G and the *discriminator* D , another neural network that can tell how “realistic” the input seems, which itself is also being updated dynamically. That means the generator is not trained to minimize the distance to a specific image but rather to fool the discriminator. That enables the model to learn in a self-supervised manner (Fig. 2.37).

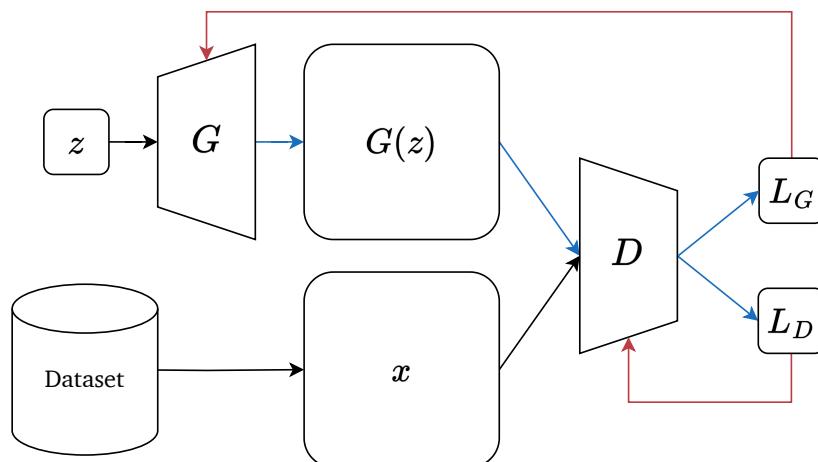


Fig. 2.37 Training scheme of a Generative Adversarial Network.

In its original formulation, the generator tries to minimize the following cross-entropy loss function:

$$L_G(z) = E_z[\log(1 - D(G(z)))] \quad (2.62)$$

where E_z is the expected value operator over all the generated images $G(x)$ and $D(G(x))$ is the discriminator's estimate of the probability that a generated image

is real. At the same time, the discriminator tries to maximize the following loss:

$$L_D(\mathbf{x}, \mathbf{z}) = E_{\mathbf{x}}[\log(D(\mathbf{x}))] + L_G(\mathbf{z}) \quad (2.63)$$

where $D(\mathbf{x})$ is the prediction for a real sample \mathbf{x} , and $E_{\mathbf{x}}$ is the expected value operator over all the authentic images. Since the second term coincides with the generator loss, competition between the two players will prompt them to thoroughly understand the data distribution, leading to improved generations.

This formulation was later improved, and many alternatives were proposed. Most notably, DCGAN [64] was the first application of the framework to fully convolutional networks. Wasserstein GAN [65] later replaced GAN's *Jensen-Shannon divergence* with Wasserstein's *earth-moving distance*, obtaining

$$L_G(\mathbf{z}) = E_{\mathbf{z}}[D_{WGAN}(G(\mathbf{z}))] \quad (2.64)$$

For a Wasserstein GAN, the discriminator D_{WGAN} provides a gradient equal to 1 almost everywhere. At the same time, for a GAN, $\ln(1 - D)$ has a flat gradient in the middle and a steep gradient elsewhere. As a result, the variance for the estimator in GAN is usually much more significant than that in Wasserstein GAN, leading to a less stable learning process.

CycleGAN [66] is an ingenious variant proposed for image-to-image translation. Its key idea is to apply two opposite transformations to an image in sequence and assess how far the result deviates from the original image (Fig. 2.38). However, in recent years, GANs have been surpassed by *stable diffusion* models [67] in many applications.

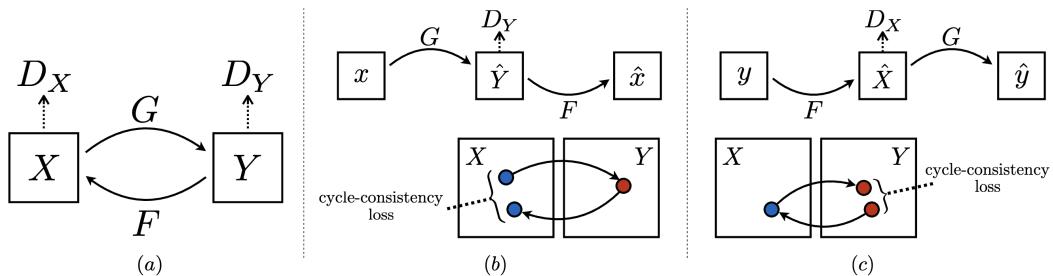


Fig. 2.38 General scheme of the CycleGAN training [66].

2.6.3 Contrastive Learning

The scaling rush we have seen in recent years has increased the need for high-quality data for supervised learning. However, labeling data is costly and time-consuming, especially for tasks that require fine-grained labels (e.g., segmentation). This obstacle has prompted the scientific community to adopt alternative learning paradigms, with the most successful being *self-supervised learning*. As introduced in Section 1.2.1, SSL utilizes a proxy task to implicitly teach models a deeper understanding of a data modality. If the training corpus is sufficiently large, models learn a general latent representation of the input distribution that can be later used for specific downstream tasks. *Contrastive learning* operates under the premise that instances exhibiting similarity should be closely aligned in the learned embedding space, whereas dissimilar ones should be positioned further apart (Fig. 2.39). That translates the proxy task into maximizing the embedding distance of *negative pairs* and minimizing that of *positive pairs*. This discrimination ability equips models with the capability to assimilate and reflect pertinent features and similarities in the dataset. The earliest formulation [68] is the following:

$$\mathcal{L}(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} \|\mathbf{f}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_j)\|_2^2 & \text{if } y_i = y_j \\ \max(0, \varepsilon - \|\mathbf{f}(\mathbf{x}_i) - \mathbf{f}(\mathbf{x}_j)\|_2)^2 & \text{if } y_i \neq y_j \end{cases} \quad (2.65)$$

where y_i and y_j are pseudo-labels indicating whether \mathbf{x}_j is considered a positive or negative sample for \mathbf{x}_i . ε is a lower-bound distance for negative samples.

Alternatives to this formulation include *triplet loss* [69], which optimizes for positive and negative samples at the same time, and *InfoNCE* [70], which uses categorical cross-entropy loss to identify the positive sample among a set of unrelated noise samples.

However, pseudo-classes that allow us to select positive and negative samples are unavailable when dealing with entirely unlabeled datasets. For this reason, alternative approaches have been designed to shift the distinction between positive and negative samples toward determining whether two inputs belong to the same class or category. For example, *SimCLR* [71] learns representations for visual inputs by maximizing agreement between differently augmented views of the same sample via a contrastive loss in the latent space. The two transformations

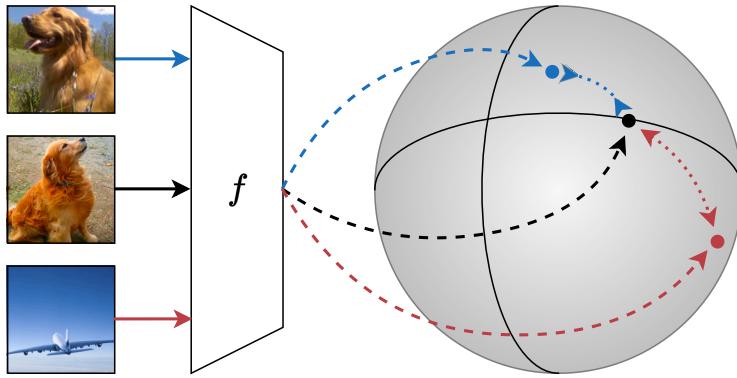


Fig. 2.39 Contrastive learning aims at creating an inner representation of the input space by pushing similar samples (*positive pairs*) closer and different ones (*negative pairs*) further away. In the picture, the encoder $f(\cdot)$ maps the samples to the latent space, represented as a sphere for simplicity.

are a random combination of cropping, flipping, color distortion, and blur, and the loss is based on cosine similarity.

$$l_{i,j} = -\log \frac{e^{\text{sim}(z_i, z_j)/\tau}}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} e^{\text{sim}(z_i, z_k)/\tau}} \quad (2.66)$$

where z_i and z_j are the latent representations of samples i and j , $\mathbb{1}_{[k \neq i]} \in \{0, 1\}$ is an indicator function evaluating to 1 only if $k \neq i$ and τ denotes a temperature parameter. Therefore, this loss accounts for both positive and negative couples simultaneously.

Another interesting approach is *BYOL* (“bootstrap your own latent”), which does not use any negative sample for optimization. Instead, it relies on two neural networks, referred to as *online* and *target* networks, that interact and learn from each other. The models have the same architecture, but the target’s weights are updated with an exponential moving average [72]:

$$\theta_t \leftarrow \tau \theta_t + (1 - \tau) \theta_o \quad (2.67)$$

where θ_t and θ_o are the target and online network weights, respectively. τ acts as a time constant determining how fast the alignment between θ_t and θ_o is. The online model learns to give the same representation to different views of the same sample using an MSE loss (Fig. 2.40). This approach avoids the need for negative samples, but in turn, creates the risk of representation collapse. Indeed,

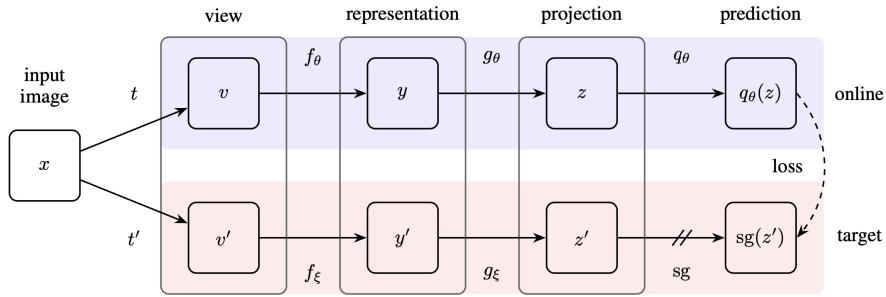


Fig. 2.40 Scheme of the BYOL training methodology [73].

a model that gives the same constant output for any input would perfectly solve the task, but would be useless in practice. Therefore, some design choices must be made to prevent mode collapse, especially during the early training stages.

More variants of contrastive learning have been proposed in recent years, but will not be explicitly mentioned as they are beyond the scope of this work. However, before closing the chapter, it is worth noting that many multimodal foundation models today rely on modules trained using contrastive learning. For instance, *CLIP* (contrastive language-image pre-training [74]) is at the core of all vision-language models (VLMs) and image generation models due to its ability to map images and text to a common latent space.

Now that we have completed our (not exhaustive) venture in the realm of deep learning and discovered some of the latest advancements it has brought to AI, it is time to shift our attention to a huge implication of the progressive upscaling of models. We recognize that this paradigm may be inadequate in certain scenarios, and research efforts should also take another parallel direction.

Bibliography

- [1] A. Bain, *Mind and body: The theories of their relation*. HS King & Company 1873 Farnborough Eng. Gregg International, 1873, vol. 4.
- [2] W. James, “The principles of psychology,” *Henry Holt*, 1890.
- [3] D. O. Hebb, “The organization of behavior,” *J. Clin. Psychol*, vol. 6, no. 3, pp. 335–307, 1949.
- [4] W. McCulloch and W. Pitts, “A logical calculus of ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.
- [5] S. Herculano-Houzel, “The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost,” *Proceedings of the National Academy of Sciences*, vol. 109, no. supplement_1, pp. 10 661–10 668, 2012.
- [6] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [7] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [8] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, “The expressive power of neural networks: A view from the width,” *Advances in neural information processing systems*, vol. 30, 2017.
- [9] V. Nair and G. 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.
- [10] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 315–323.
- [11] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, Atlanta, GA, vol. 30, 2013, p. 3.
- [12] K. He, X. Zhang, S. Ren, and J. Sun, “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.
- [13] H. J. Kelley, “Gradient theory of optimal flight paths,” *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

- [14] S. Linnainmaa, “Taylor expansion of the accumulated rounding error,” *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, 1976.
- [15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.
- [16] C. Darken, J. Chang, and J. Moody, “Learning rate schedules for faster stochastic gradient search,” in *Neural Networks for Signal Processing II Proceedings of the 1992 IEEE Workshop*, 1992, pp. 3–12.
- [17] Y. Nesterov, “A method for unconstrained convex minimization problem with the rate of convergence $\mathcal{O}(1/k^2)$,” in *Dokl. Akad. Nauk. SSSR*, vol. 269, 1983, p. 543.
- [18] D. P. Kingma, “Adam: A method for stochastic optimization,” *arxiv: 1412.6980*, 2014.
- [19] S. J. Reddi, S. Kale, and S. Kumar, “On the convergence of adam and beyond,” in *International Conference on Learning Representations*, 2018.
- [20] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2019.
- [21] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.
- [22] D. Mishkin and J. Matas, “All you need is a good init,” *arxiv: 1511.06422*, 2015.
- [23] X. Glorot and Y. 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.
- [24] R. Pascanu, “On the difficulty of training recurrent neural networks,” *arxiv: 1211.5063*, 2013.
- [25] T. Mikolov, “Statistical language models based on neural networks,” Ph.D. dissertation, Brno University of Technology, 2012.
- [26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [27] D. H. Hubel, T. N. Wiesel, *et al.*, “Receptive fields of single neurones in the cat’s striate cortex,” *J physiol*, vol. 148, no. 3, pp. 574–591, 1959.
- [28] K. Brodmann, *Vergleichende Lokalisationslehre der Grosshirnrinde in ihren Prinzipien dargestellt auf Grund des Zellenbaues*. Barth, 1909.
- [29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [30] A. Uncini *et al.*, *Fundamentals of adaptive signal processing*. Springer, 2015.

- [31] A. K. Maini, *Digital electronics: principles, devices and applications*. John Wiley & Sons, 2007.
- [32] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.
- [33] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [34] R. Sutton, Incomplete Ideas (blog), 2019.
- [35] H. Zou and T. Hastie, “Regularization and variable selection via the elastic net,” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 67, no. 2, pp. 301–320, 2005.
- [36] A. Bauer *et al.*, “Comprehensive exploration of synthetic data generation: A survey,” *arxiv: 2401.02524*, 2024.
- [37] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, “Mixup: Beyond empirical risk minimization,” in *International Conference on Learning Representations*, 2018.
- [38] T. DeVries, “Improved regularization of convolutional neural networks with cutout,” *arxiv: 1708.04552*, 2017.
- [39] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, “Cutmix: Regularization strategy to train strong classifiers with localizable features,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6023–6032.
- [40] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, “RandAugment: Practical automated data augmentation with a reduced search space,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 702–703.
- [41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [42] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML’15, Lille, France: JMLR.org, 2015, pp. 448–456.
- [43] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger, “Understanding batch normalization,” *Advances in neural information processing systems*, vol. 31, 2018.
- [44] J. L. Ba, “Layer normalization,” *arxiv: 1607.06450*, 2016.
- [45] Y. Wu and K. He, “Group normalization,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.
- [46] D. Ulyanov, “Instance normalization: The missing ingredient for fast stylization,” *arxiv: 1607.08022*, 2016.
- [47] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images,” Canadian Institute for Advanced Research, Toronto, ON, Canada, Tech. Rep., 2009.

- [48] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [49] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, “A convnet for the 2020s,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.
- [50] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [51] F. Bordes *et al.*, “An introduction to vision-language modeling,” *arxiv: 2405.17247*, 2024.
- [52] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, “Roformer: Enhanced transformer with rotary position embedding,” *Neurocomputing*, vol. 568, p. 127 063, 2024.
- [53] R. Xiong *et al.*, “On layer normalization in the transformer architecture,” in *International Conference on Machine Learning*, PMLR, 2020, pp. 10 524–10 533.
- [54] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations*, 2021.
- [55] T. Darcet, M. Oquab, J. Mairal, and P. Bojanowski, “Vision transformers need registers,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [56] M. N. Rabe and C. Staats, “Self-attention does not need $\mathcal{O}(n^2)$ memory,” *arxiv: 2112.05682*, 2021.
- [57] R. Pope *et al.*, “Efficiently scaling transformer inference,” *Proceedings of Machine Learning and Systems*, vol. 5, pp. 606–624, 2023.
- [58] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.
- [59] A. Liu *et al.*, “Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model,” *arxiv: 2405.04434*, 2024.
- [60] I. O. Tolstikhin *et al.*, “Mlp-mixer: An all-mlp architecture for vision,” *Advances in neural information processing systems*, vol. 34, pp. 24 261–24 272, 2021.
- [61] A. Trockman and J. Z. Kolter, “Patches are all you need?” *Transactions on Machine Learning Research*, 2023, Featured Certification, ISSN: 2835-8856.
- [62] R. Bommasani *et al.*, “On the opportunities and risks of foundation models,” *arxiv: 2108.07258*, 2021.
- [63] I. Goodfellow *et al.*, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.
- [64] A. Radford, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arxiv: 1511.06434*, 2015.

- [65] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International conference on machine learning*, PMLR, 2017, pp. 214–223.
- [66] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.
- [67] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.
- [68] S. Chopra, R. Hadsell, and Y. LeCun, “Learning a similarity metric discriminatively, with application to face verification,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 539–546 vol. 1.
- [69] F. Schroff, D. Kalenichenko, and J. Philbin, “Facenet: A unified embedding for face recognition and clustering,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.
- [70] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding,” *arxiv: 1807.03748*, 2018.
- [71] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, “A simple framework for contrastive learning of visual representations,” in *International conference on machine learning*, PMLR, 2020, pp. 1597–1607.
- [72] T. Lillicrap, “Continuous control with deep reinforcement learning,” *arxiv: 1509.02971*, 2015.
- [73] J.-B. Grill *et al.*, “Bootstrap your own latent-a new approach to self-supervised learning,” *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.
- [74] A. Radford *et al.*, “Learning transferable visual models from natural language supervision,” in *International conference on machine learning*, PMLR, 2021, pp. 8748–8763.

Chapter 3

Efficient and Robust AI

In the previous chapter, we marveled at the evolution that has rapidly changed what machine learning models can do. Indeed, in less than ten years, we have progressed from shallow models that significantly underperform humans at specific tasks (e.g., image classification) to massive models that fluently process and generate images, text, and many other modalities, exhibiting superhuman performance in solving multiple tasks. This rapid improvement was made possible by the massive growth in computational power and dataset size, as well as new architectures that leveraged this growth.

Although scaling models up has yielded outstanding results, not all application scenarios allow it. These are scenarios where one or more design constraints make it impossible to adopt technological solutions that require heavy computation or large training datasets. Let's break down both factors.

Large models require high computational power which indirectly increases several attributes of an AI-based system: latency, power consumption, physical dimension, hardware cost, and more.

Large models require large datasets, which are not always available for any task and scenario. Moreover, data collection and labeling are costly and time-consuming.

Since both are constrained by cost, modern AI systems have become a prerogative of big tech companies that can invest billions in training large models. This

phenomenon, however, contributes to the progressive concentration of power in the hands of a few people and the increase in social inequalities¹. Hence, the need for a progressive democratization of AI is urgent, and a shift away from simply scaling models up is necessary to provide more people with access to these technologies. Moreover, environmental impact and carbon footprint have recently become a significant issue [1].

For these reasons, a parallel research direction has been established to allow smaller models to achieve the same performance level as bigger ones in all application scenarios while remaining cost-effective. This direction is often referred to as *efficient deep learning* [2] or *frugal AI*, depending on the context.

Among the wide variety of fields that benefit and push efficient AI forward, robotics is undoubtedly one of the most promising and flourishing ones [3]. Indeed, advanced robotic applications need reliable AI systems for perception, control, and decision-making. Still, the integration of machine learning into these tasks is hindered by several strict constraints on latency, power consumption, and physical dimensions. For this reason, research on efficient deep learning is paramount to make robots progressively more intelligent.

Imagine a wheeled mobile robot that navigates in an office and offers help to workers. This system needs an accurate and fast perception system to locate walls and obstacles, detect people, and understand their needs. It typically requires computationally intensive hardware, such as graphics processing units (GPUs). However, mobile robots cannot afford to use such hardware, which would drastically reduce battery life, among other consequences. Moreover, navigation algorithms and control systems must run continuously on the same platform, thereby reducing the available computational power.

One possible solution to this strict constraint could be exploiting cloud computing. Cloud computing enables users to delegate much of the calculation to remote servers, thereby alleviating the need for onboard hardware. However, it is often not possible for two main reasons. First, cloud computing is subject to connection stability and response latency, which limit the system's processing speed. In safety-critical scenarios like robots navigating in populated environments, cloud computing may lead to sudden slowdowns and missed connections (consider that raw video streaming requires considerable bandwidth). Second, sending

¹Pope Francis, 57th World Day of Peace, 2024.

raw video to a remote cloud creates privacy issues, as malicious individuals could access sensitive content if the system is compromised.

On the contrary, *edge AI* guarantees that information is transmitted to external servers only once it has been processed and anonymized. Moreover, the system's response times are not dependent on connection quality, thereby improving the system's robustness in critical cases. For this reason, I will briefly explore methods that make edge computing more feasible and bridge the gap between state-of-the-art models and the constraints of robotic applications.

Another major issue concerning real-world applications is **generalization**. Indeed, most state-of-the-art machine learning models exploit public datasets and benchmarks. These collections of samples have become increasingly extensive and exhaustive over time, aiming to encompass a wide range of different settings. However, even extensive datasets miss some peculiarities that may arise at test time, especially when tackling unstructured, real-world scenarios. Hence, we aim to train models that are not overly sensitive to data distribution shifts and perform consistently across diverse situations. Moreover, we want models to reject noise and processing faults, which often arise in specific testing scenarios. We call this ability **robustness**. This chapter will explore the main techniques to enhance it during or after the training phase.

3.1 Model Compression

The rapid advancement of deep learning has resulted in models capable of solving complex tasks. However, the associated increase in model size and computational demands poses significant obstacles for deployment in resource-limited environments. In the previous chapters, we have already seen some efficient architectural design choices that reduce this gap. Additionally, model compression techniques provide a solution by reducing the model's footprint while maintaining its accuracy. This section provides an overview of prominent compression methods, including parameter pruning, weight quantization, and knowledge distillation, and discusses their respective advantages and limitations.

3.1.1 Quantization

Quantization is a model compression technique that reduces the precision of a neural network's weights and activations, typically from 32-bit floating-point numbers (FP32) to lower-bit-width integers, such as 8-bit integers (INT8). This reduction significantly decreases the model's memory footprint and accelerates computation, making it ideal for deployment on resource-constrained devices.

The fundamental idea is to represent numerical values with fewer bits, translating to smaller model sizes and faster arithmetic operations. For example, an FP32 model requires 4 times the memory of an INT8 one. Reducing precision can inevitably lead to some loss of accuracy. So, the goal is to minimize this loss while maximizing compression. The efficiency of quantized models depends on hardware support for low-precision arithmetic. For instance, many modern processors and specialized AI accelerators provide optimized instructions for INT8 operations.

The most straightforward way to map a tensor x to its quantized version x_Q is the so-called *affine quantization*:

$$x_Q = \text{round}\left(\frac{x}{S} + Z\right) \quad (3.1)$$

where S is the *scale factor* and Z is the *zero-point* of the conversion. The result is rounded to the closest integer via the *round* function. S and Z are computed such that any reasonable value of x is mapped to a reasonable value in the target representation. They, hence, depend on the data, the training procedure of the model, and the layer to which x belongs.

We obtain different methods and flavors depending on when and how quantization is applied. I will discuss the most popular macro-alternatives: *post-training quantization* and *quantization-aware training*.

Post-training Quantization

Post-training quantization (PTQ) occurs when quantization is applied to an existing model. That converts the model from a floating-point representation to a lower-precision fixed-point integer without requiring retraining. This method is fast and only requires a few calibration samples to determine S and Z . However,

as we compress an existing model to a smaller size, post-training quantization can lead to degraded performance. An example of when to utilize PTQ is when you already have a working model and want to increase its speed and efficiency. We can have two variants of PTQ depending on whether we quantize both weights and activations (*static quantization*) or just weights (*dynamic quantization*). In the latter case, quantization is dynamically applied at inference time for each activation depending on the input. Typically, this quantization technique will result in higher accuracy but can be costly.

Quantization-aware Training

Quantization-aware training (QAT) incorporates weight compression during a model's pre-training or fine-tuning phase. It enables enhanced performance but demands significant computational power and requires representative training data.

An example of when to use QAT is when you possess adequate training data and a higher budget. It is also helpful to remember that this process takes place during the training stage of the model, so it does not make sense to use this method with a model that has already been trained. QAT can enable *mixed-precision quantization*, which utilizes varying levels of precision for different network components. For example, some layers might use INT8, while others use FP16. That provides a balance between efficiency and accuracy.

As a final remark, quantization is a powerful tool for making deep learning models more efficient, enabling their deployment in a broader range of applications. Model size should be considered because when quantizing huge models with numerous parameters and layers, there is the risk of significant quantization error accumulation.

3.1.2 Pruning

Model pruning refers to the removal of unnecessary portions from a deep learning neural network model to reduce its size and enable more efficient inference. The simplest case is removing some weights from internal layers, creating a *sparse* architecture. Generally, only the parameter weights w are pruned, leaving the

biases \mathbf{b} untouched. Bias pruning tends to have much more significant downsides. As some parameters are removed, the model's inference performance may be degraded. Hence, it should be performed with care. Here, I will explore the primary types of pruning and their corresponding strategies.

Pruning aims to create a *sparse* network, where many connections are zeroed out. That reduces the number of parameters and operations required during inference. Essentially, it's about identifying and eliminating the parts of the network that contribute the least to its accuracy. It is well-documented that neural networks have an excess of parameters needed to generalize well and make accurate predictions [4], and that they are more over-parametrized the larger they are. The *lottery ticket hypothesis* [5] demonstrates that neural networks tend to have a specific subset of parameters essential for prediction. Model pruning is a very intuitive approach to model compression, with the hypothesis that effective model compression should remove weights that aren't being used, similarly to how the brain reduces usage of connections between neurons to emphasize essential pathways.

Like quantization, pruning can be applied at different moments in the design process:

Training-time Pruning involves integrating pruning directly into the training phase of the neural network. During training, the model is trained in a way that encourages sparsity, removing less meaningful neurons as part of the optimization process. That means the pruning decisions are made simultaneously with the weight updates during the training iterations. Train-time pruning can be implemented using regularization methods, where the penalty terms encourage sparsity, or by incorporating pruning masks into the optimization process.

Post-training pruning involves pruning the trained model after it has been fully trained. Once the model has converged, pruning techniques are applied to identify and remove less essential connections, neurons, or entire structures from the trained model. After this step, additional fine-tuning can be executed to compensate for precision loss.

Pruning techniques can be further divided into unstructured and structured:

Unstructured pruning is a more straightforward approach that cuts individual neurons within the weight matrices. That means all calculations before pruned weights would be performed, and thus, there is limited latency improvement. On the other hand, it can help denoise model weights for more consistent inference and reduce memory usage by up to 50% [6].

Structured pruning is a more architecturally minded approach. By removing entire structured groups of weights or even layers, the method reduces the scale of calculations that would have to be made in the forward pass through the model's weight graph. It has tangible improvements for model inference speed and model size [7]. Given the more ambitious goal, structured pruning methods must be more precise and intentional in pruning entire groups of weights, as the impact spreads across relationships between that node and other nodes in the graph, resulting in severe performance loss.

In essence, pruning is a valuable technique for optimizing deep learning models for deployment on resource-constrained devices by making them smaller and faster.

3.1.3 Knowledge Distillation

Knowledge distillation (KD) [8, 9] is a method for transferring knowledge from one model, known as the teacher, to another, referred to as the student. It can be considered a way to decrease the model's computational complexity, speed up inference, and reduce memory footprint if used between a large and smaller model. The original distillation approach adds a component to the student loss that considers the teacher's probability distribution. Denoting the logit distribution of the student and the teacher as \mathbf{z}_s and \mathbf{z}_t and the softmax activation as $\phi(\cdot)$, the loss is computed as:

$$L = CE(\phi(\mathbf{z}_s), \mathbf{y}) + \lambda \tau^2 \cdot D_{KL}(\phi(\mathbf{z}_s/\tau), \phi(\mathbf{z}_t/\tau)) \quad (3.2)$$

where the first component is the standard cross-entropy loss for the label \mathbf{y} , while the second is the Kullback-Leibler divergence [10] between student and teacher logit distributions. τ is a temperature coefficient for the KD softmax activation

and produces soft labels for distilling the knowledge between the teacher and the student (Fig. 3.1). λ is a parameter balancing the two loss contributions.

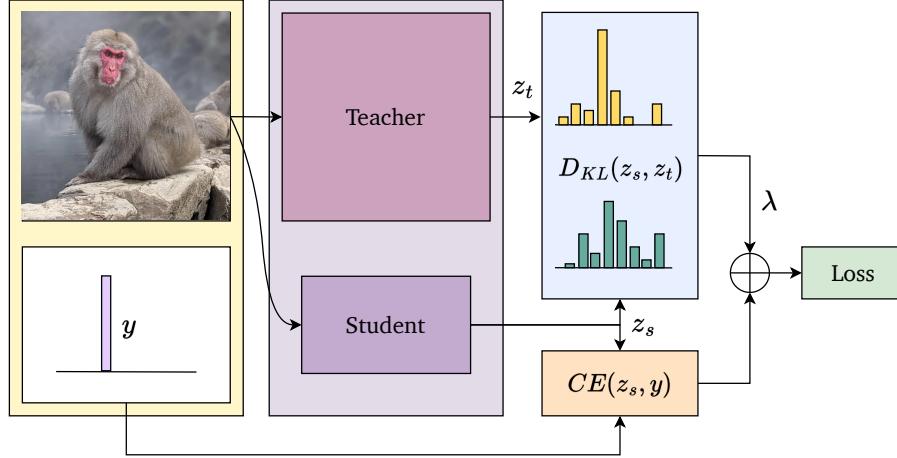


Fig. 3.1 An illustration of standard knowledge distillation.

Touvron et al. [11] introduced a hard label KD for training a data-efficient vision transformer. In this case, a simple cross-entropy contribution is added to the loss, without temperature, to match the student output distribution to the teacher’s hard prediction y_t using an additional distillation token:

$$L = \frac{1}{2}CE(\phi(z_s), y) + \frac{1}{2}CE(\phi(z_s), y_t) \quad (3.3)$$

Several other strategies for distillation have been proposed in the literature, such as distilling intermediate representations [12] or attention-based distillation [13]. Unlike these approaches (generally called offline KD), **online KD** trains the teacher and the student simultaneously [14]. It treats both models as “students” and collaboratively trains them in a single stage. Knowledge is transferred among arbitrary students during collaborative training, and soft supervision targets are generated by effectively ensembling students’ predictions.

As a final mention, **self-distillation** [15, 16] uses the same model as both teacher and student and distills knowledge within the network itself. The networks are first divided into several sections. Then the knowledge in the deeper portion of the networks is squeezed into the shallow ones.

3.2 Robustness

In machine learning, robustness denotes the capacity of a model to sustain stable predictive performance in the face of variations and changes in the input data [17]. It is a broad concept, and to better understand it, I will now delineate how performance degradation and data changes manifest in real-world scenarios. Examples of variations and changes in the input data:

- Variations in input features or object recognition patterns that challenge the inductive bias learned by the model from the training data.
- Production data distribution shifts due to natural distortions, such as lighting conditions or other environmental factors.
- Gradual data drift resulting from external factors, such as evolution in social behavior and economic conditions.
- Noise or faults caused by hardware malfunctioning that reflect in altered computations.
- Malicious input alterations deliberately introduced by an attacker to fool the model or steer its prediction in a desired direction.

These scenarios can threaten the stability of predictive performance. Fragile models often exploit irrelevant patterns and spurious correlations that do not hold up in production settings, and they fail to adapt to edge-case scenarios, which are frequently underrepresented in training samples. Gradually drifting data renders learned concepts obsolete or less representative of the current data distribution, and susceptibility to noise, faults, and adversarial attacks leads to vulnerability in over-parametrized ML models.

In this section, I will explore two main aspects of robustness that are paramount for real-time and robotic applications: *generalization* and *fault tolerance*.

3.2.1 Generalization

While deep learning models often exhibit superhuman generalization performance on public benchmarks and challenges, they can be susceptible to minute changes

in real-world test distributions. That is problematic because sometimes actual underlying data distributions are unknown and significantly underrepresented by the training data. Such mismatches are commonly observed in the real world, leading to significant performance drops. As a result, the reliability of current learning systems is substantially undermined in critical applications such as medical imaging, autonomous driving, and robotics.

Generalization has been studied from diverse perspectives to alleviate the problem created by out-of-distribution (OoD) data. In OoD generalization, models typically have access to multiple training datasets of the same task collected in different scenarios. Generalization algorithms aim to learn features from these datasets that extrapolate to unseen test environments.

Distribution shifts are usually analyzed along two directions, namely **diversity shift** and **correlation shift** [18]. Diversity shift stems from a subset of latent features where the probability of these features appearing in the training distribution ($p(\mathbf{z})$) and the test distribution ($q(\mathbf{z})$) is such that their product is zero ($p(\mathbf{z}) \cdot q(\mathbf{z}) = 0$). Intuitively, this means that novel features cause a diversity shift in one environment that is not present in the other. It is associated with the difference in the support set of the latent environment's distribution. Most datasets used for generalization benchmarks are dominated by diversity shift, such as style changes, different light conditions, and changing environments.

Conversely, correlation shift arises from spurious correlations between specific features and the target labels that differ across training and test environments. Specifically, correlation shift is caused by a subset of latent features where the probability of these features appearing in both the training distribution ($p(\mathbf{z})$) and the test distribution ($q(\mathbf{z})$) is non-zero ($p(\mathbf{z}) \cdot q(\mathbf{z}) \neq 0$), but the conditional probability of the label given these features differs between the two environments ($p(y|\mathbf{z}) \neq q(y|\mathbf{z})$ for some $y \in Y$). In simpler terms, correlation shift occurs when the relationship between certain non-causal features and the label changes between the training and testing data. That means that, unlike diversity shift, the same features are present in both environments, but their relationship with the target variable differs. Fig. 3.2 illustrates the varying levels of diversity and correlation shifts in popular image classification datasets.

In the generalization field, **domain generalization** (DG) has taken an essential spot in recent years. The primary feature of DG is highlighting specific

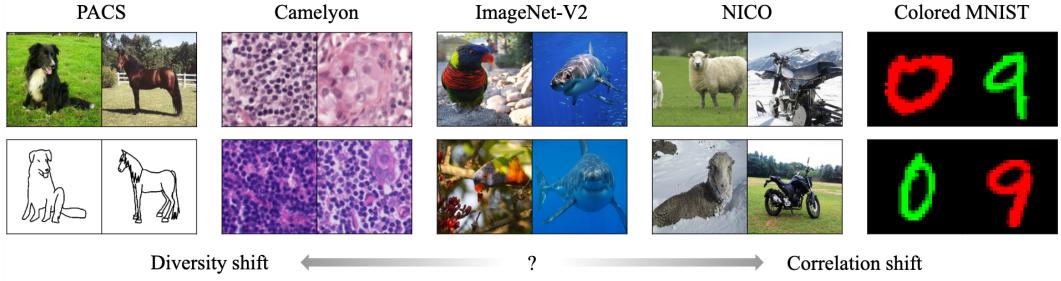


Fig. 3.2 Examples of image classification datasets demonstrating different kinds of distribution shifts [18]. While it is clear that the datasets at both ends exhibit apparent distribution shifts, in the middle, it is hard to distinguish the differences in distribution between the training dataset and the test dataset (e.g., ImageNet [19] and ImageNet-V2 [20]), which represent a large body of realistic OoD datasets.

domains as sub-datasets that share similar distributions. DG differs from *domain adaptation* (DA) because it aims to generalize to unseen target domains. In contrast, domain adaptation typically assumes we have access to unlabeled data from the target domain during training. The primary goal of DG is to learn representations that are robust to domain variations. That involves identifying features that are shared across different domains. This goal can be achieved from various angles, like representation learning, regularization, ensemble methods, and causal reasoning.

Domain Generalization

I first define necessary notations and concepts to frame the problem of domain generalization and empirical risk minimization. Secondly, I introduce a formal definition of a backbone and its constituents.

Problem Definition Given the input random variable X with values $x \in \mathcal{X}$, and the target random variable Y with values $y \in \mathcal{Y}$, the definition of *domain* is associated with the joint probability distribution P_{XY} , or $P(X, Y)$, over $\mathcal{X} \times \mathcal{Y}$. Supervised learning aims to train a classifier $f : \mathcal{X} \rightarrow \mathcal{Y}$ exploiting N available labeled examples of a dataset $D = \{x_i, y_i\}_{i=1}^N$ that are identically and independently distributed and sampled according to P_{XY} . The goal of the training process is to minimize the *empirical risk* associated with a loss function $l : \mathcal{Y} \times$

$\mathcal{Y} \rightarrow [0, +\infty)$,

$$R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N l(f(x_i), y_i) \quad (3.4)$$

by learning the classifier f . The dataset D is the only available source of knowledge to learn P_{XY} . We refer to this basic learning method as *empirical risk minimization* [21].

In domain generalization, a set of different K source domains $\mathcal{S} = \{S_k\}_{k=1}^K$ is used to learn a classifier f 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 $P_{XY}^{\mathcal{S}}$ indicates the overall source distribution learned by the classifier [22]. DG aims 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.

Backbone Definition In the scope of evaluating the DG capability of a model, I define a backbone $\mathcal{B} = f(\mathcal{A}, \mathcal{T}_B, \mathcal{D})$ as a function of three elements: the model architecture \mathcal{A} , the baseline training procedure \mathcal{T}_B (including optimization, regularization, and data augmentation), and the training data \mathcal{D} . Consequently, all three factors introduce a certain degree of variability to the domain generalization accuracy:

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

where \mathcal{T}_{DG} is the adopted DG training procedure and \mathcal{N}_{exp} is the experimentation noise. \mathcal{T}_{DG} typically includes a dedicated algorithm to address 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.

3.2.2 Fault Tolerance

Fault tolerance refers to a machine learning system's ability to function reliably despite noise, errors, or failures. In the same way modern aircraft use redundant control systems to prevent catastrophic failures, robust ML systems need built-in mechanisms to detect, mitigate, and recover from faults, ensuring consistent

performance even in challenging conditions. Here, I provide a brief overview to help readers understand the chapters that follow.

ML systems are vulnerable to various faults that can severely impact performance and reliability from data to deployment. These faults broadly fall into four categories:

- **Data Faults:** Incorrect or changing data leads to flawed models.
- **Hardware Faults:** Failures in GPUs or network issues disrupt inference, compromising output reliability.
- **Software Faults:** Bugs or dependency conflicts can break pipelines.
- **Model Faults:** Overfitting, numerical errors, or adversarial attacks can cause misclassification.

These faults can erode trust and pose safety risks. Therefore, I will explore techniques for building fault-tolerant ML systems, addressing issues from data preprocessing to infrastructure resilience. Depending on the source of the fault and the scenario, various strategies can be employed to ensure robustness against failures while balancing computational costs and complexity. For the scope of this work, I draw my attention to hardware faults common in space exploration and remote sensing.

Hardware Fault Tolerance

Since ML models are heavily used in safety-critical applications, such as automotive and aerospace, their reliability is paramount. However, evaluating neural networks is highly challenging due to the complexity of the software, which consists of hundreds of layers, and the underlying hardware, typically a parallel device or an embedded accelerator [23]. The capability to execute multiple operations in parallel is essential for neural networks; otherwise, execution would take an excessive amount of time. Despite the computing benefit of executing many operations simultaneously, parallelism has some drawbacks regarding reliability. Indeed, a fault in particularly critical units (such as the scheduler or the control unit) or shared resources (such as caches) can impact the correctness of multiple values, leading to malfunctions [24].

Recent findings suggest that transient hardware faults, such as those induced by radiation (neutrons and ions), can significantly corrupt model predictions. Unfortunately, some experimental data indicate that the radiation-induced misprediction probability can be so high as to impede the safe deployment of models at scale, underscoring the need for efficient and effective hardening solutions. This risk is substantial in space applications, where the absence of an atmosphere leaves computers unshielded from radiation. To make the reliability evaluation even more challenging, the software that executes on the hardware being tested is also highly complex. Since neural networks are probabilistic, it is hard to predict the effect of a hardware fault on the software's correctness [24].

For this reason, several methods have been proposed to effectively evaluate and improve the models' fault tolerance. **hardening** methods have proven beneficial for robustness at different levels. Hardware architecture design choices or low-level software strategies can improve fault tolerance by duplicating weights or implementing simple checksums for intermediate outputs to detect faults [25]. Other approaches propose post-training **algorithm-based fault tolerance** (ABFT), slightly modifying the functioning of internal layers (e.g., activations and pooling [26]) to implement more robust operations and reduce fault propagation. Finally, some recent works proposed **fault-aware training** methodologies that teach models to adapt to problems like bit flipping by learning redundant representations [27].

However, for hardening solutions to be effective, they should ideally be designed with experimentally obtained error models or validated through experiments, as solutions effective against simple bit flips might be insufficient against more complex real-world faults. Nonetheless, exploiting the potential of ML is likely to significantly reduce the impact of faults in network prediction, maintaining performance unchanged.

3.3 Hardware

Producer	Google Coral ²		NVIDIA ³				Unibap ⁴
Product	USB Accelerator	Dev Board	Jetson Nano	Jetson AGX Xavier	Jetson Orin Nano	Jetson AGX Orin	iX5
GPU	-	GC7000 Lite	128-core NVIDIA Maxwell	512-core NVIDIA Volta	1024-core NVIDIA Ampere	2048-core NVIDIA Ampere	AMD Radeon
TPU / VPU	Google Edge TPU	Google Edge TPU	-	64-core TPU	32-core TPU	64-core TPU	Intel Movidius Myriad X VPU
Computational Power	4 TOPS	4 TOPS	472 GOPS	32 TOPS	67 TOPS	275 TOPS	4 TOPS
Data Type	INT8	INT8	FP32	FP32	FP32	FP32	FP16
CPU	-	Arm Cortex A53	Arm Cortex A57	NVIDIA Carmel Arm	Arm Cortex A78	Arm Cortex A78	AMD Steppe Eagle
Memory	8 MB	4 GB	4GB	64GB	8 GB	64 GB	2 GB
Memory Type	SRAM	LPDDR4	LPDDR4	LPDDR4	LPDDR5	LPDDR5	DDR3
Storage	-	8 GB	16 GB	32 GB	-	64 GB	240 GB
Storage Type	-	eMMC	eMMC	eMMC	-	eMMC	SSD
Power Consumption	2 W	8.5 W	10 W	30 W	25 W	60 W	30 W
Size [mm]	65 x 30	88 x 60	70 x 45	100 x 87	70 x 45	110 x 110	100 x 100
Price	60 \$	130 \$	124 \$	1,350 \$	250 \$	1,740 \$	*

Table 3.1 Technical specifications of the main commercial edge AI hardware solutions. Each device is reported with its commercial price at the time of writing (*: price upon request). Note that the Coral USB Accelerator is not standalone and requires an external system with a CPU.

Machine learning models can be optimized for inference on a generic CPU, allowing their edge execution on almost every robotic platform. However, it may not be sufficient for deep image processing networks, resulting in inference slowdowns. Specific applications, such as visual-based robot control [28], require low-latency predictions to ensure real-time execution and a prompt response to sudden environmental changes. For this reason, specific computing platforms can be selected to accelerate the execution of AI models. Among the different commercially available solutions, those particularly relevant for the present work are:

²coral.ai

³nvidia.com/autonomous-machines/embedded-systems

⁴unibap.com

- **Nvidia Jetson:** a family of single-board computers with dedicated embedded GPUs to accelerate matrix operations with the CUDA library⁵;
- **Coral Edge Tensor Processing Unit (TPU):** an integrated circuit designed for fast neural network inference; since it is limited to integer-only operations, it requires full-integer model quantization.
- **Intel Movidius Myriad X:** a low-power vision processing unit (VPU) designed for on-device AI inference, accelerating neural networks and computer vision in edge applications. Model compilation is supported by the OpenVINO⁶ toolkit.

These devices exhibit distinct features in terms of inference performance, supported data types, power consumption, physical characteristics, and price. Table 3.1 presents an overview of different AI-oriented platforms. Considering the specific application requirements, the most suitable device can be selected for model deployment.

3.4 Application Fields

To conclude the first part of this thesis, I will outline the main application fields considered in the following chapters. Efficient and robust AI can significantly benefit real-world applications, including robotics, aerospace, and manufacturing. In these contexts, rigid constraints on latency and power consumption contrast with high-stakes requirements on quality and reliability, especially in safety-critical situations. I will restrict my scope to the three most relevant applications for this thesis: *precision agriculture*, *home assistance*, and *remote sensing*.

3.4.1 Precision Agriculture

Agriculture 3.0 and 4.0 have gradually introduced autonomous machines and interconnected sensors into several agricultural processes [29, 30], trying to introduce robust and cost-effective novel solutions into the overall production

⁵developer.nvidia.com/cuda

⁶docs.openvino.ai

chain. For instance, precision agriculture has progressively developed innovative tools for automatic harvesting [31], vegetative assessment [32], crop yield estimation [33], and an innovative and sustainable pesticide spraying robotic system [34], among others [35, 36]. Indeed, the pervasiveness of precision agriculture techniques has such a huge impact on specific fields of application that adopting them has become increasingly essential to achieve high product quality standards [37]. Moreover, the introduction of robots in agriculture is expected to have a growing impact on our economy, politics, society, culture, and security [38], and will be a primary tool in meeting the future food demands of our society [39].

Nevertheless, research on autonomous machines still requires further development and improvements to meet the necessary industrial conditions of robustness and effectiveness. Global path planning [29, 30], mapping [40], localization [41], and decision-making [42] are only some of the required tools that each year undergo heavy research from the scientific community to achieve the requirements for full automation. Among all requirements, a low financial impact constitutes a fundamental goal to provide genuinely large-scale competitive solutions [43]. Indeed, expensive sensors and demanding computational algorithms significantly impact the effectiveness of robotics platforms, essentially preventing their large-scale adoption and introduction into the agricultural world.

Recently, deep learning methodologies [44] have revolutionized the entire field of computer perception, endowing machines with an unprecedented representation of their surrounding environment [45]. Moreover, the intrinsic robustness of representation learning techniques to different factors of variation opens the possibility to achieve noteworthy perception capabilities with low-cost and low-range sensors, greatly relieving the overall cost of the target machine [46, 47]. Finally, the latest advancements in deep learning compression techniques [48] have progressively reduced latency, inference cost, memory, and storage footprint of algorithms. That effectively scales down computational requirements, enabling low-power computational devices boosted by hardware accelerators such as visual processing units (VPUs), tensor processing units (TPUs), and embedded GPU accelerators [49].

3.4.2 Home Assistance

The landscape of home assistance is undergoing a profound transformation, driven by the rapid advancements in deep learning. Once relegated to simple voice commands and basic automation, home assistance systems are now evolving into intelligent, context-aware companions, capable of anticipating needs and providing personalized support [50]. This evolution is primarily fueled by deep learning's ability to process and interpret complex sensory data, enabling systems to understand and interact with their environment [51].

The core strength of deep learning in this domain lies in its capacity to learn intricate patterns and relationships from vast amounts of data. Home assistance systems are no longer limited to pre-programmed routines; they can now adapt to individual preferences, learn daily habits, and anticipate potential issues. That is achieved by analyzing diverse data streams, including audio, video, sensor readings, and user interactions [52].

For instance, deep learning models can analyze audio to recognize individual voices, discern emotional tones, and understand complex speech patterns, enabling more natural and intuitive voice interactions [53]. Computer vision techniques allow systems to identify objects, recognize faces, and interpret gestures, facilitating seamless interaction with the physical environment [54]. By analyzing sensor data from smart home devices, deep learning algorithms can monitor energy consumption, detect anomalies, and optimize resource usage, contributing to increased efficiency and sustainability.

The integration of deep learning into home assistance systems presents several challenges. Data privacy and security are paramount, as these systems collect and process sensitive personal information. Ensuring the ethical and responsible use of deep learning algorithms is crucial for building trust and fostering widespread adoption. Additionally, developing robust and reliable models that can handle noisy and uncertain data is essential for creating seamless and intuitive user experiences.

3.4.3 Remote Sensing

Over the past few decades, the launch of numerous satellite missions with short revisit times and comparatively high-resolution sensors has provided an extensive repository of remote sensing images. The availability of open-source data by many Earth-observation satellites has made remote sensing more feasible [55]. Open-source data sets from several satellite missions, such as Sentinel-2, are available at no cost. These satellites are equipped with multi-spectral sensors that have a short revisit time and good spatial and spectral resolution, allowing researchers to test modern image analysis techniques to extract detailed information about the target object [56] and monitor dynamic processes on Earth. Additionally, it has become easier to estimate and classify biophysical parameters using several data sources. Overall, the new scenario has created opportunities for land cover monitoring, change detection, and large-scale processing using multi-temporal and multi-source images [57].

Integrating AI directly onboard satellites profoundly changes satellite observation of the Earth, enabling real-time data processing and decision-making capabilities [58]. This progress marks a significant departure from the traditional approach, where data is transmitted to ground stations for post-processing, resulting in delays that can be harmful in critical situations such as natural disasters or environmental monitoring. However, while onboard artificial intelligence has great potential, it presents several technical challenges.

Zhang et al. [58] outline several key challenges, including the difficulty of efficiently integrating AI models with onboard hardware and managing the power consumption of these systems. Benchmarking studies, such as the work by Ziaja et al. [59], have further underscored the difficulties in deploying deep learning models on space-qualified hardware. Their study systematically evaluated the performance of various neural network architectures on onboard space platforms, evaluating the trade-offs between model complexity and onboard resource constraints.

A key milestone in this field was the ESA φ -sat-1 mission, which demonstrated the feasibility of using deep learning algorithms for onboard processing [60]. In this mission, a convolutional neural network was deployed to filter

out cloud-covered images directly in space, marking one of the first successful implementations of a deep neural network on a satellite platform.

This overview of the main application fields covered in the thesis concludes the first part of our journey. I hope those who needed it have learned (or at least grasped) the conceptual pillars of the modern deep learning cathedral. I have explored data, architectures, and training methods to ensure our models are up and running. Now, it is time to put this knowledge to good use and explore how novel machine learning methodologies can enhance the accuracy, efficiency, and robustness of autonomous systems.

Bibliography

- [1] J. Stojkovic, E. Choukse, C. Zhang, I. Goiri, and J. Torrelas, “Towards greener llms: Bringing energy-efficiency to the forefront of llm inference,” *arxiv: 2403.20306*, 2024.
- [2] G. Menghani, “Efficient deep learning: A survey on making deep learning models smaller, faster, and better,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023.
- [3] M. Soori, B. Arezoo, and R. Dastres, “Artificial intelligence, machine learning and deep learning in advanced robotics, a review,” *Cognitive Robotics*, vol. 3, pp. 54–70, 2023.
- [4] C. Rasmussen and Z. Ghahramani, “Occam’s razor,” *Advances in neural information processing systems*, vol. 13, 2000.
- [5] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in *International Conference on Learning Representations*, 2018.
- [6] I. Lazarevich, A. Kozlov, and N. Malinin, “Post-training deep neural network pruning via layer-wise calibration,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 798–805.
- [7] G. Fang, X. Ma, M. Song, M. B. Mi, and X. Wang, “Depgraph: Towards any structural pruning,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 16 091–16 101.
- [8] C. Buciluă, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.
- [9] G. Hinton, O. Vinyals, J. Dean, *et al.*, “Distilling the knowledge in a neural network,” *arxiv: 1503.02531*, 2015.
- [10] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.
- [11] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, “Training data-efficient image transformers & distillation through attention,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 10 347–10 357.

- [12] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” *arXiv: 1412.6550*, 2014.
- [13] S. Zagoruyko and N. Komodakis, “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer,” *arXiv: 1612.03928*, 2016.
- [14] Q. Guo *et al.*, “Online knowledge distillation via collaborative learning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 020–11 029.
- [15] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, “Be your own teacher: Improve the performance of convolutional neural networks via self distillation,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3713–3722.
- [16] S. Hahn and H. Choi, “Self-knowledge distillation in natural language processing,” *arxiv: 1908.01851*, 2019.
- [17] H. B. Braiek and F. Khomh, “Machine learning robustness: A primer,” in *Trustworthy AI in Medical Imaging*, Elsevier, 2025, pp. 37–71.
- [18] N. Ye *et al.*, “Ood-bench: Quantifying and understanding two dimensions of out-of-distribution generalization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7947–7958.
- [19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [20] B. Recht, R. Roelofs, L. Schmidt, and V. Shankar, “Do imangenet classifiers generalize to imangenet?” In *International conference on machine learning*, PMLR, 2019, pp. 5389–5400.
- [21] V. N. Vapnik, “An overview of statistical learning theory,” *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.
- [22] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, “Domain generalization: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 4, pp. 4396–4415, 2023.
- [23] P. Rech, “Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions,” *IEEE Transactions on Nuclear Science*, vol. 71, no. 4, pp. 377–404, 2024.
- [24] Y. Ibrahim *et al.*, “Soft errors in dnn accelerators: A comprehensive review,” *Microelectronics Reliability*, vol. 115, p. 113 969, 2020.
- [25] F. Libano *et al.*, “Selective hardening for neural networks in fpgas,” *IEEE Transactions on Nuclear Science*, vol. 66, no. 1, pp. 216–222, 2018.
- [26] F. F. dos Santos *et al.*, “Analyzing and increasing the reliability of convolutional neural networks on gpus,” *IEEE Transactions on Reliability*, vol. 68, no. 2, pp. 663–677, 2018.

- [27] N. Cavagnero, F. Dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, “Transient-fault-aware design and training to enhance dnns reliability with zero-overhead,” in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2022, pp. 1–7.
- [28] L. Marchionna, G. Pugliese, M. Martini, S. Angarano, F. Salvetti, and M. Chiaberge, “Deep instance segmentation and visual servoing to play jenga with a cost-effective robotic system,” *Sensors*, vol. 23, no. 2, p. 752, 2023.
- [29] V. Mazzia, F. Salvetti, D. Aghi, and M. Chiaberge, “Deepway: A deep learning waypoint estimator for global path generation,” *Computers and Electronics in Agriculture*, vol. 184, p. 106 091, 2021.
- [30] F. Salvetti, S. Angarano, M. Martini, S. Cerrato, and M. Chiaberge, “Waypoint generation in row-based crops with deep learning and contrastive clustering,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part VI*, Springer, 2023, pp. 203–218.
- [31] J. J. Roldán *et al.*, “Robots in agriculture: State of art and practical experiences,” *Service robots*, pp. 67–90, 2018.
- [32] G. Zhang, T. Xu, Y. Tian, H. Xu, J. Song, and Y. Lan, “Assessment of rice leaf blast severity using hyperspectral imaging during late vegetative growth,” *Australasian Plant Pathology*, vol. 49, pp. 571–578, 2020.
- [33] A. Feng, J. Zhou, E. D. Vories, K. A. Sudduth, and M. Zhang, “Yield estimation in cotton using uav-based multi-sensor imagery,” *Biosystems Engineering*, vol. 193, pp. 101–114, 2020.
- [34] D. Deshmukh, D. K. Pratihar, A. K. Deb, H. Ray, and N. Bhattacharyya, “Design and development of intelligent pesticide spraying system for agricultural robot,” in *International Conference on Hybrid Intelligent Systems*, Springer, 2020, pp. 157–170.
- [35] A. Khaliq, V. Mazzia, and M. 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.
- [36] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, “A compilation of uav applications for precision agriculture,” *Computer Networks*, vol. 172, p. 107 148, 2020.
- [37] L. Comba, P. Gay, J. Primicerio, and D. R. Aimonino, “Vineyard detection from unmanned aerial systems images,” *computers and Electronics in Agriculture*, vol. 114, pp. 78–87, 2015.
- [38] R. Sparrow and M. Howard, “Robots in agriculture: Prospects, impacts, ethics, and policy,” *Precision Agriculture*, vol. 22, no. 3, pp. 818–833, 2021.
- [39] T. Duckett *et al.*, “Agricultural robotics: The future of robotic agriculture,” *arxiv: 1806.06762*, 2018.

- [40] S. Garg *et al.*, “Semantics for robotic mapping, perception and interaction: A survey,” *arxiv*: 2101.00443, 2021.
- [41] S. Saeedi *et al.*, “Navigating the landscape for real-time localization and mapping for robotics and virtual and augmented reality,” *Proceedings of the IEEE*, vol. 106, no. 11, pp. 2020–2039, 2018.
- [42] T. Mota, M. Sridharan, and A. Leonardis, “Commonsense reasoning and deep learning for transparent decision making in robotics,” in *European conference on multiagent systems*, 2020.
- [43] S. J. LeVoir, P. A. Farley, T. Sun, and C. Xu, “High-accuracy adaptive low-cost location sensing subsystems for autonomous rover in precision agriculture,” *IEEE Open Journal of Industry Applications*, vol. 1, pp. 74–94, 2020.
- [44] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [45] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A survey of deep learning techniques for autonomous driving,” *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.
- [46] A. Boschi, F. Salvetti, V. Mazzia, and M. Chiaberge, “A cost-effective person-following system for assistive unmanned vehicles with deep learning at the edge,” *Machines*, vol. 8, no. 3, p. 49, 2020.
- [47] M. Martini, S. Cerrato, F. Salvetti, S. Angarano, and M. Chiaberge, “Position-agnostic autonomous navigation in vineyards with deep reinforcement learning,” in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2022, pp. 477–484.
- [48] B. 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.
- [49] V. Mazzia, A. Khalil, F. Salvetti, and M. Chiaberge, “Real-time apple detection system using embedded systems with hardware accelerators: An edge ai application,” *IEEE Access*, vol. 8, pp. 9102–9114, 2020.
- [50] G. Beraldo, R. De Benedictis, A. Cesta, F. Fracasso, and G. Cortellessa, “Toward ai-enabled commercial telepresence robots to combine home care needs and affordability,” *IEEE Robotics and Automation Letters*, vol. 8, no. 10, pp. 6691–6698, 2023.
- [51] A. Navone, M. Martini, S. Angarano, and M. Chiaberge, “Online learning of wheel odometry correction for mobile robots with attention-based neural network,” in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2023, pp. 1–6.

-
- [52] A. A. Cantone, M. Esposito, F. P. Perillo, M. Romano, M. Sebillo, and G. Vitiello, “Enhancing elderly health monitoring: Achieving autonomous and secure living through the integration of artificial intelligence, autonomous robots, and sensors,” *Electronics*, vol. 12, no. 18, p. 3918, 2023.
 - [53] A. Eirale, M. Martini, L. Tagliavini, D. Gandini, M. Chiaberge, and G. Quaglia, “Marvin: An innovative omni-directional robotic assistant for domestic environments,” *Sensors*, vol. 22, no. 14, p. 5261, 2022.
 - [54] Z. Cao, Z. Wang, S. Xie, A. Liu, and L. Fan, “Smart help: Strategic opponent modeling for proactive and adaptive robot assistance in households,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 18 091–18 101.
 - [55] A. Francis and M. Czerkawski, “Major tom: Expandable datasets for earth observation,” in *IGARSS 2024-2024 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2024, pp. 2935–2940.
 - [56] A. M. Wijata *et al.*, “Taking artificial intelligence into space through objective selection of hyperspectral earth observation applications: To bring the “brain” close to the “eyes” of satellite missions,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 11, no. 2, pp. 10–39, 2023.
 - [57] M. Martini, V. Mazzia, A. Khaliq, and M. Chiaberge, “Domain-adversarial training of self-attention-based networks for land cover classification using multi-temporal sentinel-2 satellite imagery,” *Remote Sensing*, vol. 13, no. 13, p. 2564, 2021.
 - [58] B. Zhang *et al.*, “Progress and challenges in intelligent remote sensing satellite systems,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 1814–1822, 2022.
 - [59] M. Ziaja *et al.*, “Benchmarking deep learning for on-board space applications,” *Remote Sensing*, vol. 13, no. 19, p. 3981, 2021.
 - [60] G. Giuffrida *et al.*, “The Φ -sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–14, 2021.

Part II

Making AI Efficient

In the following stages of our journey, I will explore how novel machine learning methodologies can enhance the accuracy, efficiency, and robustness of autonomous systems. I will examine some published research papers I worked on and highlight their main contributions and merits.

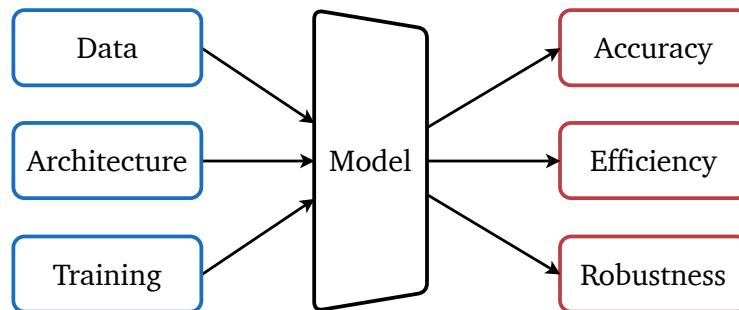


Fig. 3.3 The elements constituting a machine learning model (left) and its properties (right).

Let us examine the scheme in Fig. 3.3 again. In the previous sections, I have discussed the importance of data, architecture, and training in building a machine learning model.

This part groups three primary studies that propose novel architectures or training methods for improving inference **efficiency** without losing **accuracy**. In particular, Chapter 4 presents a fully self-attention-based model for pose-based human action recognition. Chapter 5 introduces an edge-AI GAN model for image super-resolution trained with knowledge distillation. Finally, Chapter 6 presents an ultra-low-power model to enhance indoor localization using ultra-wideband technology.

Chapter 4

Action Transformer: A Self-Attention Model for Human Action Recognition

Original Paper: *V. Mazzia, S. Angarano, F. Salvetti, F. Angelini, and M. Chiaberge, “Action Transformer: A Self-attention Model for Short-time Pose-based Human Action Recognition”, Pattern Recognition, vol. 124, 2022.* [1]

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 applications, such as robotics [5], surveillance and security [6, 7], autonomous vehicles [8, 9], and automatic video captioning [10, 11, 12]. Most previous works adopted datasets characterized by samples with a long temporal duration [13, 14, 15]. 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, this work focuses on short-term HAR, which aims to continuously classify actions within short past time steps (up to a second). This approach is fundamental for targeting real-time applications. In robotics, for example, HAR should be solved promptly to react to sudden behavioral changes, relying only on near-past information.

Works in this field can be subdivided into two broad categories: video-based and depth-based methodologies [16]. In both, the input consists of a sequence

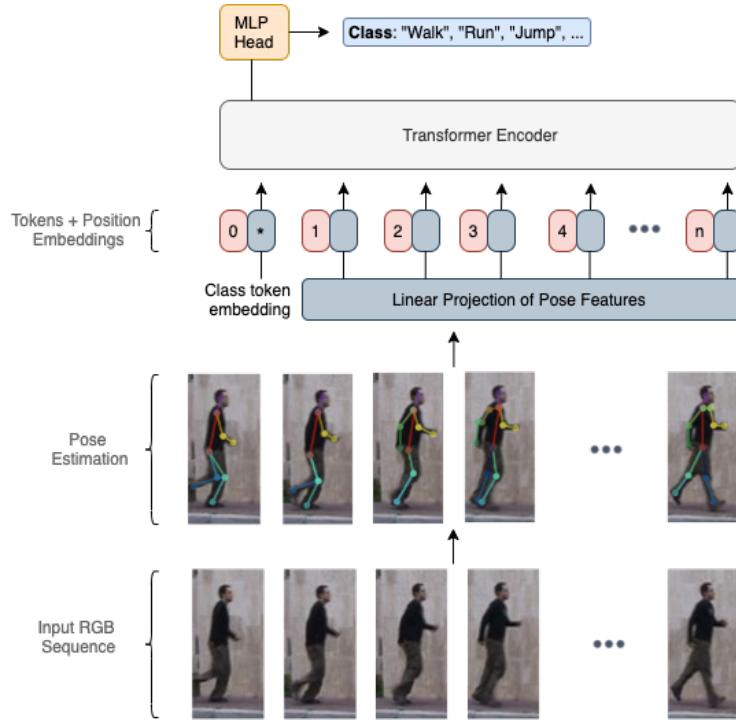


Fig. 4.1 Overview of the Action Transformer architecture. Pose estimations are linearly projected into the model’s dimension, and together with the class token, they form the input tokens of the transformer encoder. As for Vision Transformer models [2, 3, 4], a learnable positional embedding is added to each input token. Then, only the output class token is passed through a multi-layer perceptron head to obtain the final class prediction.

of points representing body joints, extracted from the corresponding RGB frame [17], in the case of video-based methods, or a point cloud, in the case of depth-based ones. In the latter, body joints are provided as 3D coordinates (skeletal data), such as those captured by Kinect sensors [18, 19], that can be possibly projected onto the 2D space [20, 21]. On the contrary, this work focuses on a video-based analysis, where body joints are already provided as 2D coordinates (pose data) by pose detection algorithms such as OpenPose [22] and PoseNet [23]. This characteristic makes 2D HAR methodologies applicable to a great range of applications using a simple RGB camera. Conversely, skeletal data requires particular sensors to be acquired, such as the Kinect or other stereo cameras. That raises substantial limitations, including availability, cost, a limited working range (up to 5-6 meters in the case of Kinect [24]), and performance degradation in outdoor environments.

Angelini *et al.* first collected the MPOSE dataset for video-based short-time HAR [25], obtaining 2D poses by processing several popular HAR video datasets with OpenPose. Moreover, the authors proposed ActionXPose, a methodology that increases model robustness to occlusions. The sequences were classified using MLSTM-FCN [26], which exploits a combination of 1D convolutions, LSTM [27], and Squeeze-and-excitation attention [28]. The same authors successively applied their approach to anomaly detection [29, 30] and expanded MPOSE with the novel ISLD and ISLD-Additional-Sequences datasets. Conversely, [21] first applied OpenPose to extract 2D poses from the Kinetics-400 RGB dataset [20] and used graph convolutions to capture spatial and temporal information. Similarly, [31, 32] applied graph convolutions to pose information using two streams that extract information from both joints and bones to represent the skeleton. [33] introduced knowledge distillation and dense-connectivity to explore the spatiotemporal interactions between appearance and motion streams along different hierarchies. At the same time, [34] put together a hierarchical spatial reasoning network and a temporal stack learning network to extract discriminative spatial and temporal features. On the other hand, [35] proposed an ensemble of two independent bone-based and joint-based models using a unified spatial-temporal graph convolutional operator. Conversely, [36] first applied self-attention [37] to the skeletal-based HAR problem. More recently, [38], inspired by [39], employed self-attention to overcome the locality of the convolutions, again adopting a two-stream ensemble method, where self-attention is applied on both temporal and spatial information.

Unlike previous methodologies, this paper [1] presents an architecture for HAR entirely based on the Transformer encoder, without any convolutional or recurrent layers. Moreover, I focus on verifying the suitability of my method for low-latency and real-time applications. For this reason, I introduce MPOSE2021, a new 2D pose-based dataset specifically designed for short-time HAR. Traditional datasets used by previous works, such as 3D NTU RGB+D [18, 19] or Kinetics-Skeleton [20, 21], include long temporal sequences that must be entirely scanned to make the correct classification. In contrast, the proposed MPOSE2021 dataset includes samples with a temporal duration of no more than 30 frames. That makes it a new and more suitable benchmark for testing the short-time and low-latency performance of HAR models.

I compare AcT with other state-of-the-art baselines to highlight the advantages of the proposed approach. To highlight the effectiveness of self-attention, I inspect

the model, providing visual insights into the results, and study how reducing the input temporal sequence length affects accuracy. I also conduct extensive experimentation on model latency on low-power devices to verify the suitability of AcT for real-time applications.

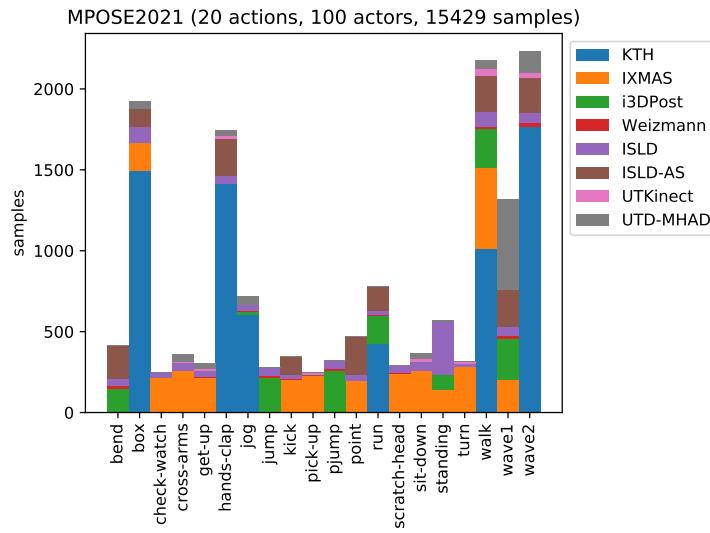


Fig. 4.2 The number of samples of MPOSE2021 divided by action. The colors show the distribution of precursor datasets among classes, highlighting the unbalanced nature of the data. The final dataset contains 15429 samples, each representing one of 100 actors performing one of 20 actions.

4.1 Dataset

In this section, MPOSE2021 is presented as an RGB-based dataset designed for short-time, pose-based HAR. As in [25, 29, 30], video data have been previously collected from popular HAR “precursor” datasets, i.e. Weizmann [40], i3DPost [41], IXMAS [42], KTH [43], UTKinetic-Action3D (RGB only) [44], UTD-MHAD (RGB only) [45], ISLD, and ISLD-Additional-Sequences [25].

Due to the heterogeneity of actions across different datasets, labels are remapped to 20 standard classes. Actions that cannot be remapped accordingly are discarded. Therefore, precursor videos are divided into non-overlapping samples (clips) of 30 frames each whenever possible, and tail samples with more than 20 frames are retained. The peculiarity of a reduced number of time steps

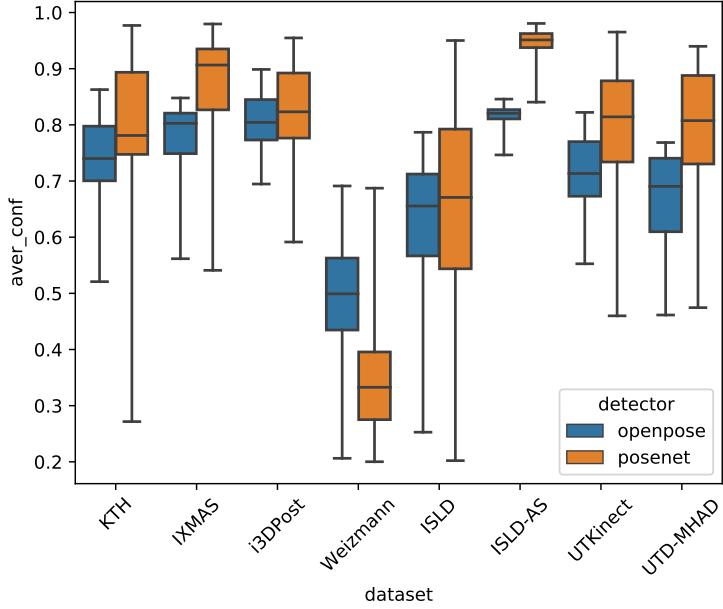


Fig. 4.3 Comparison between OpenPose and PoseNet average confidence (aver_conf) in the different MPOSE2021 sub-datasets. The detectors achieve different average confidence levels based on the considered precursor dataset.

contrasts with other publicly available datasets, stimulating the development of methodologies that require low latency to perform predictions. That would largely benefit many real-world applications that require real-time perception of the actions performed by humans nearby.

Subsequently, clips that do not contain a single action are discarded. Moreover, ambiguous clips are relabelled whenever possible or discarded otherwise. This process leads to 15429 samples, each representing a single actor performing a single action. The total number of distinct actors in MPOSE2021 is 100, and the number of samples for each action is reported in Fig. 4.2, which also shows the distribution of precursor datasets.

OpenPose [22] and PoseNet [23] are used to extract landmarks from the samples. The average confidence for each sample is computed as the mean across landmarks and frames. It turns out that the two detectors achieve different average confidences based on the considered precursor dataset. The box plot of Fig. 4.3 describes the comparison statistics.

Due to the significant sample heterogeneity and the high number of actors, three different training/testing splits are defined for MPOSE2021, namely Split1,

Split2, and Split3, by randomly selecting 21 actors for testing and using the remaining actors for training. This division makes the proposed dataset a challenging benchmark for effectively assessing and comparing the accuracy and robustness of different methodologies. Moreover, the suggested evaluation procedure requires testing a target model on each split using ten validation folds and averaging the results across all splits. That makes it possible to produce statistics and reduces the possibility of overfitting the split testing set with an accurate choice of hyperparameters.

With MPOSE2021, I aim to provide an end-to-end and easy-to-use benchmark for robustly comparing state-of-the-art methodologies in the short-time human action recognition task. I thus release a code repository¹ to access the different levels of the dataset (video, RGB frames, 2D poses). Moreover, I have open-sourced a practical Python package that allows access, visualization, and preprocessing of poses using standard functions. The Python package can be easily installed with the command `pip install mpose`.

4.2 Methodology

In this section, I briefly describe the architecture of the AcT network (Fig. 4.1) and recall some preliminary concepts associated with the Transformer model [37].

4.2.1 Architecture

A video input sequence with t frames of dimension $h \times w$ and c channels $\mathbf{X}_{\text{RGB}} \sim (t, h, w, c)$ is pre-processed by a multi-person 2D pose estimation network

$$[\mathbf{X}_{2Dpose}] = F_{2Dpose}(\mathbf{X}_{\text{RGB}}) \quad (4.1)$$

that extracts a list of n 2D poses $\mathbf{X}_{2Dpose} \sim (t, p)$, 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 \mathbf{X}_{2Dpose} is separately

¹github.com/PIC4SeRCentre/MPOSE2021

processed by the AcT network. At inference time, all detected poses in the video frame can be batch-processed by the AcT model, simultaneously producing a prediction for each of the n subjects (Fig. 4.4).

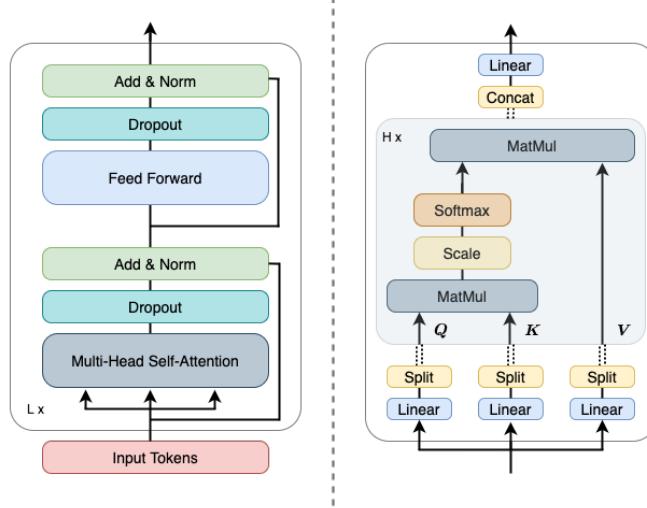


Fig. 4.4 Transformer encoder layer architecture (left) and schematic overview of a multi-head self-attention block (right). Input tokens go through L encoder layers and H self-attention heads.

Firstly, the t poses are mapped to a higher dimension D_{model} using a linear projection map $\mathbf{W}^{l_0} \sim (P, D_{model})$. As in BERT [46] and Vision Transformers [2, 3, 47, 48], 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 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 $\mathbf{X}_{pos} \sim (t + 1, D_{model})$ 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 [37, 49], obtaining

$$\mathbf{X}^L = F_{enc}(\mathbf{X}^{l_0}) = F_{enc}([\mathbf{x}_{CLS}^{l_0}; \mathbf{X}_{2Dpose}] + \mathbf{X}_{pos}) \quad (4.2)$$

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

$$\hat{\mathbf{z}} = \text{MLP}_{\text{head}}(\mathbf{x}_{\text{CLS}}^L) \quad (4.3)$$

where $\hat{\mathbf{z}}$ 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 essential to note that the network's nature enables it to accept a reduced number of frames as input, even when trained with a fixed t . That provides additional flexibility at inference time, making AcT more adaptable than other models.

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

The resulting network is a lightweight solution capable of accurately predicting actions for multiple people in a video stream. The advantage of building on 2D pose estimations enables effective real-time performance with low latency and energy consumption. To reduce the number of hyperparameters and linearly scale the dimension of AcT, I fix $D_{\text{model}}/H = 64$, varying H , D_{mlp} , and L to obtain different versions of the network. A simple grid search using train and validation sets determines lower and upper bounds for the four parameters. In particular, as summarized in Table 4.1, I present the four AcT versions along with their respective numbers of parameters. The four models (micro, small, medium, and large) differ in the number of heads and layers, substantially impacting the number of trainable parameters.

4.3 Experiments

This section describes my primary experiments to investigate the benefits of utilizing a fully self-attentional model for 2D pose-based HAR. First, the four

variants of AcT described in Section 4.2 are compared to existing state-of-the-art methodologies and baselines on MPOSE2021. For a specific comparison with ST-TR [38] and MS-G3D [35], I use additional ensemble versions of my model, named AcT- μ ($x n$), where n represents the number of ensembled instances. Then, I further analyze the network’s behavior to gain visual insight into the attention mechanism and study its performance under a reduction in temporal information. Finally, I measure model latency for all the designed architectures on two different CPU types, demonstrating that AcT can be easily used for real-time applications.

4.3.1 Experimental Setting

In the following experiments, I utilize both the OpenPose and PoseNet versions of the MPOSE 2021 dataset. Either dataset has $t = 30$ and $p = 52$ or 68 features, respectively. In particular, for OpenPose, I follow the same preprocessing as in [25], obtaining 13 keypoints with four parameters each: position (x, y) and velocities (v_x, v_y). In contrast, PoseNet samples contain 17 keypoints with the same information. The training set is composed of 9,421 samples and 2,867 for testing. The remaining 3,141 instances are used for 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 available².

Training		Regularization	
Training epochs	350	Weight decay	1e-4
Batch size	512	Label smoothing	0.1
Optimizer	AdamW	Dropout	0.3
Warmup epochs	40%	Random flip	50%
Step epochs	80%	Random noise σ	0.03

Table 4.2 Hyperparameters used in AcT experiments.

In Table 4.1, all hyperparameters for the four versions of the AcT architecture are summarized, and in Table 4.2, all settings related to the training procedure are listed. The AdamW optimization algorithm [51] is employed for all training with the same scheduling proposed in [37], but with a step drop of the learning rate λ to 1e-4 at a fixed percentage (80%) of the total number of epochs. I employ

²github.com/PIC4SeRCentre/AcT

MPOSE2021 Split		OpenPose 1		OpenPose 2		OpenPose 3	
Model	Parameters	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]	Accuracy [%]	Balanced [%]
MLP	1,334k	82.66 ± 0.33	74.56 ± 0.56	84.41 ± 0.60	74.58 ± 1.00	83.48 ± 0.58	76.60 ± 0.77
Conv1D	4,037k	88.18 ± 0.64	81.97 ± 1.40	88.93 ± 0.43	80.49 ± 0.95	88.67 ± 0.38	83.93 ± 0.58
REMNet [50]	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 [25]	509k	87.60 ± 0.98	82.13 ± 1.50	88.42 ± 0.70	81.28 ± 1.40	89.96 ± 1.00	86.65 ± 1.60
MLSTM-FCN [26]	368k	88.62 ± 0.74	83.55 ± 0.88	90.19 ± 0.68	83.84 ± 1.20	89.80 ± 0.94	87.33 ± 0.67
TTR [38]	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) [35]	2,868k	89.90 ± 0.50	85.29 ± 0.98	90.16 ± 0.64	83.08 ± 1.10	90.39 ± 0.44	87.48 ± 1.20
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 [38]	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) [35]	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 4.3 Benchmark of different models for short-time HAR on MPOSE2021 splits using OpenPose 2D skeletal representations.

TensorFlow³ to train the proposed network on a PC with 32 GB of 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. I utilize publicly available code for other state-of-the-art models and employ the same hyperparameters and optimizer settings as described by the authors in almost all cases. The only exceptions are the learning rate, the number of epochs, and the batch size, which are adapted to my dataset to obtain better learning curves.

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 ± 1.10	87.47 ± 0.35	78.51 ± 0.78	87.46 ± 0.67	81.31 ± 0.58
REMNet [50]	4,269k	84.75 ± 0.65	77.23 ± 0.94	86.17 ± 0.68	75.79 ± 1.30	86.31 ± 0.60	79.20 ± 0.79
ActionXPose [25]	509k	75.98 ± 0.72	64.47 ± 1.10	79.94 ± 1.10	67.05 ± 1.40	77.34 ± 1.40	66.86 ± 1.40
MLSTM-FCN [26]	368k	76.17 ± 0.84	64.75 ± 1.10	79.04 ± 0.72	65.62 ± 1.40	77.84 ± 1.30	67.05 ± 1.20
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 4.4 Benchmark of different models for short-time HAR on MPOSE2021 splits using PoseNet 2D skeletal representations.

³tensorflow.org

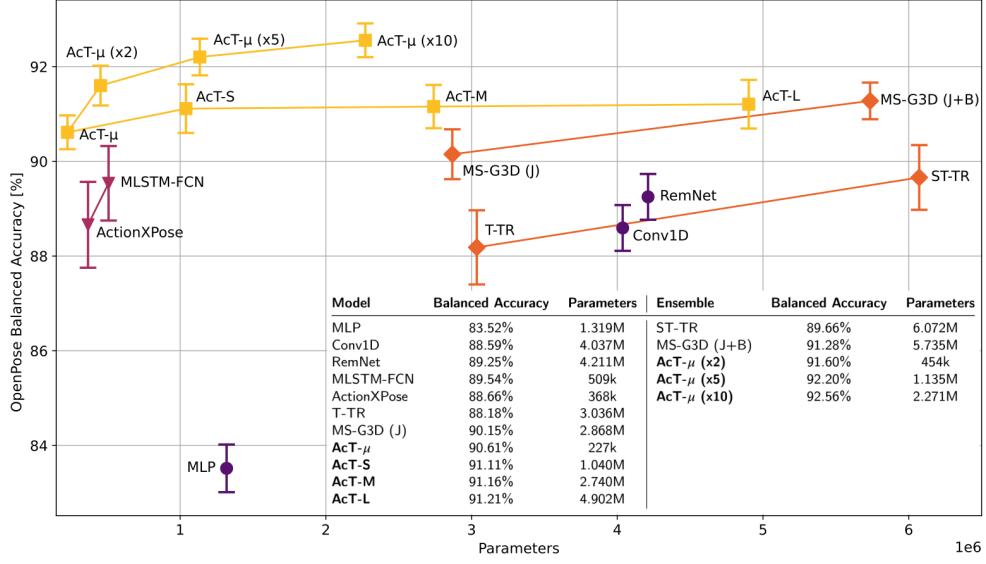


Fig. 4.5 Visual representation of the benchmark of different models for short-time HAR on MPOSE2021 splits using OpenPose 2D skeletal representations. For brevity and clarity, the average balanced accuracy is reported for the three splits of MPOSE2021. The lines connect models that use the same methodology.

4.3.2 Results

I extensively experimented on MPOSE2021, considering some baselines, common HAR architectures, and my proposed AcT models. I report the mean and standard deviation of 10 models trained using different validation splits to obtain statistically relevant results. All models' validation splits are constant and correspond to 10% of the training set, maintaining the same class distribution. The benchmark is executed for both OpenPose and PoseNet data and repeated for all three training and testing splits provided by MPOSE2021. The baselines chosen for the benchmark are a Multi-layer Perceptron (MLP), a fully convolutional model (Conv1D), and REMNet, which is a more sophisticated convolutional network with attention and residual blocks proposed in [50] for time series feature extraction. In particular, the MLP is designed as a stack of three fully connected (FC) layers, each with 512 neurons, followed by a dropout layer and a final FC layer with as many output nodes as the number of classes. Instead, the Conv1D model is built by 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 512 filters, followed by dropout and the same FC output layer as in the other baselines.

Regarding state-of-the-art comparisons, four popular multivariate time series classification models, particularly HAR, are reproduced and tested. Among those, MLSTM-FCN [26] combines convolutions, spatial attention, and an LSTM block, and its improved version, ActionXPose [25], uses additional preprocessing, allowing the model to exploit more correlations in the data and, hence, be more robust against noisy or missing pose detections. On the other hand, MS-G3D [35] uses spatial-temporal graph convolutions to make the model aware of spatial relations between skeletal keypoints, while ST-TR [38] 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. I also report the achieved balanced accuracy for each model and use it as the primary evaluation metric to account for the uneven class distribution.

The experimentation results for OpenPose are reported in Table 4.3 and, in synthesis, in Fig. 4.5. The fully convolutional baseline significantly outperforms the MLP, while REMNet demonstrates that introducing attention and residual blocks further enhances 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, slightly improves the understanding of actions compared to models like REMNet. MS-G3D, in its joint-only (J) version, achieves further accuracy improvements by exploiting graph convolutions and providing the network with information on the spatial relationships between keypoints. On the other hand, ST-TR demonstrates performance comparable to that of all other single-shot models, despite leveraging graph information (e.g., MS-G3D).

The proposed AcT model demonstrates the potential of pure Transformer-based architectures, as all four versions outperform other methodologies while showing smaller standard deviations. Moreover, even the smallest AcT- μ (227k parameters) can extract general and robust features from temporal correlations in sequences. Increasing the number of parameters, a constant improvement in

balanced accuracy is observed for split 3, while splits 1 and 2 exhibit oscillations. The difference between splits reflects the amount of information that training sets convey and the extent to which the model can learn from them. So, it is evident that AcT scales best on split three because it presents complex correlations that a bigger model learns more easily. On the contrary, AcT- μ can extract almost all the information relevant for generalization from split 2, as the accuracy increases only slightly with more complex models.

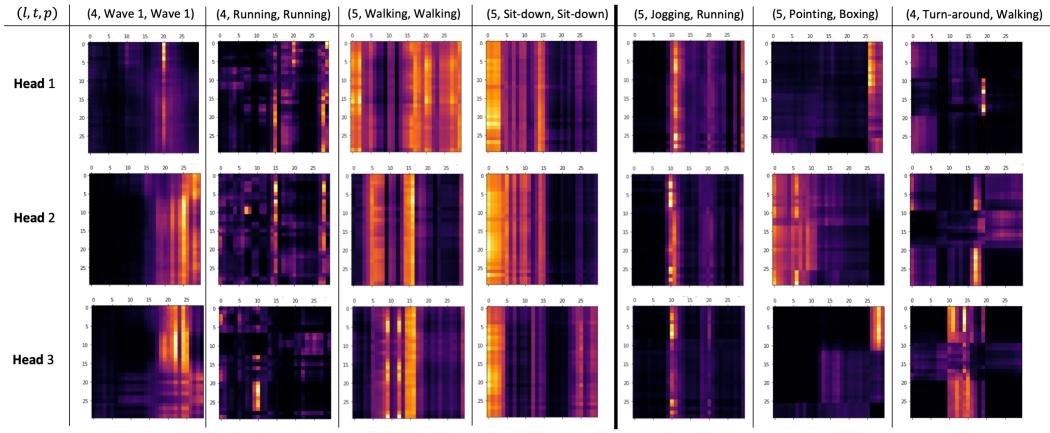


Fig. 4.6 Self-attention weights for MPOSE2021 test samples. (l, t, p) represents the AcT-M l -th layer, the true label, and the predicted label, respectively. The three rightmost columns display three attention maps for a failed prediction, while the others show attention maps from correct classifications. It is clear from all examples how the model focuses on certain particular frames of the series to extract a global representation of the scene.

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 effectively reduce model variance and enhance performance by exploiting independent representations learned by each network, so comparing them with single architectures is unfair. For this reason, I create three ensemble versions of AcT- μ to have an even confrontation, with 2, 5, and 10 instances, respectively. To compute ensemble predictions, I average the output logits of the network instances and then apply a softmax function. The results reported at the bottom of Table 4.3 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), comprising 5 and 10 instances, respectively, achieve even higher accuracy on all splits, with only around 1 to 2 million parameters. That demonstrates how the balancing effect of the ensemble enhances model predictions, even without providing the network with additional information.

Since 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, excluding MS-G3D, ST-TR, and AcT-L. In general, the results give similar insights. The tested models are the same as in the previous case, with all necessary modifications made to accommodate the different input formats. In the MLP case, however, performance seriously degrades as networks tend to overfit the input data strongly after a few epochs, so the results are not included in Table 4.4. 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 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 I 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. Additionally, it is notable that Conv1D outperforms REMNet, demonstrating less susceptibility to overfitting, and that the standard deviations are more pronounced than in the case of OpenPose.

4.3.3 Model Introspection

To gain insight into the frames the AcT model attends to, I extract the self-attention weights at different stages of the network. In Fig. 4.6, MPOSE2021 test samples are propagated through the AcT-M model, and attention weights A for the three heads are presented. It can be seen that the model focuses on specific frames of the sequence when a particular 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 mainly 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.

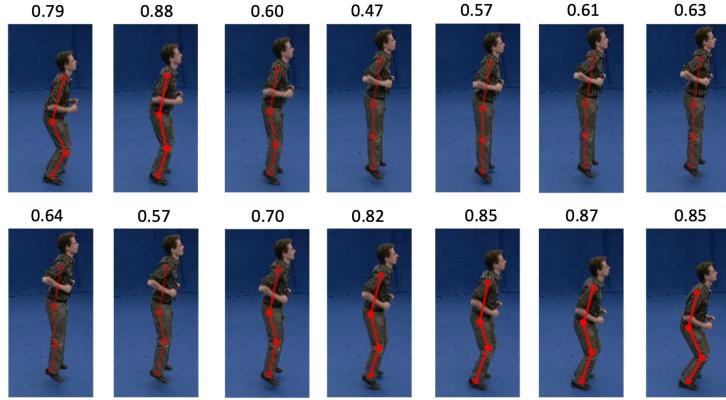


Fig. 4.7 Self-attention of the [CLS] token, computed as the normalized sum of the last layer of the different heads. Scores give a direct insight into the frames exploited by AcT to produce the classification output. The example clearly shows how bending positions are more insightful for the network in predicting the *jumping-in-place* action. In the image, the attention score defines the skeleton’s alpha channel.

Instead, in Fig. 4.7, 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, providing 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, I 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. 4.8, I present how the test-set balanced accuracy is affected by frame dropping. The two curves show a reduction starting from the beginning and end of the temporal sequence. 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 potential for adapting the model to applications with varying temporal constraints.

Finally, I also study the positional embeddings of the AcT-M model by analyzing their cosine similarity, as shown in Fig. 4.9. Nearby positions demonstrate a high level of similarity, and distant ones are orthogonal or even opposite. This pattern

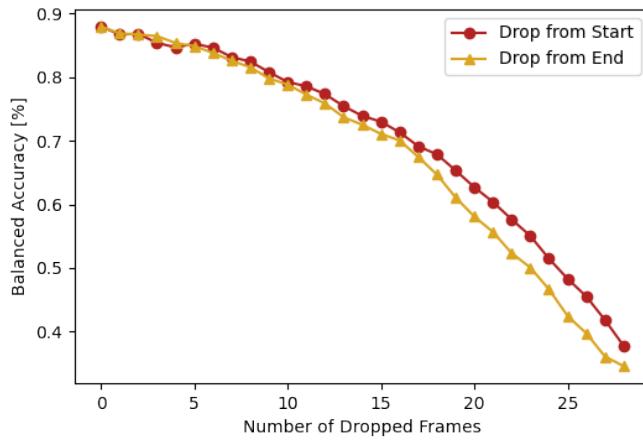


Fig. 4.8 AcT-M balanced accuracy with an incremental reduction of temporal information. Due to the intrinsic nature of the network, it is possible to reduce the number of temporal steps without retraining or any explicit adaptation.

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.

4.3.4 Latency

I test the performance of all the considered models for real-time applications. To do so, I use the TFLite benchmark tool⁴, which allows running TensorFlow Lite models on different computing systems and collecting statistical results on latency and memory usage. In my case, two CPUs are employed to measure model speed 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. 4.10, where only the MLP has been ignored because, despite being the fastest-running model, its accuracy results are much lower than those of its competitors. The graph illustrates the tremendous computational efficiency of Transformer-based architectures, whereas convolutional and recurrent networks result in significantly higher 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

⁴tensorflow.org/lite/performance/measurement

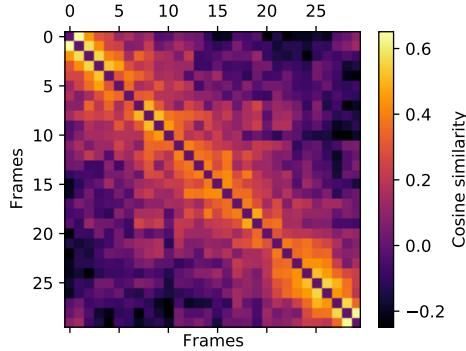


Fig. 4.9 Cosine similarities of the learned t position embeddings of AcT-M model.

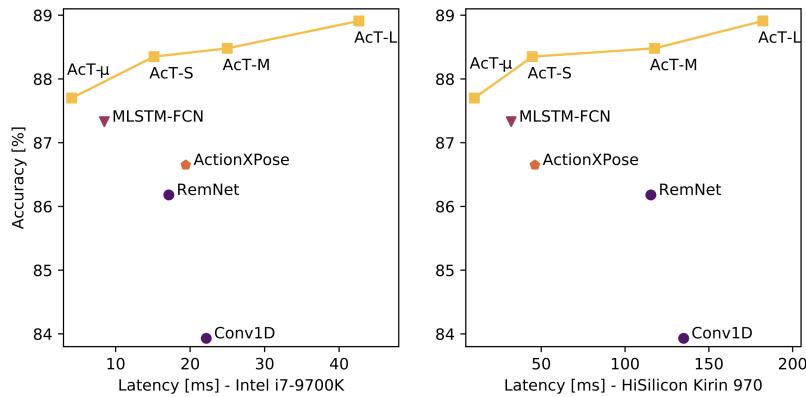


Fig. 4.10 Study of the latency of different tested models on a high-performance Intel CPU and a mobile phone equipped with an ARM-based CPU.

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 AcT- μ still outperforms them in both 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 hardware. Indeed, REMNet and Conv1D run as fast as AcT-M with significantly lower accuracies, and AcT- μ is ten times quicker. Nothing changes concerning MLSTM-FCN and ActionXpose, which are less accurate and three times slower than AcT- μ .

4.4 Conclusion

In this paper, I explored the direct application of a purely Transformer-based network to human action recognition. I introduced the AcT network, which significantly outperforms commonly adopted models for HAR with a simple and fully self-attentional architecture. To limit computational and power requests, building on previous HAR and pose estimation research, the proposed methodology utilizes 2D skeletal representations of short time sequences, providing an accurate and low-latency solution for real-time applications. Moreover, I introduced MPOSE2021, a large-scale, open-source dataset for short-term human action recognition, as an attempt to establish a formal benchmark for future research in this area. Extensive experimentation with the proposed methodology clearly demonstrates the effectiveness of AcT and poses the basis for a meaningful impact on many practical computer vision applications. In particular, the remarkable efficiency of AcT could be exploited for Edge AI, achieving good performance even on computationally limited devices.

Following the development of Action Transformer, the next chapter continues the investigation of efficient deep learning models tailored for robotic applications. While this work focused on temporal understanding through skeletal data, I will address the complementary challenge of visual enhancement in low-bandwidth scenarios, which is critical for tasks such as teleoperation. To this end, I will present a compact and fast solution for Single Image Super-Resolution (SISR), further extending the thesis's central goal: bridging state-of-the-art deep learning with the practical constraints of real-world autonomous systems.

Bibliography

- [1] V. Mazzia, S. Angarano, F. Salvetti, F. Angelini, and M. Chiaberge, “Action transformer: A self-attention model for short-time pose-based human action recognition,” *Pattern Recognition*, vol. 124, p. 108 487, 2022.
- [2] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations*, 2021.
- [3] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, “Training data-efficient image transformers & distillation through attention,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 10 347–10 357.
- [4] S. D’Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun, “Convit: Improving vision transformers with soft convolutional inductive biases,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 2286–2296.
- [5] I. Rodríguez-Moreno, J. M. Martínez-Otzeta, I. Goienetxea, I. Rodriguez-Rodriguez, and B. Sierra, “Shedding light on people action recognition in social robotics by means of common spatial patterns,” *Sensors*, vol. 20, no. 8, p. 2436, 2020.
- [6] G. Vallathan, A. John, C. Thirumalai, S. Mohan, G. Srivastava, and J. C.-W. Lin, “Suspicious activity detection using deep learning in secure assisted living iot environments,” *The Journal of Supercomputing*, vol. 77, no. 1573-0484, pp. 3242–3260, 2021.
- [7] X. Wang and G. Srivastava, “The security of vulnerable senior citizens through dynamically sensed signal acquisition,” *Transactions on Emerging Telecommunications Technologies*, e4037, 2021.
- [8] M. Martin *et al.*, “Drive&act: A multi-modal dataset for fine-grained driver behavior recognition in autonomous vehicles,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 2801–2810.
- [9] H. Ben-Younes, É. Zablocki, P. Pérez, and M. Cord, “Driving behavior explanation with multi-level fusion,” *Pattern Recognition*, p. 108 421, 2021.

- [10] L. Zhou, Y. Zhou, J. J. Corso, R. Socher, and C. Xiong, “End-to-end dense video captioning with masked transformer,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8739–8748.
- [11] Y. Tu, C. Zhou, J. Guo, S. Gao, and Z. Yu, “Enhancing the alignment between target words and corresponding frames for video captioning,” *Pattern Recognition*, vol. 111, p. 107702, 2021.
- [12] B. Wan, W. Jiang, Y.-M. Fang, M. Zhu, Q. Li, and Y. Liu, “Revisiting image captioning via maximum discrepancy competition,” *Pattern Recognition*, vol. 122, p. 108358, 2022.
- [13] L. Huang, Y. Huang, W. Ouyang, and L. Wang, “Part-aligned pose-guided recurrent network for action recognition,” *Pattern Recognition*, vol. 92, pp. 165–176, 2019, ISSN: 0031-3203.
- [14] G. Varol, I. Laptev, and C. Schmid, “Long-term temporal convolutions for action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1510–1517, 2018.
- [15] D. Luvizon, D. Picard, and H. Tabia, “Multi-task deep learning for real-time 3d human pose estimation and action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, 2020.
- [16] L. Song, G. Yu, J. Yuan, and Z. Liu, “Human pose estimation and its application to action recognition: A survey,” *Journal of Visual Communication and Image Representation*, p. 103055, 2021.
- [17] J. Li, X. Liu, M. Zhang, and D. Wang, “Spatio-temporal deformable 3d convnets with attention for action recognition,” *Pattern Recognition*, vol. 98, p. 107037, 2020, ISSN: 0031-3203.
- [18] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang, “Ntu rgb+ d: A large scale dataset for 3d human activity analysis,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1010–1019.
- [19] J. Liu, A. Shahroudy, M. Perez, G. Wang, L.-Y. Duan, and A. C. Kot, “Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 10, pp. 2684–2701, 2019.
- [20] J. Carreira and A. Zisserman, “Quo vadis, action recognition? a new model and the kinetics dataset,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6299–6308.
- [21] S. Yan, Y. Xiong, and D. Lin, “Spatial temporal graph convolutional networks for skeleton-based action recognition,” in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018, pp. 1113–1122.
- [22] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, “Openpose: Realtime multi-person 2d pose estimation using part affinity fields,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 172–186, 2019.

- [23] G. Papandreou, T. Zhu, L.-C. Chen, S. Gidaris, J. Tompson, and K. Murphy, “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.
- [24] B. Langmann, K. Hartmann, and O. Loffeld, “Depth camera technology comparison and performance evaluation.,” in *ICPRAM* (2), 2012, pp. 438–444.
- [25] F. Angelini, Z. Fu, Y. Long, L. Shao, and S. M. Naqvi, “2d pose-based real-time human action recognition with occlusion-handling,” *IEEE Transactions on Multimedia*, vol. 22, no. 6, pp. 1433–1446, 2020.
- [26] F. Karim, S. Majumdar, H. Darabi, and S. Harford, “Multivariate lstm-fcns for time series classification,” *Neural Networks*, vol. 116, pp. 237–245, 2019, ISSN: 0893-6080.
- [27] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [28] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.
- [29] F. Angelini, J. Yan, and S. M. Naqvi, “Privacy-preserving online human behaviour anomaly detection based on body movements and objects positions,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2019, pp. 8444–8448.
- [30] F. Angelini and S. M. Naqvi, “Joint rgb-pose based human action recognition for anomaly detection applications,” in *2019 22th International Conference on Information Fusion (FUSION)*, IEEE, 2019, pp. 1–7.
- [31] L. Shi, Y. Zhang, J. Cheng, and H. Lu, “Skeleton-based action recognition with directed graph neural networks,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7912–7921.
- [32] L. Shi, Y. Zhang, J. Cheng, and H. Lu, “Two-stream adaptive graph convolutional networks for skeleton-based action recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 026–12 035.
- [33] W. Hao and Z. Zhang, “Spatiotemporal distilled dense-connectivity network for video action recognition,” *Pattern Recognition*, vol. 92, pp. 13–24, 2019, ISSN: 0031-3203.
- [34] C. Si, Y. Jing, W. Wang, L. Wang, and T. Tan, “Skeleton-based action recognition with hierarchical spatial reasoning and temporal stack learning network,” *Pattern Recognition*, vol. 107, p. 107 511, 2020, ISSN: 0031-3203.
- [35] Z. Liu, H. Zhang, Z. Chen, Z. Wang, and W. Ouyang, “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.
- [36] S. Cho, M. Maqbool, F. Liu, and H. Foroosh, “Self-attention network for skeleton-based human action recognition,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2020, pp. 635–644.

- [37] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [38] C. Plizzari, M. Cannici, and M. Matteucci, “Skeleton-based action recognition via spatial and temporal transformer networks,” *Computer Vision and Image Understanding*, vol. 208, p. 103 219, 2021.
- [39] I. Bello, B. Zoph, A. Vaswani, J. Shlens, and Q. V. Le, “Attention augmented convolutional networks,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3286–3295.
- [40] L. Gorelick, M. Blank, E. Shechtman, M. Irani, and R. Basri, “Actions as space-time shapes,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 12, pp. 2247–2253, 2007.
- [41] N. Gkalelis, H. Kim, A. Hilton, N. Nikolaidis, and I. Pitas, “The i3dpost multi-view and 3d human action/interaction database,” in *2009 Conference for Visual Media Production*, IEEE, 2009, pp. 159–168.
- [42] D. Weinland, R. Ronfard, and E. Boyer, “Free viewpoint action recognition using motion history volumes,” *Computer vision and image understanding*, vol. 104, no. 2-3, pp. 249–257, 2006.
- [43] C. Schuldert, I. Laptev, and B. Caputo, “Recognizing human actions: A local svm approach,” in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, IEEE, vol. 3, 2004, pp. 32–36.
- [44] L. Xia, C.-C. Chen, and J. K. Aggarwal, “View invariant human action recognition using histograms of 3d joints,” in *2012 IEEE computer society conference on computer vision and pattern recognition workshops*, IEEE, 2012, pp. 20–27.
- [45] C. Chen, R. Jafari, and N. Kehtarnavaz, “Utd-mhad: A multimodal dataset for human action recognition utilizing a depth camera and a wearable inertial sensor,” in *2015 IEEE International conference on image processing (ICIP)*, IEEE, 2015, pp. 168–172.
- [46] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “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)*, J. Burstein, C. Doran, and T. Solorio, Eds., Association for Computational Linguistics, 2019, pp. 4171–4186.
- [47] A. Berg, M. O’Connor, and M. T. Cruz, “Keyword Transformer: A Self-Attention Model for Keyword Spotting,” in *Proc. Interspeech 2021*, 2021, pp. 4249–4253.
- [48] M. Caron *et al.*, “Emerging properties in self-supervised vision transformers,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9650–9660.
- [49] M. X. Chen *et al.*, “The best of both worlds: Combining recent advances in neural machine translation,” *CoRR*, vol. abs/1804.09849, 2018.

- [50] S. Angarano, V. Mazzia, F. Salvetti, G. Fantin, and M. Chiaberge, “Robust ultra-wideband range error mitigation with deep learning at the edge,” *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104 278, 2021.
- [51] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2019.

Chapter 5

Super-Resolution at the Edge with Knowledge Distillation

Original Paper: *S. Angarano, F. Salvetti, M. Martini, and M. Chiaberge, “Generative Adversarial Super-resolution at the Edge with Knowledge Distillation”, Engineering Applications of Artificial Intelligence, vol. 123, 2023.* [1]

Over the last decade, Deep Learning (DL) techniques have become increasingly prevalent in robotic systems and applications, significantly enhancing automation in perception [4, 5], navigation, and control [6, 7] tasks. The development of machine learning algorithms is paving the way for advanced levels of autonomy in mobile robots, significantly increasing the reliability of both uncrewed aerial vehicles (UAVs) and uncrewed ground vehicles (UGVs) [4]. Nonetheless, the adoption of mobile robots for mapping and exploration [8], search and rescue [9], or inspection [10, 11] missions in harsh, previously unseen environments can provide substantial advantages and reduce the risks for human operators. In this context, the successful transmission of images acquired by the robot to the ground station often assumes significant relevance to the task at hand, allowing human operators to receive real-time information, monitor the mission’s state, make critical planning decisions, and analyze the scenario. Moreover, unknown outdoor environments may present unexpected extreme characteristics that still hinder the release of uncrewed mobile robots in the complete absence of human supervision. Although novel DL-based autonomous navigation algorithms are currently under investigation in disparate outdoor contexts such as tunnel

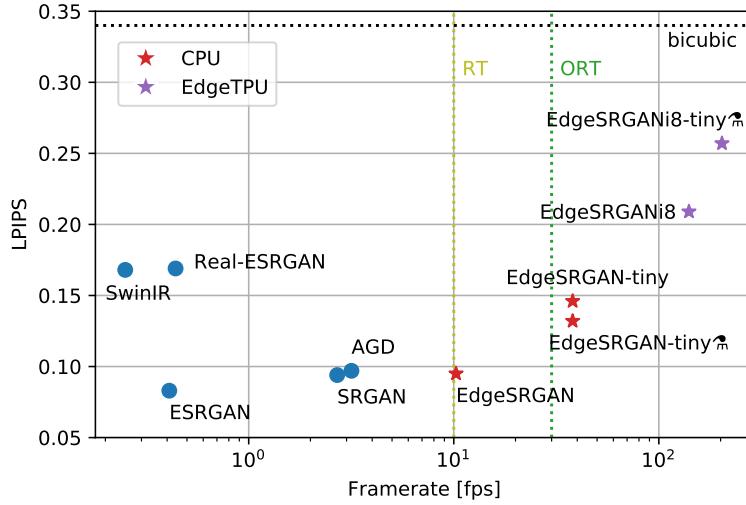


Fig. 5.1 LPIPS [2] results (lower is better) on Set5 [3] vs framerate (80×60 input) of different visual-oriented SISR methods for $\times 4$ upsampling. Real-time (RT) and over-real-time (ORT) framerates are marked as references. My models, marked with *, reach real-time performance with a competitive perceptual similarity index on the CPU. Edge TPU models can further increase inference speed far beyond real-time, still outperforming the bicubic baseline.

exploration [12, 13, 14], row-crops navigation [15, 16], and underwater [17, 18], complete or partial remote teleoperation remains the most reliable control strategy in uncertain scenarios. Indeed, irregular terrain, lighting conditions, and the loss of the localization signal can lead navigation algorithms to fail. As a direct consequence of navigation errors, the robotic platform can become stuck in critical states that require or prefer human intervention.

However, visual data transmission for robot teleoperation, monitoring, or online data processing requires a stable, continuous stream of frames, which may be drastically affected by poor bandwidth conditions due to the long distance of the robot or by constitutive factors of the specific environment. Additionally, UAVs and high-speed platforms require the pilot to receive the image stream at a high frame rate to follow the vehicle's motion in non-line-of-sight situations. A straightforward yet effective solution to mitigate poor bandwidth conditions and meet high-frequency transmission requirements is to reduce the resolution of the transmitted image. On the other hand, heavy image compression with massive loss of detail can compromise image usability.

To this end, I propose EdgeSRGAN [1], a novel deep learning model for Single-Image Super-Resolution (SISR) at the edge, which addresses the problem of efficient image transmission. My intuition relies on a lightweight neural network, allowing us to send low-resolution images at a high transmission rate with scarce bandwidth and reconstruct the high-resolution image on the pilot's mobile device. Moreover, the successful adoption of edge AI in various engineering applications [19, 20, 21] has yielded encouraging results in enabling the execution of DL models on ultra-low-power embedded devices.

Single-Image Super-Resolution, also referred to as super-sampling or image restoration, aims to reconstruct a high-resolution (HR) image from a single low-resolution (LR) input image, while preserving details and the information contained within the image. Therefore, SISR, together with image denoising, is an ill-posed, underdetermined inverse problem, as a multiplicity of possible solutions exists given an input low-resolution image. Recently, learning-based methods have rapidly reached state-of-the-art performance and are universally recognized as the most popular approach for Super-Resolution. Such approaches rely on learning common patterns from multiple LR-HR pairs in a supervised fashion. In literature, SRCNN [22] was the first example of a CNN applied to single-image super-resolution. It has been followed by multiple methods applying standard deep learning methodologies to SISR, such as residual learning [23, 24], dense connections [25], residual feature distillation [26], attention [27, 28, 29], self-attention, and transformers [30, 31, 32]. All these works focus on content-based SR, in which the objective is to reconstruct an image with high pixel fidelity, and the training is based on a content loss, such as mean square error or mean absolute error.

In parallel, other works have proposed Generative Adversarial Networks (GAN) [33] for SISR, aiming to reconstruct visually pleasing images. In this case, the focus is not on pixel values, but on perceptual indices that aim to reflect how humans perceive image quality. That is usually implemented using perceptual losses and adversarial training, and is referred to as visual-based SR. SRGAN [34] first proposed adversarial training and was later followed by other works [24, 35, 36]. With robotic image transmission as a target application in mind, in this work, I particularly focus on visual-based SR, aiming to reconstruct visually pleasing images to be used by human operators for real-time teleoperation and monitoring.

In recent years, efficient deep neural networks for SR have been proposed to reduce the number of parameters while keeping high-quality performances [37]. However, most proposed architectural solutions are designed for content-based training to minimize the difference between the high-resolution image and the network output. Among them, [38] proposed a simple model that handles SR as a bilinear upsampling residual compensation. Despite obtaining high-quality images, this approach has high inference latency due to the double prediction required. Diversely, [39] based their study entirely on targeting edge AI chips, proposing an ultra-tiny model composed of only one layer.

As already stated, I prefer GAN-based SR to enhance the visual appearance of produced images for robotic applications. However, successful studies of efficient GANs are scarce in the literature. Recently, knowledge distillation (KD) has emerged as a promising option for compressing deep models and GANs as well [40, 41]. KD was initially introduced in 2015 through the visionary work of [42], where a teacher-student framework was proposed as a knowledge transfer mechanism. More recent works evolved such concept in disparate variants: FitNets [43] introduced the idea of involving also intermediate representations in the distillation process, Attention Transfer (AT) [44] proposes an attention-based distillation, and Activation Boundaries (AB) [45] interestingly focuses on the distilled transfer of activation boundaries formed by hidden neurons, further advanced in [46]. Specifically, considering the KD application in SR, Feature Affinity KD (FAKD) [47] utilizes intermediate feature affinity distillation for PSNR-focused SR. I also found this approach a good starting point for GAN-based SR. Diversely, [48] investigates a progressive knowledge distillation method for data-free training. Besides KD, [35] recently proposed an automated machine learning (Auto-ML) framework to search for optimal neural model structure, and filter pruning has been used as another optimization technique [49].

Differently from previous works, my model optimization for edge SR is composed of three main steps: first, an edge-oriented architectural definition is performed; then, I leverage teacher-student knowledge distillation to reduce the dimension of my model further; lastly, I perform TensorFlow Lite (TFLite) conversion and quantization to shift the network execution to CPUs and edge TPUs with maximum inference speed.

SISR has been recently proposed in a few robotic applications where a high level of detail is beneficial to support the specific task. Research on the indoor teleoperation of mobile robots primarily focuses on enhancing user experience by combining Deep Learning methods with Virtual Reality [50, 51, 52], but overlooks the potential bottleneck caused by connectivity degradation in harsh conditions. In contrast, a significant effort has been devoted to SISR for underwater robotics perception [53, 54], effectively addressing the challenge of high-quality image acquisition underwater for accurate object and species detection. Besides autonomous navigation applications, interesting contexts include robotic surgery [55, 56] and medical robot research [57], where SISR can provide substantial advantages, improving visibility and increasing the level of detail required for delicate, high-precision movements by the surgeon. Similarly, a detailed image a robot acquires is needed for monitoring and inspections. For example, [58] uses a Super-Resolution model to enhance the online crack detection and in-situ analysis of bridge weaknesses. Nonetheless, no relevant works proposed so far have identified Super-Resolution as an efficient solution for image transmission to support robot teleoperation and exploration of unknown environments in bandwidth-degraded conditions.

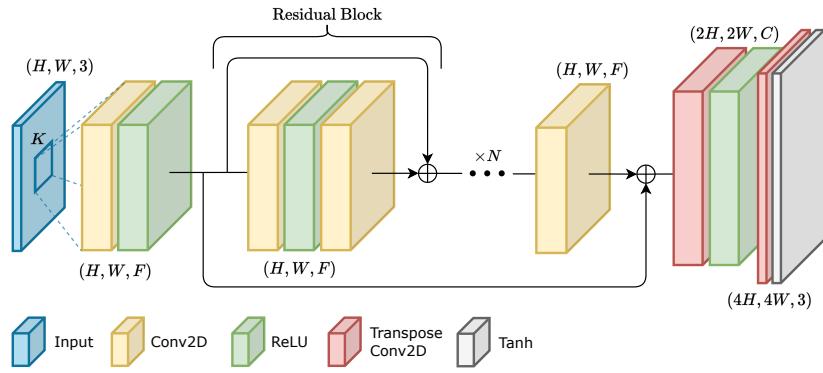


Fig. 5.2 EdgeSRGAN Generator Architecture.

Hence, I propose an edge AI computationally efficient Super Resolution neural network to provide fast inference on CPUs and edge TPU devices. To this end, I employ several optimization steps to enhance the performance of my model while minimizing the quality drop. I refine the architecture of the original SR-GAN [34] to accelerate inference and enable model quantization. Nonetheless, I experimented with a teacher-student knowledge distillation technique for SISR

to further enhance the reconstructed image of my tiny model. I draw inspiration from the work of [47] and achieve a remarkable improvement across all considered metrics.

I perform experiments to validate the proposed methodology from multiple perspectives, including numerical and qualitative analysis of my model’s reconstructed images and inference efficiency on both CPU and edge TPU devices. For example, as shown in Fig. 5.1, EdgeSRGAN achieves real-time performance with a competitive perceptual similarity index compared with other visual-oriented SISR methods. Moreover, I test the performance of my system for robotic applications. In particular, I focus on image transmission for teleoperation in the case of bandwidth degradation, and I also perform tests with the popular robotic middleware ROS2.

5.1 Methodology

In this section, I introduce all the components of the proposed methodology. I use an adversarial approach to obtain an optimal balance between pixel-wise fidelity and perceptual quality. For this reason, I take inspiration from three of the most popular GAN-based solutions for SISR: SRGAN [34], ESRGAN [59], and AGD [35]. The proposed method aims to obtain a real-time SISR model (EdgeSRGAN) with minimal performance drop compared to state-of-the-art solutions. For this reason, I mix successful literature practices with computationally efficient elements to obtain a lightweight architecture. Then, I design the network training procedure to leverage a combination of pixel-wise loss, perceptual loss, and adversarial loss. To further optimize the inference time, I apply knowledge distillation to transfer the performance of EdgeSRGAN to an even smaller model (EdgeSRGAN-tiny). Furthermore, I study the effect of quantization on the network’s latency and accuracy. Finally, I propose an additional inference-time network interpolation feature to allow real-time balancing between pixel-wise precision and photo-realistic textures.

5.1.1 Network Architecture

As previously done by [59], I adapt the original design of SRGAN and propose modifications to both the architecture and training procedure. However, in my case, the modifications aim for both efficiency and performance. To obtain a lighter architecture, I reduce the depth of the model by using only $N = 8$ Residual Blocks instead of the original 16. In particular, I use simple residuals instead of the Residual-in-Residual Dense Blocks (RRDB) proposed by [59] as they are less computationally demanding. For the same reason, I replaced the PReLU activation functions with basic ReLU. I also remove Batch Normalization to allow the model to converge more effectively without generating artifacts [59]. Finally, I use Transpose Convolution for the upsampling head instead of Sub-pixel Convolution [60]. Despite its popularity and effectiveness, Sub-pixel Convolution is computationally demanding due to the Pixel Shuffling operation, which rearranges feature channels spatially. I choose instead to trade some performance for efficiency and apply Transpose Convolutions, taking precautions to avoid problems such as checkerboard artifacts [61]. The complete EdgeSRGAN architecture is described in Fig. 5.2. The adopted discriminator model is the same as that used in [34, 59], as it serves only training purposes and is not needed at inference time. Its architecture is described in Fig. 5.3.

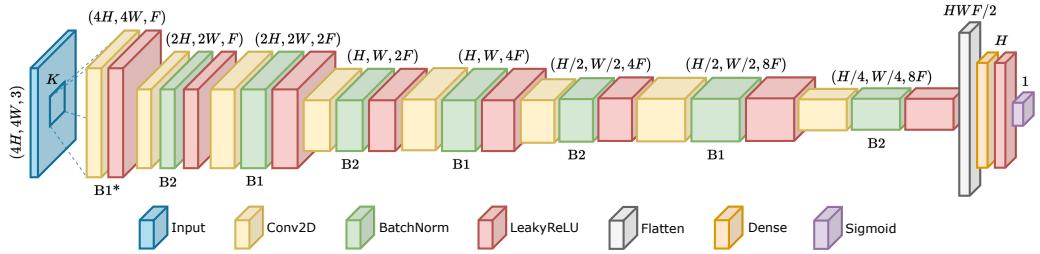


Fig. 5.3 EdgeSRGAN Discriminator Architecture. The model progressively reduces the spatial dimensions of the image by alternating blocks with strides 1 (B1) and 2 (B2). The first block (marked with *) does not apply batch normalization.

5.1.2 Training Methodology

The training procedure is divided into two sections, as it is common practice in generative adversarial SISR. The first part consists of classic supervised training

using pixel-wise loss. In this way, I help the generator to avoid local minima and generate visually pleasing results in the subsequent adversarial training. I use the mean absolute error (MAE) loss for optimization, as it has recently been shown to provide better convergence than the mean squared error (MSE) [62, 24, 27, 59].

$$L_{\text{MAE}} = \sum_{i=1}^B \|y_i^{\text{HR}} - y_i^{\text{SR}}\|_1 \quad (5.1)$$

where y^{HR} is the ground-truth high resolution image, y^{SR} is the output of the generator, and B is the batch size. I validate the model using the Peak Signal-to-Noise Ratio (PSNR) metric.

In the second phase, the resulting model is fine-tuned in an adversarial manner, optimizing a loss that combines adversarial loss and perceptual loss. As presented in [34], the generator G training loss can be formulated as

$$L_G = L_G^P + \xi L_G^A + \eta L_{\text{MAE}} \quad (5.2)$$

L_G^P is the perceptual VGG19 loss, computed as the Euclidean distance between the feature representations of a reconstructed image SR and the reference image HR. The features are extracted using the VGG19 network [63] pre-trained on ImageNet:

$$L_G^P = \sum_{i=1}^B \|\phi(y_i^{\text{HR}}) - \phi(y_i^{\text{SR}})\|_2 \quad (5.3)$$

where ϕ is the perceptual model VGG. L_G^A is the adversarial generator loss, defined as

$$L_G^A = -\log(D(y_{\text{SR}})) \quad (5.4)$$

where D is the discriminator. With this loss, the generator tries to fool the discriminator by generating images indistinguishable from the real HR ones. ξ and η are used to balance the weight of different loss components. The discriminator's weights D are optimized using a symmetrical adversarial loss,

which effectively discriminates between HR and SR images.

$$L_D = \log(D(y_{\text{SR}})) - \log(D(y_{\text{HR}})) \quad (5.5)$$

I optimize both models simultaneously, without alternating weight updates like in most seminal works on GANs. The overall training methodology is summarized in Fig. 5.4.

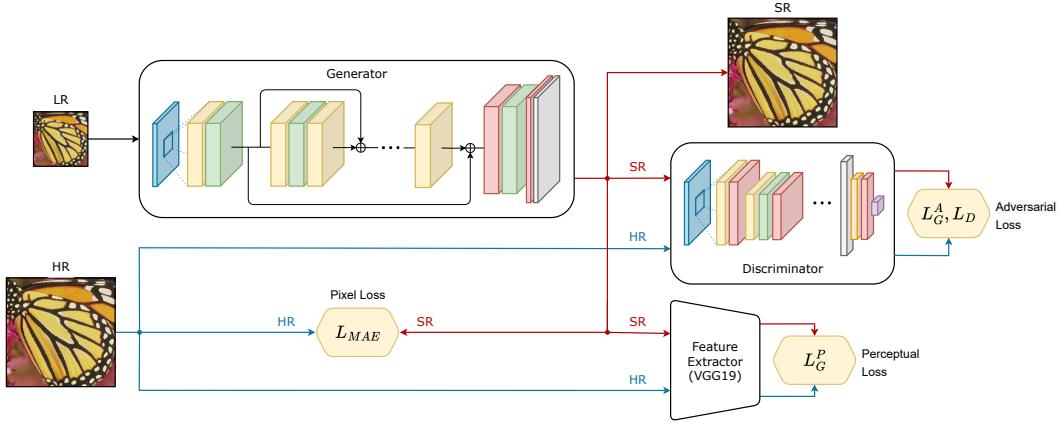


Fig. 5.4 EdgeSRGAN Training Methodology.

5.1.3 Knowledge Distillation

Knowledge Distillation (KD) has gained increasing interest in deep learning due to its ability to transfer knowledge from larger models to simpler ones efficiently. In particular, KD has been applied in some SISR works to compress the texture reconstruction capability of cumbersome models and obtain efficient real-time networks. However, to my knowledge, KD has never been applied to GAN SISR models. For this reason, I adapt an existing technique, developed for SISR and called Feature Affinity-based Knowledge Distillation (FAKD) [47], to the GAN training approach. The FAKD methodology transfers second-order statistical information to the student by aligning feature affinity matrices at different layers of the networks. This constraint helps address the issue that regression problems generate unbounded solution spaces. Indeed, most of the KD methods have only tackled classification tasks. Given a layer l of the network, the feature map F_l

extracted from that layer (after the activation function) has the following shape:

$$F_l \sim (B, C, W, H) \quad (5.6)$$

where B is the batch size, C is the number of channels, W and H are the width and the height of the tensor. I first flatten the tensor along the last two components, obtaining the three-dimensional feature map

$$F_l \sim (B, C, WH) \quad (5.7)$$

which now holds all the spatial information along a single axis. I define the affinity matrix A_l as the product

$$A_l = \tilde{F}_l^\top \cdot \tilde{F}_l \quad (5.8)$$

where \cdot is the matrix multiplication operator and the transposition \top swaps the last two dimensions of the tensor. \tilde{F}_l is the normalized feature map, obtained as

$$\tilde{F}_l = \frac{F_l}{\|F_l\|_2} \quad (5.9)$$

Differently from [47], the norm is calculated for the whole tensor and not only along the channel axis. Moreover, I find better convergence using the Euclidean norm instead of its square. In this way, the affinity matrix has a shape

$$A_l \sim (B, WH, WH) \quad (5.10)$$

and the total distillation loss L_{Dist} becomes

$$L_{\text{Dist}} = \frac{1}{N_L} \left(\sum_{l=1}^{N_L} \|A_l^T - A_l^S\|_1 \right) + \lambda \|y_{\text{SR}}^T - y_{\text{SR}}^S\|_1 \quad (5.11)$$

where N_L is the number of distilled layers. Unlike [47], I sum the loss across all tensor dimensions and average the results obtained for different layers. These modifications experimentally lead to better training convergence. I also add another loss component, weighted by λ , which optimizes the model to generate outputs that are close to the teacher's. In my experimentation, the distillation loss

is added to the overall training loss, weighted by the parameter γ . The overall distillation scheme is summarized in Fig. 5.5.

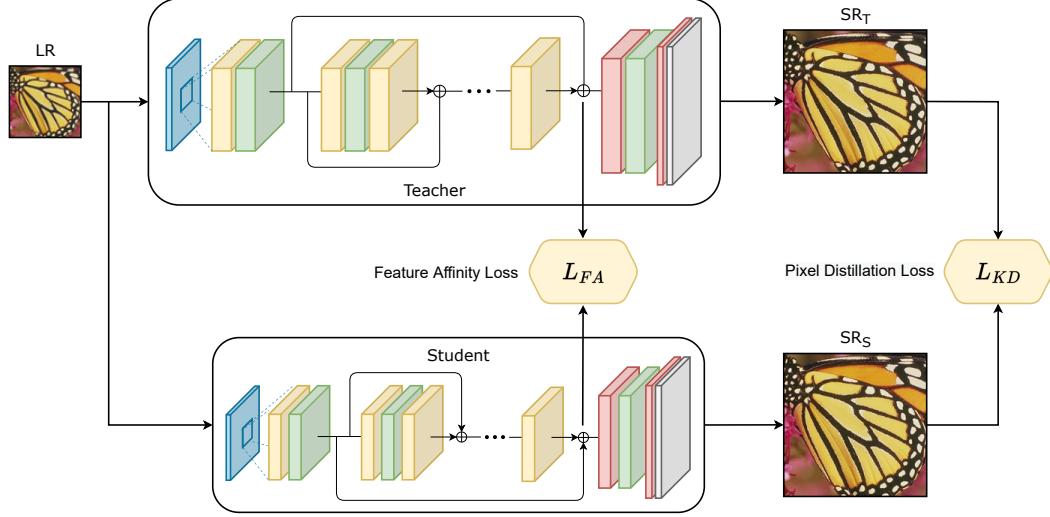


Fig. 5.5 EdgeSRGAN Distillation Process.

5.1.4 Model Interpolation

Following the procedure proposed in [59], I adopt a flexible and effective strategy to obtain a tunable trade-off between a content-oriented and GAN-trained model. This feature can be particularly useful for real-time applications, enabling the SISR network to quickly adapt to users' needs. Indeed, some real-world scenarios may require better perceptual quality, for example, when a human pilot must remotely control a robot. On the other hand, when images are used to directly feed perception, autonomous navigation, and mapping algorithms, higher pixel fidelity might be beneficial. To achieve this goal, I linearly interpolate model weights layer-by-layer, according to the following formula:

$$\theta_G^{\text{Int}} = \alpha \theta_G^{\text{PSNR}} + (1 - \alpha) \theta_G^{\text{GAN}} \quad (5.12)$$

where θ_G^{Int} , θ_G^{PSNR} , and θ_G^{GAN} are the weights of the interpolated model, the PSNR model, and the GAN fine-tuned model, respectively. $\alpha \in [0, 1]$ is the interpolation weight. I report both qualitative and quantitative interpolation results for EdgeSRGAN in Section 5.2.3. I avoid the alternative technique of directly

interpolating network outputs: applying this method in real time would require running two models simultaneously. Moreover, Wang *et al.* [59] report that this approach does not guarantee an optimal trade-off between noise and blur.

5.1.5 Quantization

To further reduce the inference latency of EdgeSRGAN, I apply optimization methods to the model, which reduces computational effort at the cost of a slight loss in performance. Several techniques have been developed to increase model efficiency in the past few years [64], from which the employed method is chosen. I reduce the number of bits used to represent network parameters and activation functions with TFLite¹. This strategy significantly enhances efficiency, albeit with a moderate impact on performance. I quantize weights, activations, and math operations through scale and zero-point parameters following the methodology presented by Jacob *et al.* [64]:

$$r = S(q - Z) \quad (5.13)$$

where r is the original floating-point value, q is the quantized integer value, and S and Z are the quantization parameters (scale and zero point). A fixed-point multiplication approach is employed to address the non-integer scale of S . This strategy drastically reduces memory and computational demands due to the high efficiency of integer computations on microcontrollers. For my experimentation, I deploy the quantized model on a Google Coral edge TPU USB Accelerator².

5.2 Experiments

5.2.1 Experimental Setting

In this section, I define the implementation details of my method and the procedure I followed to train and validate the efficiency of EdgeSRGAN. As previously done by most GAN-based SISR works, I train the network on the high-quality

¹tensorflow.org/lite

²coral.ai

Method	Scale	Params	Framerate (80×60) [fps]		Framerate (160×120) [fps]	
			CPU	EdgeTPU	CPU	EdgeTPU
SwinIR [32]	$\times 4$	11.9M	0.25 ± 0.01	-	0.06 ± 0.01	-
ESRGAN [59]		16.7M	0.40 ± 0.01	-	0.10 ± 0.01	-
Real-ESRGAN [36]		16.7M	0.44 ± 0.01	-	0.11 ± 0.01	-
SRGAN [34]		1.5M	2.70 ± 0.08	-	0.95 ± 0.02	-
AGD [35]		0.42M	3.17 ± 0.12	-	0.88 ± 0.01	-
EdgeSRGAN		0.66M	10.26 ± 0.11	140.23 ± 1.50	2.66 ± 0.02	10.63 ± 0.03
EdgeSRGAN-tiny		0.09M	37.99 ± 1.42	203.16 ± 3.03	11.76 ± 0.20	20.57 ± 0.05
SwinIR [32]	$\times 8$	12.0M	0.23 ± 0.01	-	0.06 ± 0.01	-
EdgeSRGAN		0.71M	7.70 ± 0.31	14.26 ± 0.06	1.81 ± 0.04	-
EdgeSRGAN-tiny		0.11M	24.53 ± 1.28	41.55 ± 0.38	5.81 ± 0.29	-

Table 5.1 Framerate comparison of different methods for $\times 4$ and $\times 8$ upsampling, with two different input resolutions (80×60 and 160×120). The results are presented as the mean and standard deviation of 10 independent experiments, each comprising 100 predictions. The current content-oriented SISR state-of-the-art, SwinIR [32], is reported as a reference. Real-time and over-real-time framerates are in blue and red, respectively. The proposed solution is the only one compatible with EdgeTPU devices and allows reaching real-time performance in both conditions.

DIV2K dataset [65] with a scaling factor of 4. The dataset contains 800 training samples and 100 validation samples. I train my model with input images of size 24x24 pixels, selecting random patches from the training set. I apply data augmentation by randomly flipping or rotating the images by multiples of 90° . I adopt a batch size of 16.

For the standard EdgeSRGAN implementation, I chose $N = 8$, $F = 64$, $K = 3$, and $D = 1024$, obtaining a generator with approximately 660,000 parameters and a discriminator of over 23 million (due to the fully connected head). The discriminator is built with $F = 64$, $K = 3$, $D = 512$, and with a coefficient for LeakyReLU $\alpha = 0.2$. I first train EdgeSRGAN pixel-wise for 5×10^5 steps using the Adam optimizer with a constant learning rate of 1×10^{-4} . Then, the model is fine-tuned in the adversarial setting described in Section 5.1 for 1×10^5 steps. The Adam optimizer is used for both the generator and the discriminator, with a learning rate of 1×10^{-5} . This learning rate is further divided by 10 after 5×10^4 steps. For the loss function, I set $\xi = 1 \times 10^{-3}$ and $\eta = 0$.

To obtain an even smaller model for my distillation experiments, I built EdgeSRGAN-tiny by choosing $N = 4$, $F = 32$, and $D = 256$. I further shrink the size of the discriminator by eliminating the first compression stage (B1)

from each block (see Fig. 5.3). In this configuration, I also remove the batch normalization layer from the first B2 block to be coherent with the larger version. The obtained generator and discriminator contain approximately 90,000 and 2.75 million parameters. The pre-training procedure is the one described for EdgeSRGAN, while the adversarial training is performed with the additional distillation loss ($\gamma = 1 \times 10^{-2}$, $\lambda = 1 \times 10^{-1}$) of Eq. (5.11). EdgeSRGAN is used as a teacher model, distilling its layers 2, 5, and 8 into EdgeSRGAN-tiny’s layers 1, 2, and 4. The model is trained with a learning rate of 1×10^{-4} , which is further divided by 10 after 5×10^4 steps. For the loss function, I set $\xi = 1 \times 10^{-3}$ and $\eta = 0$.

Finally, I create a third version of my model to upscale images with a factor of 8. To do so, I modify the first transpose convolution layer of EdgeSRGAN and EdgeSRGAN-tiny to have a stride of 4 instead of 2, while leaving the rest of the architecture unchanged. The training procedure for these models is analogous to the ones used for the $\times 4$ models, with the main difference of adding a pixel-based component to the adversarial loss by posing $\eta = 1 \times 10^2$.

The optimal training hyperparameters are determined by running a random search and selecting the best-performing models on the DIV2K validation set. During GAN training, I use PSNR to validate the models during content-based loss optimization and LPIPS [2] (with AlexNet backbone). I utilize TensorFlow 2 and a workstation equipped with 64 GB of RAM, an Intel i9-12900K CPU, and an NVIDIA 3090 RTX GPU to conduct all the training experiments.

5.2.2 Real-time Performance

Since the main focus of the proposed methodology is to train an optimized SISR model that can be efficiently run at the edge in real-time, I first report an inference speed comparison between the proposed method and other literature methodologies. All the results are presented in Table 5.1 as the mean and standard deviation of 10 independent experiments, each consisting of 100 predictions. I compare the proposed methodology with other GAN-based methods [34, 59, 36, 35] and with the current state-of-the-art in content-oriented SISR, SwinIR [32]. Since the original implementations of the GAN-based solutions only consider $\times 4$ upsampling, for the $\times 8$ comparison, I only report the results of SwinIR. I

select two different input resolutions for the experimentation, (80×60) and (160×120) , in order to target (320×240) and (640×480) resolutions for $\times 4$ upsampling and (640×480) and (1280×960) for $\times 8$ upsampling, respectively. This choice is justified because (640×480) is a standard resolution provided by most cameras' native video stream. I also report the number of parameters for all the models.

For all the methods considered, I measure the CPU timings using the model format of the original implementation (PyTorch or TensorFlow) on a MacBook Pro with an Intel i5-8257U processor. The concept of real-time performance strongly depends on the downstream task. For robotic monitoring and teleoperation, I consider 10 fps as the minimum real-time framerate, considering “over-real-time” everything above 30 fps, which is the standard framerate for most commercial cameras. The proposed methodology outperforms all other methods in inference speed, achieving real-time performance on the CPU in nearly all testing conditions. It is worth noting that AGD is specifically designed to reduce latency for GAN-based SR and has fewer parameters than EdgeSRGAN; however, it still fails to achieve real-time performance without a GPU.

Additionally, I report the frame rate of the EdgeSRGAN INT8-quantized models on an EdgeTPU Coral USB Accelerator. The proposed solution is the only one compatible with such devices and allows reaching over-real-time performance for (80×60) input resolution. It must be emphasized that the $\times 8$ models with an (160×120) input resolution cannot target the EdgeTPU device due to memory limitations.

Method	Set5 [3]			Set14 [66]			BSD100 [67]			Manga109 [68]			Urban100 [69]		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Bicubic	28.632	0.814	0.340	26.212	0.709	0.441	26.043	0.672	0.529	25.071	0.790	0.318	23.236	0.661	0.473
SwinIR [32]	32.719	0.902	0.168	28.939	0.791	0.268	27.834	0.746	0.358	31.678	0.923	0.094	27.072	0.816	0.193
SRGAN [34]	32.013	0.893	0.191	28.534	0.781	0.294	27.534	0.735	0.396	30.292	0.906	0.111	25.959	0.782	0.244
ESRGAN [59] \dagger	32.730	0.901	0.181	28.997	0.792	0.275	27.838	0.745	0.371	31.644	0.920	0.097	27.028	0.815	0.201
AGD [35]	31.708	0.889	0.178	28.311	0.775	0.291	27.374	0.729	0.385	29.413	0.897	0.118	25.506	0.767	0.250
EdgeSRGAN	31.729	0.889	0.191	28.303	0.774	0.301	27.359	0.728	0.405	29.611	0.897	0.120	25.469	0.764	0.266
EdgeSRGAN-tiny	30.875	0.873	0.204	27.796	0.761	0.320	26.999	0.717	0.418	28.233	0.871	0.163	24.695	0.733	0.325

Table 5.2 Quantitative comparison of different methods for content-oriented $\times 4$ upsampling. The current SISR state-of-the-art method, SwinIR [32], and the bicubic baseline are reported as references. \uparrow : higher is better, \downarrow : lower is better, \dagger : trained on DIV2K [65] + Flickr2K [70] + OST [71]

Model	Set5 [3]			Set14 [66]			BSD100 [67]			Manga109 [68]			Urban100 [69]		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Bicubic	28.632	0.814	0.340	26.212	0.709	0.441	26.043	0.672	0.529	25.071	0.790	0.318	23.236	0.661	0.473
SwinIR [32]	32.719	0.902	0.168	28.939	0.791	0.268	27.834	0.746	0.358	31.678	0.923	0.094	27.072	0.816	0.193
SRGAN [34]	29.182	0.842	0.094	26.171	0.701	0.172	25.447	0.648	0.206	27.346	0.860	0.076	24.393	0.728	0.158
ESRGAN[59]†	30.459	0.852	0.083	26.283	0.698	0.139	25.288	0.649	0.168	28.478	0.860	0.065	24.350	0.733	0.125
Real-ESRGAN [36]†	26.617	0.807	0.169	25.421	0.696	0.234	25.089	0.653	0.282	25.985	0.836	0.149	22.671	0.686	0.214
AGD [35]	30.432	0.861	0.097	27.276	0.739	0.160	26.219	0.688	0.214	28.163	0.870	0.076	24.732	0.743	0.170
EdgeSRGAN	29.487	0.837	0.095	26.814	0.715	0.176	25.543	0.644	0.210	27.679	0.855	0.081	24.268	0.716	0.170
EdgeSRGAN-tiny	28.074	0.803	0.146	26.001	0.702	0.242	25.526	0.658	0.292	25.655	0.804	0.140	23.332	0.672	0.269
EdgeSRGAN-tiny‡	29.513	0.841	0.132	26.950	0.727	0.220	26.174	0.673	0.282	27.106	0.845	0.130	24.117	0.704	0.249

Table 5.3 Quantitative comparison of different methods for visual-oriented $\times 4$ upsampling. The current SISR state-of-the-art method, SwinIR [32], and the bicubic baseline are reported as references. \uparrow : higher is better, \downarrow : lower is better. †: trained on DIV2K [65] + Flickr2K [70] + OST [71].

Model	Set5 [3]			Set14 [66]			BSD100 [67]			Manga109 [68]			Urban100 [69]		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Bicubic	24.526	0.659	0.533	23.279	0.568	0.628	23.727	0.546	0.713	21.550	0.646	0.535	20.804	0.515	0.686
SwinIR [32]	27.363	0.787	0.284	25.265	0.652	0.428	24.984	0.606	0.537	25.246	0.800	0.229	23.023	0.646	0.375
EdgeSRGAN	26.462	0.755	0.321	24.507	0.626	0.460	24.590	0.587	0.567	23.840	0.753	0.294	22.001	0.592	0.463
EdgeSRGAN-tiny	26.025	0.732	0.359	24.286	0.615	0.488	24.383	0.577	0.591	23.154	0.723	0.353	21.680	0.570	0.520
EdgeSRGAN	25.307	0.680	0.228	23.585	0.558	0.348	23.547	0.514	0.386	22.719	0.680	0.257	21.102	0.522	0.374
EdgeSRGAN-tiny	25.523	0.693	0.280	23.976	0.589	0.399	24.163	0.557	0.475	22.874	0.695	0.317	21.477	0.546	0.459

Table 5.4 Quantitative performance of the proposed method for $\times 8$ upsampling. The current state-of-the-art SISR (SwinIR [32]) and bicubic methods are reported as references. \uparrow : higher is better, \downarrow : lower is better.

5.2.3 Super-Resolution Results

To present quantitative results on image super-resolution, I refer to content-oriented SR for models trained with content-based loss only and visual-oriented SR for models trained with adversarial and perceptual losses. Content-based loss (mean absolute or squared error) maximizes PSNR and SSIM, while adversarial and perceptual losses maximize visual quality. I test EdgeSRGAN models on five benchmark datasets (Set5 [3], Set14 [66], BSD100 [67], Manga109 [68], and Urban100 [69]) measuring PSNR, SSIM, and LPIPS. I follow the standard procedure for SISR adopted in [32], where the metrics are computed on the luminance channel Y of the YCbCr converted images. Also, S pixels are cropped from each image border, where S is the model scale factor.

Table 5.2 and Table 5.3 show the comparison with other methods for content-oriented and visual-oriented $\times 4$ SR, respectively. I report results of other GAN-based methodologies [34, 59, 36, 35] as well as the current content-oriented SOTA SwinIR [32] and bicubic baseline, as reference. Unlike what is usually

found in literature, I refer to the OpenCV³ bicubic resize implementation instead of the one present in MATLAB. For visually oriented SR, I also report the results of the distilled tiny model, EdgeSRGAN-tiny². The proposed method achieves competitive results in all metrics, albeit with some degradation for small models due to the considerable weight reduction. The distillation method aids EdgeSRGAN-tiny training by transferring knowledge from the standard model and mitigating degradation resulting from the reduced number of parameters. Note that ESRGAN and RealESRGAN are trained on Flickr2K [70], and OST [71] datasets in addition to DIV2K. Table 5.4 reports results of the $\times 8$ models, together with SwinIR and bicubic. Additionally, in this case, the proposed models achieve competitive results, and knowledge distillation helps mitigate performance degradation in the tiny model. As a final qualitative evaluation, Fig. 5.6 compares the super-resolved images obtained by EdgeSRGAN with the considered state-of-the-art solutions. My model yields comparable results, highlighting more texture and detail than networks trained with pixel loss (L_{MAE}), while remaining faithful to the ground truth image.

Model Interpolation

I report the results of network interpolation on the benchmark datasets in Fig. 5.8. I consider α values between 0 and 1 with a step size of 0.1, where 0 implies a fully visual-oriented model and 1 a fully content-oriented one. All results refer to the standard EdgeSRGAN model for $\times 4$ upsampling. This procedure effectively demonstrates how choosing the desired trade-off between content-oriented and visually oriented SR is possible simply by adjusting the interpolation weight α . An increase in the weight value causes an improvement of the content-related metrics PSNR and SSIM and a worsening of the perceptual index LPIPS. This behavior holds for all the test datasets, validating the proposed approach. This procedure can be easily carried out in a real-time application and only requires computing the interpolated weights once. Thus, it does not affect the inference speed in any way. For an additional visual evaluation, Fig. 5.7 reports the outputs obtained for increasing values of α on a random dataset sample.

³docs.opencv.org

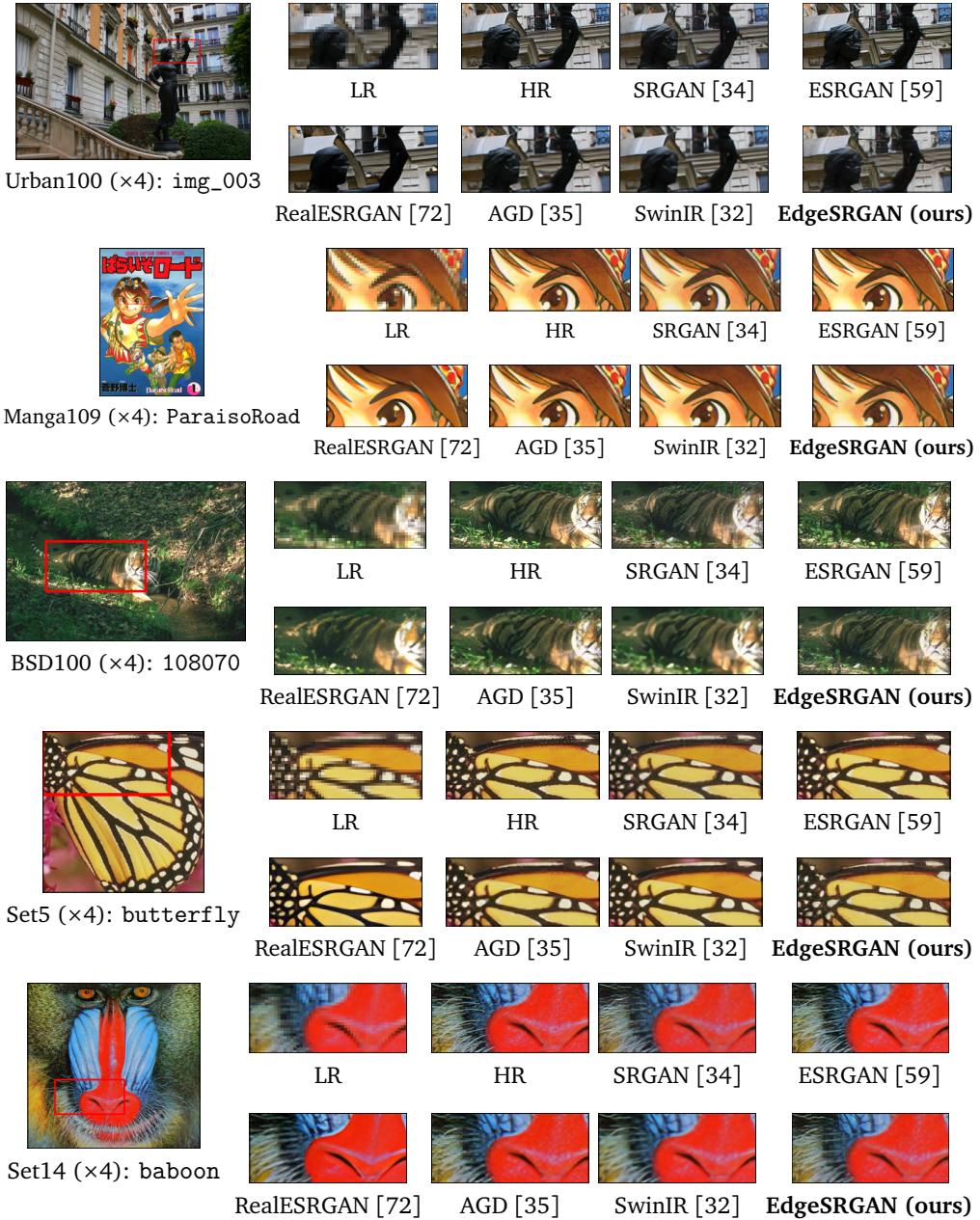


Fig. 5.6 Visual comparison of bicubic image SR ($\times 4$) methods on random samples from the considered datasets. EdgeSRGAN achieves results that are comparable to state-of-the-art solutions with $\sim 10\%$ of the weights.

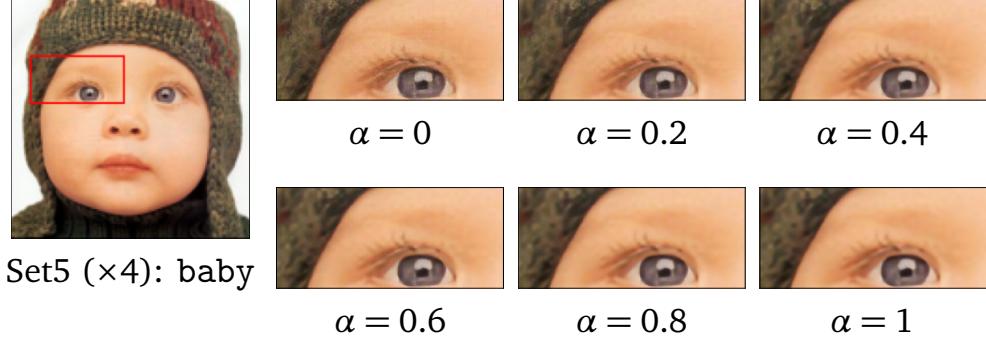


Fig. 5.7 Visual comparison of interpolated EdgeSRGAN for different values of α . Values closer to 1 generate outputs focused on content fidelity, while small values go towards visually pleasing results.

Model Quantization

To target edge TPU devices and reach over-real-time inference results, I follow the quantization scheme of Eq. (5.13) for both weights and activations to obtain a full-integer model. Since quantized models must have a fixed input shape, I generate a full-integer network for each input shape of the testing samples. I use the 100 images from the DIV2K validation set as a representative dataset to calibrate the quantization algorithm. I refer to the INT8-quantized standard model as EdgeSRGANi8. As for the tiny model, I optimize the distilled network EdgeSRGANi8-tiny*. Results for the visual-oriented optimized models are shown in Table 5.5. Due to the full-integer models' reduced activation and weight, I experience a significant increase in inference speed up to over-real-time at the cost of degradation in SR performance. All the proposed quantized models still outperform the bicubic baseline on the perceptual index LPIPS and therefore represent a good option for applications that require extremely fast inference. Different models for visual-oriented $\times 4$ upsampling are compared in Fig. 5.1. I compare LPIPS performance on the Set5 dataset with the framerate.

Model	Scale	Set5 [3]			Set14 [66]			BSD100 [67]			Manga109 [68]			Urban100 [69]		
		PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
EdgeSRGANi8	$\times 4$	27.186	0.721	0.209	24.714	0.475	0.342	23.675	0.484	0.438	25.601	0.712	0.221	22.802	0.580	0.341
EdgeSRGANi8-tiny*		27.330	0.710	0.257	24.807	0.562	0.390	23.837	0.485	0.481	25.299	0.696	0.286	22.580	0.538	0.454
EdgeSRGANi8	$\times 8$	24.433	0.602	0.312	22.846	0.477	0.440	22.609	0.422	0.492	22.227	0.603	0.342	20.525	0.433	0.499
EdgeSRGANi8-tiny		24.956	0.642	0.333	23.487	0.532	0.461	23.591	0.494	0.544	22.445	0.632	0.386	21.125	0.489	0.548

Table 5.5 Quantitative performance of the full-integer quantized models for $\times 4$ and $\times 8$ visual-based SR. \uparrow : higher is better, \downarrow : lower is better.

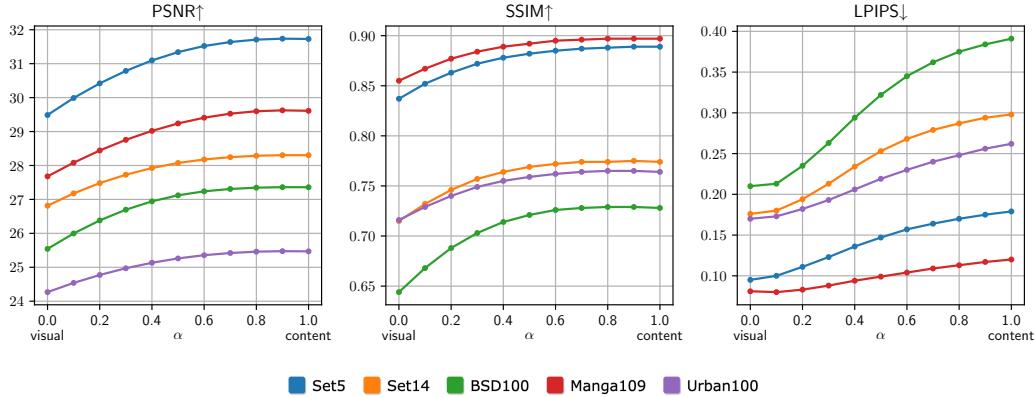


Fig. 5.8 EdgeSRGAN network interpolation results on the benchmark datasets for $\times 4$ upsampling. Changing the network interpolation weight α , it is possible to select the desired trade-off between content-oriented and visual-oriented SR. \uparrow : higher is better, \downarrow : lower is better.

5.2.4 Ablation Study

To further verify the effectiveness of my model for real-time super-resolution, I conduct an ablation study to analyze the effect of my architectural design choices. In particular, I benchmark EdgeSRGAN at four progressive steps, reporting fidelity, perceptual performance, and inference speed. The steps I consider are the following:

1. Reducing the number of residual blocks N ;
2. Replacing the Pixel Shuffle upsampling stage with Transpose Convolutions;
3. Removing Batch Normalization;
4. Replacing PReLU activations with ReLU.

The last step corresponds to the final version of EdgeSRGAN. For each step of the model, I use the same training procedure described in Section 5.1.2 and measure the inference speed on the CPU at input resolutions of (80x60) and (160x120). All the results are reported in Table 5.6. The experimentation confirms that each compression step gains substantial inference speed by trading minimal perceptual quality. Overall, I observe a 3.7% LPIPS perceptual quality improvement and a 280% inference speed increase.

Model	Params	Set5			Set14			BSD100			Manga100			Urban100			Inference Speed (fps)	
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	80x60	160x120
SRGAN	1.5M	29.18	0.842	0.094	26.17	0.701	0.172	25.45	0.648	0.206	27.35	0.860	0.076	24.39	0.728	0.158	2.00 ± 0.03	0.48 ± 0.01
$N = 8$	956k	29.38	0.839	0.088	26.55	0.703	0.170	25.08	0.628	0.207	27.49	0.852	0.085	24.21	0.718	0.168	2.47 ± 0.01	0.62 ± 0.01
TransposeConv	663k	28.98	0.829	0.113	26.46	0.706	0.204	25.25	0.641	0.243	26.72	0.833	0.116	23.66	0.689	0.214	9.16 ± 0.31	2.52 ± 0.03
No BatchNorm	661k	29.40	0.838	0.105	26.65	0.709	0.194	25.09	0.630	0.236	27.54	0.851	0.091	24.01	0.707	0.191	9.91 ± 0.16	2.56 ± 0.06
ReLU	661k	29.49	0.837	0.095	26.81	0.715	0.176	25.54	0.644	0.210	27.68	0.855	0.081	24.27	0.716	0.170	10.26 ± 0.11	2.66 ± 0.02

Table 5.6 Results of the ablation study conducted on EdgeSRGAN at four different steps. The last step corresponds to the final model. Overall, I observe a 3.7% LPIPS perceptual quality improvement and a 280% inference speed increase. ↑: higher is better, ↓: lower is better.

5.2.5 Image Transmission for Mobile Robotics

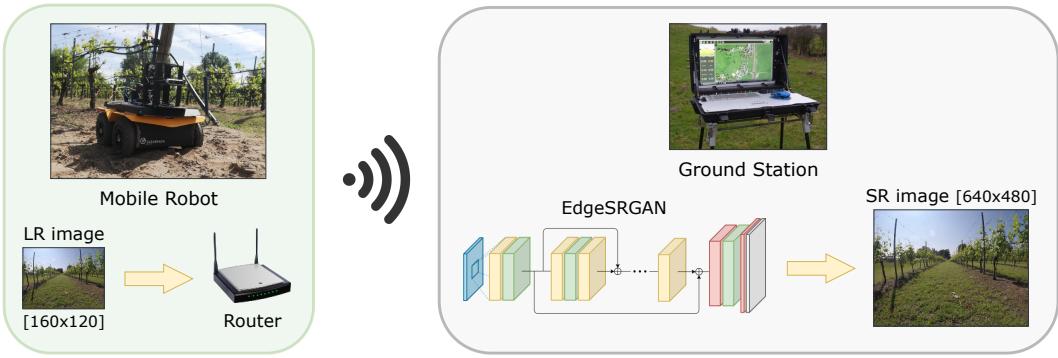


Fig. 5.9 Efficient image transmission system with EdgeSRGAN for mobile robotic applications in outdoor environments.

My real-time SISR can provide competitive advantages in various practical engineering applications. In this section, I target a specific use case of mobile robotics, proposing my EdgeSRGAN system as an efficient deep learning-based solution for real-time image transmission. Indeed, robot remote control in unknown terrains requires a reliable visual data transmission at a satisfying frame rate, preserving robustness even in bandwidth-degraded conditions. This requirement is particularly relevant for high-speed platforms and UAVs. Dangerous or delicate tasks, such as tunnel exploration, inspection, or open-space missions, all require an available visual stream for human supervision, regardless of the autonomy level of the platform. Over the last few years, the robotics community has focused on developing globally shared solutions for robot software and architectures, as well as handling data communications between multiple platforms and devices. ROS2 [73] is the standard operating system for robotic platforms. It is a middleware based on the Data Distribution System (DDS) protocol, where application nodes communicate with each other through a topic using a publisher/subscriber

mechanism. However, despite recent attempts to improve the reliability and efficiency of message and data packet communications between nodes and platforms, heavier data transmission, such as image streaming, remains unoptimized and unreliable.

The typical practical setting for robot teleoperation and exploration in unknown environments comprises a ground station and a rover connected to the same wireless network. As shown in Fig. 5.9, I adopted this ground station configuration to test the transmission of images through a ROS2 topic, as should be done in any robotic application to stream what the robot sees or to receive visual data and feed perception and control algorithms for autonomous navigation and mapping. For this experiment, I use both an Intel RealSense D435i camera⁴ and a Logitech C920 webcam⁵ mounted on a Clearpath Jackal robot⁶, together with a Microhard BulletPlus⁷ router for image transmission. The available image resolutions with RealSense cameras, the standard RGBD sensors for visual perception in robotics, are (320×240) and (640×480) , whereas the framerate typically varies between 15 and 30 fps.

Despite the absence of substantial bandwidth limitations, transmission delays, or partial loss of packets, the maximum resolution and framerate allowed by ROS2 communication are extremely low: I find that at 30 fps, the maximum transmissible resolution for RGB is (120×120) with a bandwidth of 20 Mb/s while reducing the framerate to 5 fps the limit is (320×240) . This strict trade-off between frame rate and resolution hinders the high-speed motion of a robotic platform during a mission, increasing the risk of collision due to reduced scene supervision. Even selecting *best effort* in the Quality of Service (QoS) settings, which manage the reception of packages through topics, the detected performances are always scarce.

Adopting my real-time Super-Resolution system ensures the timely arrival of RGB and depth images via ROS2. Thanks to the fast-inference performance of EdgeSRGAN, I can stream low-resolution images (80×60) at a high framerate (30 fps) and receive a high-resolution output: (320×240) with a $\times 4$ image upsampling and (640×480) with a $\times 8$ upsampling, showing a clear improve-

⁴intelrealsense.com/depth-camera-d435i

⁵logitech.com/en-en/products/webcams/c920-pro-hd-webcam

⁶clearpathrobotics.com/jackal-small-unmanned-ground-vehicle

⁷microhardcorp.com/BulletPlus-NA2

ment on standard performance. My system allows the ground station to access the streaming data through a simple ROS topic. Hence, it provides multiple competitive advantages in robotic teleoperation and autonomous navigation, as the human operator can directly utilize high-resolution images for remote control. Moreover, they can be used to feed computationally hungry algorithms, such as sensorimotor agents, visual odometry, or visual SLAM, which I may prefer to run on the ground station to conserve the robot’s constrained power resources and significantly boost the mission’s autonomy level. In Fig. 5.10, I report a qualitative comparison to highlight the effectiveness of EdgeSRGAN for real-world robotic scenarios. In particular, I consider apple monitoring, navigation in vineyards, drone surveillance for autonomous rovers, and tunnel inspection.

I also test video transmission performance in a more general framework to reproduce all the potential bandwidth conditions. I use the well-known video streaming library GStreamer⁸ to transmit video samples changing the available bandwidth. I progressively reduce the bandwidth from 10 Mbps to 10 kbps using the Wondershaper library⁹ and measure the framerate at the receiver side. I use 10 seconds of the standard video sample *smtpe* natively provided by GStreamer *videotestsrc* video source at 30 fps, and I encode it for transmission using MJPEG and H.264 video compression standards. The encoding is performed offline to ensure that all the available resources are reserved for transmission only. Indeed, most cameras provide hardware-encoded video sources, eliminating the need for software compression. To be consistent with the other experiments, I continue to use (640×480) and (320×240) as high resolutions and (160×120) and (80×60) as low resolutions. Each experiment is performed 10 times to verify consistency in the results.

Fig. 5.11 presents the average framerate achieved with different bandwidths. Streaming video directly, without any middleware such as ROS2, ensures higher transmission performance. However, as expected, streaming high-resolution images is impossible because of low bandwidth, and the framerate quickly drops to very low values, making it unsuitable for real-time applications. On the other hand, lower resolutions can be streamed with minimal frame drop, even with lower available bandwidths. H.264 compression shows the same behavior as MJPEG but shifts to lower bandwidths. Indeed, H.264 is more sophisticated and

⁸gstreamer.freedesktop.org

⁹github.com/magnific0/wondershaper

efficient, as it uses temporal frame correlation in addition to spatial compression. In a practical application with a specific bandwidth constraint, a suitable combination of a low-resolution video source and an SR model can be selected to meet the desired frame rate requirements on the available platform (CPU or edge TPU). This mechanism can also be dynamically activated and deactivated automatically, depending on the current connectivity, to avoid framerate drops and ensure smooth image transmission.

5.3 Conclusion

In this paper, I propose a novel Edge AI model for SISR exploiting the Generative Adversarial approach. Inspired by popular state-of-the-art solutions, I design EdgeSRGAN, which achieves comparable results while being an order of magnitude smaller in terms of the number of parameters. My model is 3 times faster than SRGAN, 30 times faster than ESRGAN, and 50 times faster than SwinIR while retaining similar or even better LPIPS performance. I applied knowledge distillation to EdgeSRGAN to gain additional inference speed and obtained an even smaller network (EdgeSRGAN-tiny), which gains an additional 4x speed with limited performance loss. Moreover, model quantization is used to optimize the model for execution on an Edge TPU. At the same time, network interpolation was implemented to allow potential users to balance the model output between pixel-wise fidelity and perceptual quality. Extensive experimentation on several datasets confirms the effectiveness of my model in terms of both performance and latency. Finally, I considered the application of my solution for robot teleoperation, highlighting the validity and robustness of EdgeSRGAN in various practical scenarios where the transmission bandwidth is limited.

After addressing the challenge of visual enhancement under strict resource and latency constraints with EdgeSRGAN, the next chapter extends the exploration of efficient deep learning to the domain of indoor robot localization. In particular, I focus on developing a compact and robust model for mitigating ultra-wideband range errors, further reinforcing the central objective of this thesis: designing deployable, high-performance AI solutions for real-world autonomous systems operating under uncertainty and computational constraints.

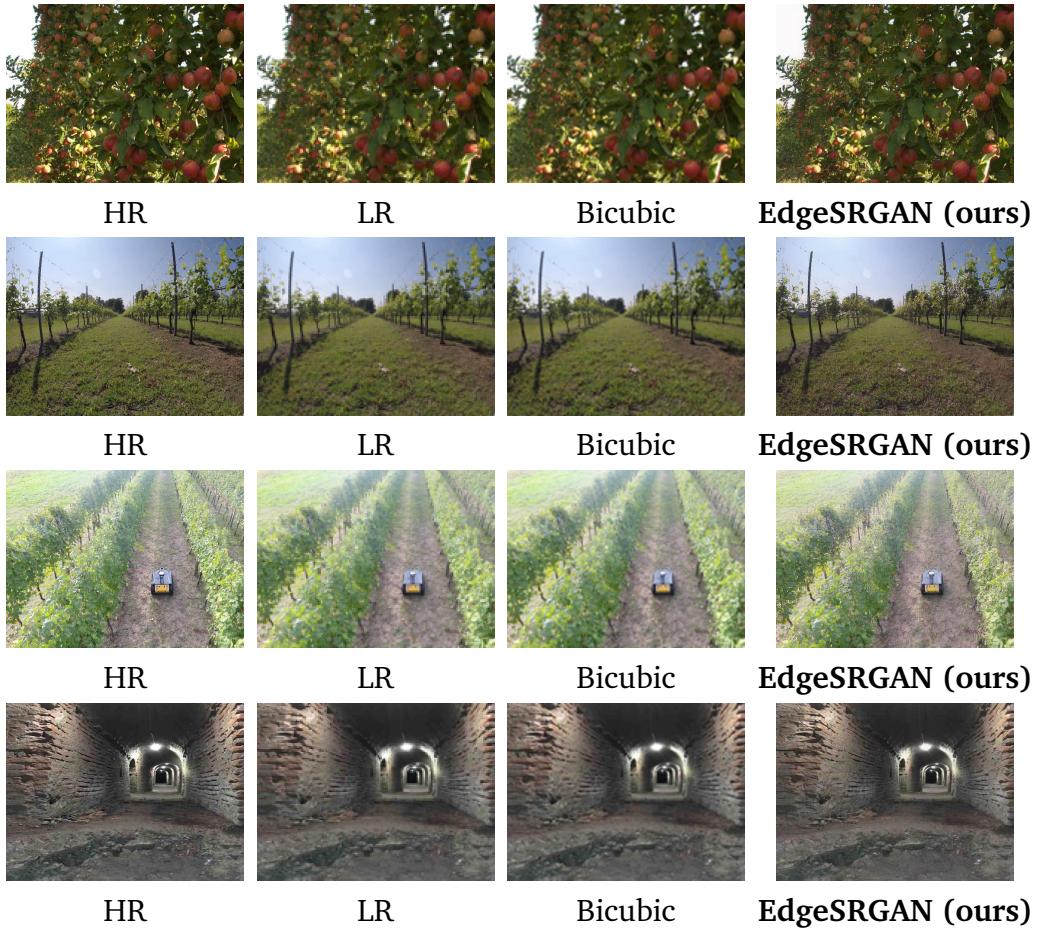


Fig. 5.10 Qualitative demonstration of applying EdgeSRGAN ($\times 4$) on real scenarios (zoom for more detail). From top to bottom: apple monitoring, vineyard navigation, drone surveillance for autonomous rovers, and tunnel inspection.

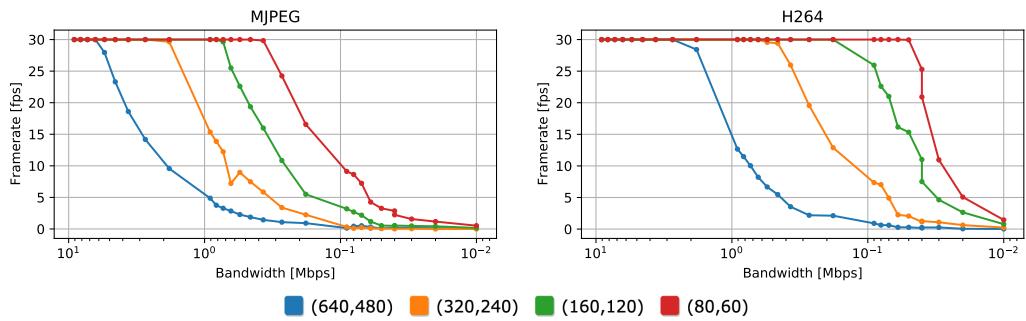


Fig. 5.11 Framerate results vs. bandwidth for video transmission at different input resolutions with MJPEG and H. 264 compression. Bandwidth is on a logarithmic scale.

Bibliography

- [1] S. Angarano, F. Salvetti, M. Martini, and M. Chiaberge, “Generative adversarial super-resolution at the edge with knowledge distillation,” *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106 407, 2023.
- [2] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, “The unreasonable effectiveness of deep features as a perceptual metric,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 586–595.
- [3] M. Bevilacqua, A. Roumy, C. Guillemot, and M.-L. A. Morel, “Low-complexity single-image super-resolution based on nonnegative neighbor embedding,” in *British Machine Vision Conference (BMVC)*, 2012.
- [4] R. de Queiroz Mendes, E. G. Ribeiro, N. dos Santos Rosa, and V. Grassi Jr, “On deep learning techniques to boost monocular depth estimation for autonomous navigation,” *Robotics and Autonomous Systems*, vol. 136, p. 103 701, 2021.
- [5] F. Zhu, Y. Zhu, V. Lee, X. Liang, and X. Chang, “Deep learning for embodied vision navigation: A survey,” *arXiv: 2108.04097*, 2021.
- [6] P. Roy and C. Chowdhury, “A survey of machine learning techniques for indoor localization and navigation systems,” *Journal of Intelligent & Robotic Systems*, vol. 101, no. 3, pp. 1–34, 2021.
- [7] X. Xiao, B. Liu, G. Warnell, and P. Stone, “Motion planning and control for mobile robot navigation using machine learning: A survey,” *Autonomous Robots*, pp. 1–29, 2022.
- [8] I. Lluvia, E. Lazcano, and A. Ansuategi, “Active mapping and robot exploration: A survey,” *Sensors*, vol. 21, no. 7, p. 2445, 2021.
- [9] D. S. Drew, “Multi-agent systems for search and rescue applications,” *Current Robotics Reports*, vol. 2, no. 2, pp. 189–200, 2021.
- [10] C. Yuan, B. Xiong, X. Li, X. Sang, and Q. Kong, “A novel intelligent inspection robot with deep stereo vision for three-dimensional concrete damage detection and quantification,” *Structural Health Monitoring*, vol. 21, no. 3, pp. 788–802, 2022.
- [11] F. Yin, “Inspection robot for submarine pipeline based on machine vision,” in *Journal of Physics: Conference Series*, IOP Publishing, vol. 1952, 2021, p. 022 034.

- [12] T. Rouček *et al.*, “Darpa subterranean challenge: Multi-robotic exploration of underground environments,” in *International Conference on Modelling and Simulation for Autonomous Systems*, Springer, 2019, pp. 274–290.
- [13] D. Tardioli *et al.*, “Ground robotics in tunnels: Keys and lessons learned after 10 years of research and experiments,” *Journal of Field Robotics*, vol. 36, no. 6, pp. 1074–1101, 2019.
- [14] T. Elmokadem and A. V. Savkin, “A method for autonomous collision-free navigation of a quadrotor uav in unknown tunnel-like environments,” *Robotica*, vol. 40, no. 4, pp. 835–861, 2022.
- [15] M. Martini, S. Cerrato, F. Salvetti, S. Angarano, and M. Chiaberge, “Position-agnostic autonomous navigation in vineyards with deep reinforcement learning,” in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2022, pp. 477–484.
- [16] D. Aghi, S. Cerrato, V. Mazzia, and M. Chiaberge, “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.
- [17] D. Li, J. Xu, H. He, and M. Wu, “An underwater integrated navigation algorithm to deal with dvl malfunctions based on deep learning,” *IEEE Access*, vol. 9, pp. 82 010–82 020, 2021.
- [18] J. E. Almanza-Medina, B. Henson, and Y. V. Zakharov, “Deep learning architectures for navigation using forward looking sonar images,” *IEEE Access*, vol. 9, pp. 33 880–33 896, 2021.
- [19] J. Chen and X. Ran, “Deep learning with edge computing: A review,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.
- [20] S. Angarano, V. Mazzia, F. Salvetti, G. Fantin, and M. Chiaberge, “Robust ultra-wideband range error mitigation with deep learning at the edge,” *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104 278, 2021.
- [21] D. Liu, H. Kong, X. Luo, W. Liu, and R. Subramaniam, “Bringing ai to edge: From deep learning’s perspective,” *Neurocomputing*, 2021.
- [22] C. Dong, C. C. Loy, K. He, and X. Tang, “Image super-resolution using deep convolutional networks,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.
- [23] J. Kim, J. K. Lee, and K. M. Lee, “Accurate image super-resolution using very deep convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 1646–1654.
- [24] B. Lim, S. Son, H. Kim, S. Nah, and K. Mu Lee, “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.

- [25] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, “Residual dense network for image super-resolution,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2472–2481.
- [26] J. Liu, J. Tang, and G. Wu, “Residual feature distillation network for lightweight image super-resolution,” in *European Conference on Computer Vision*, Springer, 2020, pp. 41–55.
- [27] Y. Zhang, K. Li, K. Li, L. Wang, B. Zhong, and Y. Fu, “Image super-resolution using very deep residual channel attention networks,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 286–301.
- [28] T. Dai, J. Cai, Y. Zhang, S.-T. Xia, and L. Zhang, “Second-order attention network for single image super-resolution,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11 065–11 074.
- [29] B. Niu *et al.*, “Single image super-resolution via a holistic attention network,” in *European conference on computer vision*, Springer, 2020, pp. 191–207.
- [30] J. Cao, Y. Li, K. Zhang, and L. Van Gool, “Video super-resolution transformer,” *arxiv*: 2106.06847, 2021.
- [31] H. Chen *et al.*, “Pre-trained image processing transformer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 299–12 310.
- [32] J. Liang, J. Cao, G. Sun, K. Zhang, L. Van Gool, and R. Timofte, “Swinir: Image restoration using swin transformer,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1833–1844.
- [33] I. Goodfellow *et al.*, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.
- [34] C. 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.
- [35] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, “Autogan-distiller: Searching to compress generative adversarial networks,” in *Proceedings of the 37th International Conference on Machine Learning*, 2020, pp. 3292–3303.
- [36] X. Wang, L. Xie, C. Dong, and Y. Shan, “Real-esrgan: Training real-world blind super-resolution with pure synthetic data,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 1905–1914.
- [37] Y. Li *et al.*, “Ntire 2022 challenge on efficient super-resolution: Methods and results,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 1062–1102.
- [38] M. S. Sajjadi, B. Scholkopf, and M. Hirsch, “Enhancenet: Single image super-resolution through automated texture synthesis,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 4491–4500.

- [39] P. N. Michelini, Y. Lu, and X. Jiang, “Edge-sr: Super-resolution for the masses,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022, pp. 1078–1087.
- [40] A. Aguinaldo, P.-Y. Chiang, A. Gain, A. Patil, K. Pearson, and S. Feizi, “Compressing gans using knowledge distillation,” *arXiv: 1902.00159*, 2019.
- [41] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.
- [42] G. Hinton, O. Vinyals, J. Dean, *et al.*, “Distilling the knowledge in a neural network,” *arxiv: 1503.02531*, 2015.
- [43] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “Fitnets: Hints for thin deep nets,” *arXiv: 1412.6550*, 2014.
- [44] S. Zagoruyko and N. Komodakis, “Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer,” *arXiv: 1612.03928*, 2016.
- [45] B. Heo, M. Lee, S. Yun, and J. Y. Choi, “Knowledge transfer via distillation of activation boundaries formed by hidden neurons,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3779–3787.
- [46] B. Heo, J. Kim, S. Yun, H. Park, N. Kwak, and J. Y. Choi, “A comprehensive overhaul of feature distillation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1921–1930.
- [47] Z. He, T. Dai, J. Lu, Y. Jiang, and S.-T. Xia, “Fakd: Feature-affinity based knowledge distillation for efficient image super-resolution,” in *2020 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2020, pp. 518–522.
- [48] Y. Zhang, H. Chen, X. Chen, Y. Deng, C. Xu, and Y. Wang, “Data-free knowledge distillation for image super-resolution,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7852–7861.
- [49] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” in *International Conference on Learning Representations*, 2017.
- [50] M. K. Zein, M. Al Aawar, D. Asmar, and I. H. Elhajj, “Deep learning and mixed reality to autocomplete teleoperation,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 4523–4529.
- [51] H. Hedayati, M. Walker, and D. Szafir, “Improving collocated robot teleoperation with augmented reality,” in *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, 2018, pp. 78–86.
- [52] P. Stotko *et al.*, “A vr system for immersive teleoperation and live exploration with a mobile robot,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 3630–3637.

- [53] S. Ooyama, H. Lu, T. Kamiya, and S. Serikawa, “Underwater image super-resolution using srcnn,” in *International Symposium on Artificial Intelligence and Robotics 2021*, SPIE, vol. 11884, 2021, pp. 177–182.
- [54] M. Islam, P. Luo, and J. Sattar, “Simultaneous enhancement and super-resolution of underwater imagery for improved visual perception,” in *Robotics*, M. Toussaint, A. Bicchi, and T. Hermans, Eds., ser. Robotics: Science and Systems, MIT Press Journals, 2020, ISBN: 9780992374761.
- [55] R. Wang, D. Zhang, Q. Li, X.-Y. Zhou, and B. Lo, “Real-time surgical environment enhancement for robot-assisted minimally invasive surgery based on super-resolution,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 3434–3440.
- [56] A. Brodie and N. Vasdev, “The future of robotic surgery,” *The Annals of The Royal College of Surgeons of England*, vol. 100, no. Supplement 7, pp. 4–13, 2018.
- [57] D. E. Martinez, W. Meinhold, J. Oshinski, A.-P. Hu, and J. Ueda, “Super resolution for improved positioning of an mri-guided spinal cellular injection robot,” *Journal of Medical Robotics Research*, vol. 6, no. 01n02, p. 2140002, 2021.
- [58] H. Bae, K. Jang, and Y.-K. An, “Deep super resolution crack network (srcnet) for improving computer vision-based automated crack detectability in in situ bridges,” *Structural Health Monitoring*, vol. 20, no. 4, pp. 1428–1442, 2021.
- [59] X. Wang *et al.*, “Esrgan: Enhanced super-resolution generative adversarial networks,” in *The European Conference on Computer Vision Workshops (ECCVW)*, 2018.
- [60] W. 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 (CVPR)*, 2016.
- [61] A. Odena, V. Dumoulin, and C. Olah, “Deconvolution and checkerboard artifacts,” *Distill*, 2016.
- [62] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, “Loss functions for image restoration with neural networks,” *IEEE Transactions on computational imaging*, vol. 3, no. 1, pp. 47–57, 2016.
- [63] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.
- [64] B. 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.
- [65] E. Agustsson and R. Timofte, “Ntire 2017 challenge on single image super-resolution: Dataset and study,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 126–135.

- [66] R. Zeyde, M. Elad, and M. Protter, “On single image scale-up using sparse-representations,” in *International conference on curves and surfaces*, Springer, 2010, pp. 711–730.
- [67] D. Martin, C. Fowlkes, D. Tal, and J. Malik, “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, IEEE, vol. 2, 2001, pp. 416–423.
- [68] Y. Matsui *et al.*, “Sketch-based manga retrieval using manga109 dataset,” *Multimedia Tools and Applications*, vol. 76, no. 20, pp. 21 811–21 838, 2017.
- [69] J.-B. Huang, A. Singh, and N. Ahuja, “Single image super-resolution from transformed self-exemplars,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5197–5206.
- [70] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, and L. Zhang, “Ntire 2017 challenge on single image super-resolution: Methods and results,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 114–125.
- [71] X. Wang, K. Yu, C. Dong, and C. C. Loy, “Recovering realistic texture in image super-resolution by deep spatial feature transform,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 606–615.
- [72] H. Chen *et al.*, “Real-world single image super-resolution: A brief review,” *Information Fusion*, vol. 79, pp. 124–145, 2022.
- [73] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, eabm6074, 2022.

Chapter 6

Robust Ultra-wideband Range Error Mitigation at the Edge

Original Paper: *S. Angarano, V. Mazzia, F. Salvetti, G. Fantin, and M. Chiaberge, “Robust Ultra-wideband Range Error Mitigation with Deep Learning at the Edge”, Engineering Applications of Artificial Intelligence, vol. 102, 2021.* [1]

Precise localization is at the core of several engineering systems, and due to its intrinsic scientific relevance, it has been extensively researched in recent years [2, 3]. Either outdoor or indoor applications could primarily benefit from it in fields as diverse as telecommunications [4], service robotics [5], healthcare [6], search and rescue [7], and autonomous driving [8]. Nevertheless, accurate positioning in non-line-of-sight (NLoS) conditions remains an open research problem. Multipath effects, reflections, refractions, and other propagation phenomena could easily lead to errors in the position estimation [9, 10, 11].

Ultra-wideband (UWB) is the state-of-the-art technology for wireless localization, rapidly growing in popularity [12], offering decimeter-level accuracy and increasingly smaller and cheaper transceivers [13]. With a bandwidth larger than 500 MHz and extremely short transmit pulses, UWB offers high temporal and spatial resolution and considerable multipath effect error mitigation compared to other radio-frequency technologies [14]. Nevertheless, UWB is still primarily affected by the NLoS condition, Fig. 6.1, in which the range estimates based on time-of-arrival (TOA) are typically positively biased [15, 16]. That is particularly true for indoor localization, where ranging errors introduced by multipath and

NLoS conditions can lead to significant deviations from the actual position [17]. So, robust and effective mitigation is necessary to prevent significant localization errors.

Several approaches have been proposed to address the NLoS problem. With many anchor nodes available, NLoS identification is the preferred choice. Indeed, once an NLoS anchor is identified, it can be easily eliminated from the pool of nodes used for the trilateration algorithm [18]. The majority of the proposed methodologies found in the literature utilize channel and waveform statistics [19, 20, 21], likelihood ratio tests or binary hypothesis tests [18, 22], and machine learning techniques. In the latter case, either hand-designed techniques, such as support vector machines (SVM) [23], Gaussian processes (GP) [24], or representation learning models have been investigated [25, 9].

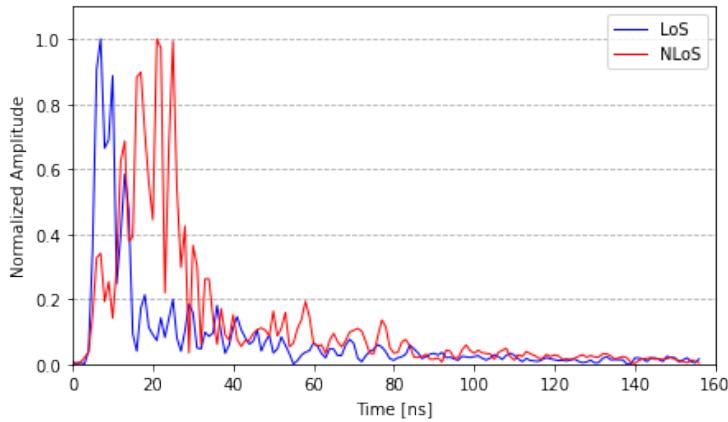


Fig. 6.1 LoS and NLoS CIRs with normalized amplitude in an indoor environment. In the NLoS case, the signal travels along many routes until it reaches the antenna. That makes the ToA estimation ambiguous.

Despite the simplicity of applying NLoS identification, [26], in almost all practical situations, there is not a sufficient number of anchors available to exclude some of them. So, most research community efforts focus on range and direct localization mitigation. Regarding the latter, even though studies have shown excellent position estimation in multipath environments [27, 28, 29], the collected training data is incredibly site-specific. Therefore, collecting data on one site does not allow the resulting model to be exploited in another location. On the other hand, range mitigation is far less site-specific and does not require a large amount of data to achieve satisfactory results [15]. Range error mitigation is mainly performed with similar techniques as NLoS identification [24, 30, 31, 32] and

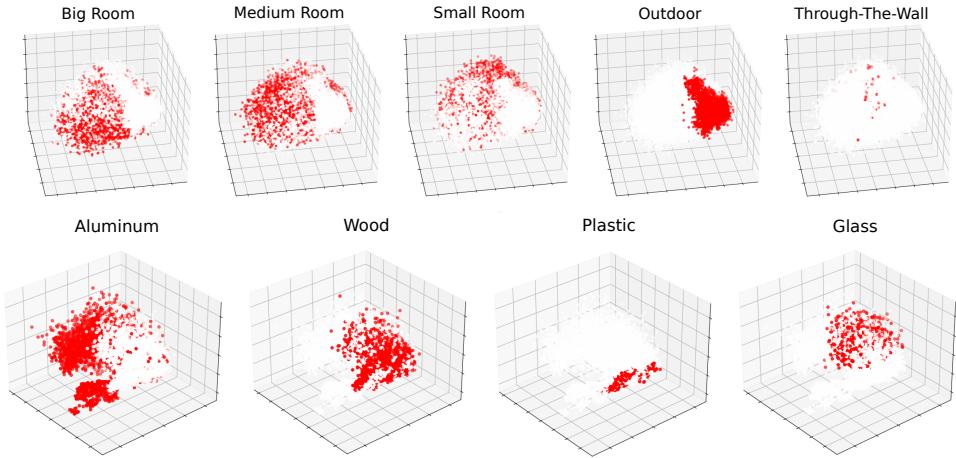


Fig. 6.2 Principal Component Analysis (PCA), projecting the original 157 CIR dimensions into a three-dimensional space. It is clear how rooms cover a similar data space, completely separated by the outdoor scenario. Moreover, the same applies to materials, where more dense molecular structures affect the signal differently.

also with more extreme conditions as error mitigation for through-the-wall (TTW) [33]. Moreover, following the advancements brought by representation learning techniques in many fields of research [34, 35, 36], Bregar & Mohorčič attempted to perform range error estimation directly from the channel impulse response (CIR) using a deep learning model [37]. Nevertheless, as a preliminary study, no relevance has been given to studying the network, optimizing it, and making it general to different environments.

This article [38, 1, 39] focuses on investigating a novel efficient deep learning model that performs an effective range error mitigation, using only the raw CIR signal at the edge. Indeed, range error mitigation should be performed directly on the platform where the UWB tag is attached. So, energy consumption and computational power play a decisive role in the significant applicability of our methodology. I adopt the latest advancements in deep learning architectural techniques [40, 41], and graph optimization [42] to improve ranging by nearly 45% (NLoS) and 34% (LoS) in an unknown indoor environment. The model absorbs up to barely 1 mJ of energy for a single inference. Moreover, our proposed methodology does not require additional NLoS identification models. Still, it can extract valuable features to estimate the correct range error directly from the CIR

in both LoS and NLoS states. All of our training and testing code and data are open source and publicly available¹ [43].

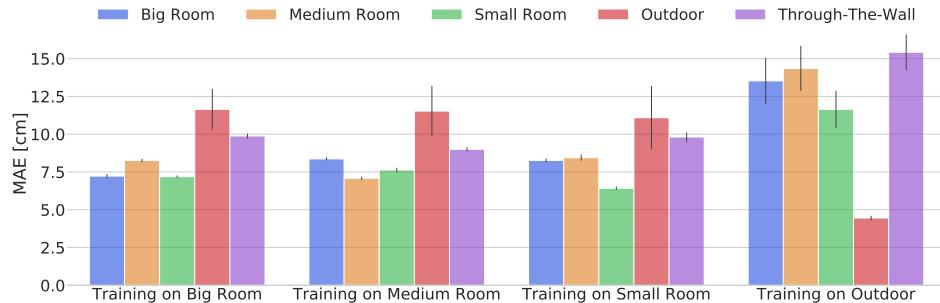


Fig. 6.3 Analysis of generalization capabilities in different environments of a generic representation learning model trained directly on the CIR waveform. Each bar represents the average of 20 independent MLP models. Results show that rooms with different sizes and configurations lead to very similar losses. Moreover, outdoor and TTW scenarios should be considered as separate settings and cannot be corrected without including appropriate samples in the training set.

6.1 Dataset

To visualize the distribution of the acquired instances in the data space, I exploit Principal Component Analysis (PCA) to project the 157 dimensions of each CIR signal into a three-dimensional space, saving most of the original variance. As shown in Fig. 6.2, the first analysis highlights the correlations between data points in the different analyzed environments. A prevalence of samples from the big room is found in the lower central part of the plot, while the medium and small room samples are more prevalent on the left and upper sides of the distribution, respectively. Nevertheless, it is clear how rooms cover a similar data space, which implies a potential transferability of statistics learned in different indoor environments. On the other hand, the outdoor set is situated entirely on the right side of the plot.

The same procedure is followed for materials, considering four object classes for clearness: aluminum plate, plastic bins, wooden door, and glass. In this case, a remarkable separation is noticeable, as the metal samples occupy the left part of the graph, while light objects, such as plastic, wood, and glass, occupy the right

¹zenodo.org/record/4399187

area. Moreover, the spatial distribution of timber occupies specific zones that exhibit distinct features compared to those of plastic and glass. The presented qualitative analysis provides initial visual evidence of the meaningfulness of the data and draws some conclusions on how a representation learning model could perform. For example, a generic model trained on measures taken with only plastic instances would more easily mitigate the error caused by wood and yield less accurate estimations for metal samples.

Finally, a Multilayer Perceptron (MLP) is trained and tested on dataset splits to assess the generalization capabilities of a generic representation learning model trained directly on the CIR waveform. After the method's validity is verified on the whole dataset, a series of tests are conducted to study the effect of different environments and obstacles on the models' performance. The network is trained on a specific dataset from the same setting or material and tested on other possible scenarios. In this way, it is possible to determine whether the approach holds absolute generality regarding such factors. Regarding environmental influence, as shown in Fig. 6.3, metrics indicate that rooms with different sizes and configurations result in minimal losses (less than 2 cm), compared to those caused by outdoor measurements or more extreme conditions, such as TTW. Indeed, samples taken in open space show the worst results because they are taken in a completely different scenario. Thus, models struggle to adapt to situations in which multipath components are absent; however, an improvement is achieved in almost all cases. Regarding obstacles, a more marked distinction is noticed. As already evident from the PCA analysis, heavy materials have a significantly different impact on UWB signals compared to wood, plastic, and glass. However, there is almost always an improvement in the raw mean absolute error (MAE). That means that models can learn how to compensate for part of the error independently of the obstacles. A dataset containing sufficient examples for a wide variety of materials can lead to excellent results in many different scenarios.

6.2 Methodology

In this section, I propose a Deep Neural Network (DNN) to solve the range error mitigation problem. Moreover, I present some optimization and quantization

techniques used to increase the computational efficiency of the network. Since UWB anchors are low-power localization devices directly connected to the mobile robot's board, any error compensation technique should be applied locally on the platform to ensure real-time execution with latency compatible with the robot's control frequency. The method should also be as efficient as possible to minimize its impact on the system's overall energy and computational demand. In designing our solution, I primarily optimized the model to reduce memory occupancy and computational effort during inference.

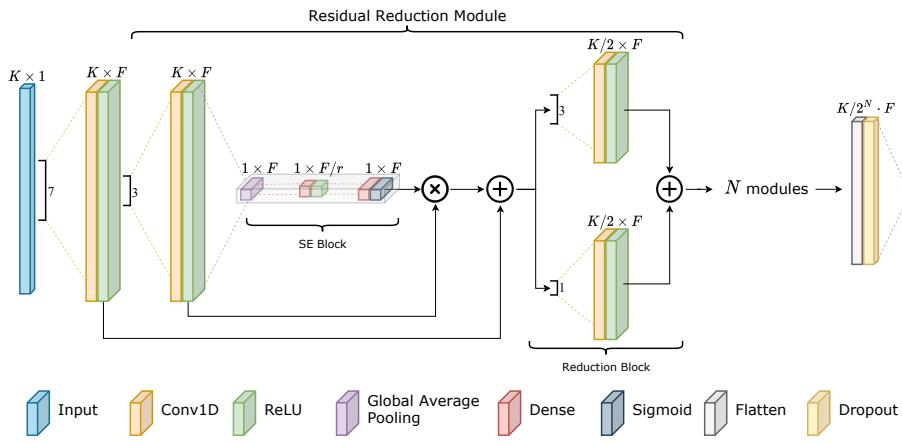


Fig. 6.4 Overview of the REMNet architecture. The model's input is the $K \times 1$ tensor representing the CIR of the measurement. The dimensionality is reduced by N subsequent Residual Reduction Modules (RRM) with a feature attention mechanism. Finally, a fully connected layer composes the high-level extracted features and outputs the range error estimation.

6.2.1 Architecture

I consider the following model for a generic UWB range measurement:

$$\hat{d} = d + \Delta d \quad (6.1)$$

where the actual distance d is intrinsically affected by an error Δd giving the final measurement outcome \hat{d} . The error depends on several factors, among which the most important is the environment and obstacles, resulting in generally worse performance in NLoS conditions.

I formulate the mitigation problem as a regression of the compensation factor Δd , which should be added to the measured range to obtain the actual distance between the two sensors. Therefore, I design a DNN model that predicts an estimate \hat{y} for the true latent error $y = \Delta d$ as a non-linear function of the input CIR vector x . I call the proposed architecture Range Error Mitigation Network (REMNet). I must emphasize that I do not distinguish between LoS and NLoS measurements; instead, I allow the network to learn how to compensate for both conditions autonomously. Therefore, a measurement classification label is not predicted, but the model implicitly performs it during mitigation. Such an approach enables an algorithm that can be applied continuously on board without requiring an additional classification step.

Due to the one-dimensional nature of the data, I select 1D convolutional layers as building blocks of the network. I denote with K the number of temporal samples of the input CIR vector x . I first extract F low-level features with a 1D convolution. The network architecture consists of a stack of N Residual Reduction Modules (RRMs) that learn deep features from high-level features while reducing the temporal dimensionality K . I developed this module by adopting well-known strategies from the deep learning literature, such as residual connections [44], attention mechanisms [45, 46, 47], and sparsely connected graphs [48]. All these methodologies have proven effective in guaranteeing trainable and well-converging networks and are therefore suitable for range error mitigation. The core of the RRM is composed of a residual unit followed by a reduction block:

$$\text{RRM}(x) = \text{Red}(\text{Res}(x)) \quad (6.2)$$

The residual unit has a 1D convolution followed by a Squeeze-and-Excitation (SE) block [46] on the residual branch:

$$\text{Res}(x) = \text{SE}(\text{Conv1D}(x)) + x \quad (6.3)$$

The SE block applies a feature attention mechanism by self-gating each extracted feature with a scaling factor obtained as a non-linear function of itself. Denoting

with \mathbf{x} the $K \times F$ tensor of feature maps extracted by the convolutional layer, I first squeeze it with a global average pooling layer that aggregates the tensor along the temporal dimension, obtaining a single statistic for each feature. The excitation step is then performed with a stack of one bottleneck fully connected (FC) layer that reduces the feature dimension F of a factor r and another FC layer that restores the dimensionality to F with sigmoid activation. This activation outputs F independent scaling factors between 0 and 1 multiplied by the input \mathbf{x} , allowing the network to focus on the most prominent features. Overall, the SE output is computed as:

$$\text{SE}(\mathbf{x}) = \text{FC}_2 \left(\text{FC}_1 \left(\frac{1}{K} \sum_i \mathbf{x}_{ij} \right) \right) \cdot \mathbf{x} \quad (6.4)$$

where

$$\begin{aligned} \text{FC}_1(\mathbf{x}) &= \max(0, \mathbf{x} W_1 + b_1), \quad W_1 \sim (F, \frac{F}{r}), \quad b_1 \sim (\frac{F}{r}) \\ \text{FC}_2(\mathbf{x}) &= \text{sigmoid}(\mathbf{x} W_2 + b_2), \quad W_2 \sim (\frac{F}{r}, F), \quad b_2 \sim (F) \end{aligned} \quad (6.5)$$

The residual unit is followed by a reduction block, which halves the temporal dimension K with two parallel convolutional branches with a stride of 2:

$$\text{Red}(\mathbf{x}) = \text{Conv1D}_1(\mathbf{x}) + \text{Conv1D}_2(\mathbf{x}) \quad (6.6)$$

where both Conv1D₁ and Conv1D₂ have F channels, but different kernel sizes to extract different features.

After N Residual Reduction Modules, I end up with a tensor with shape $(\frac{K}{2^N}, F)$. I flatten it into a single vector and apply a dropout layer to avoid overfitting and help generalization. Finally, an FC layer with linear activation computes an estimate of the compensation value Δd . Except for this final layer and the second FC layer in the SE blocks, I always apply a ReLU non-linearity to all the layers. All the convolutional layers are also zero-padded, so the temporal dimension

is only reduced by the strided convolutions of the reduction block. The overall network architecture overview is presented in Fig. 6.4.

6.2.2 Model Compression

As already mentioned, a UWB range error mitigation technique should respect constraints on memory, power, and latency requirements to be applicable in real-time and onboard. For this reason, I investigate different graph optimization and quantization methods to decrease model size and computational cost. In the literature, several techniques for increasing neural network efficiency have been proposed [49, 50, 51, 52, 42]. In particular, I focus on the following main approaches:

- network pruning and layer fusing, which consist of optimizing the graph by removing low-weight nodes that give almost no contribution to the outcome, and fusing different operations to increase efficiency.
- weight quantization, which reduces the number of bits required to represent each network parameter.
- activation quantization, which reduces the representation dimension of values during the feed-forward pass, thus reducing also the computational demand.
- quantization-aware training, in which the network is trained considering a-priori the effect of quantization, trying to compensate for it.

I produce five versions of REMNet, depending on the adopted techniques. The first is the plain float32 network with no modifications. I apply graph optimization to this first model without quantization to investigate its effect on precision and inference efficiency. The third version is obtained by quantizing the weights to 16 bits, while activations and operations are still represented as 32-bit floating points. The last two models, however, deal with 8-bit full-integer quantization.

This strategy is the most radical in increasing network efficiency by changing the representation of both weights and activations to integers, significantly reducing memory and computational demands due to the high efficiency of integer

computations. However, a significant problem is how to manage the feed-forward pass of the network completely by integer operations. I follow the methodology presented by Jacob et al. [42] in which each weight and activation is quantized with the following scheme:

$$r = S(q - Z) \quad (6.7)$$

where r is the original floating-point value, q the integer quantized value, and S and Z are the quantization parameters, respectively scale and zero point. A fixed-point multiplication approach is employed to address the non-integer scale of S . Thus, all computations are performed with integer-only arithmetic, making inference possible on devices that do not support floating-point operations. I obtain two full-integer models by adopting both post-training quantization and quantization-aware training. This second approach adds fake nodes to the network graph to simulate quantization effects during training. In this way, the gradient descent procedure can consider the integer loss in precision.

6.3 Experiments

In this section, I perform an experimental evaluation of the proposed neural efficient architecture for range error mitigation. Moreover, I test the accuracy and performance of different optimized versions of the network on various heterogeneous devices, collecting data on energy and computational requirements.

6.3.1 Experimental Setting

In the following experiments, I employ the presented dataset, keeping the medium-sized room as the testing set aside. Indeed, instead of stratified sampling of the available data, I decided to perform all tests with indoor instances. That is more similar to an actual infield application and better highlights the generalization capabilities of the proposed methodology. All experiments are performed with 36,023 and 13,210 training and testing data points, respectively, excluding TTW and outdoor measurements. Finally, due to their distinct nature, explicitly

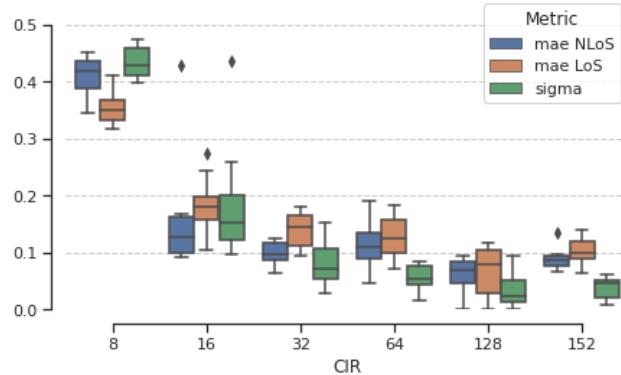


Fig. 6.5 Network performances with different CIR sizes K , starting from the dimension suggested by [37]. Progressively training with fewer input features degrades the network's performance. An input with eight dimensions appears to be the minimum required to obtain an acceptable range error estimation.

labeled LoS samples are used to evaluate the network's ability to recognize this condition and act accordingly.

The final test utilizes the best-developed model for a 3D positioning task to evaluate the impact of range mitigation on localization accuracy. The medium room is chosen as the testing environment, as its samples have not been used to train the networks. Four UWB anchors are placed in the room, and a fixed tag is put in the center. First, the laser tracker precisely measures the nodes' position to provide ground truth, and then the data acquisition begins. Two situations are considered: a fully LoS scenario and a critical NLoS one. Once the samples have been collected, they are prepared for the processing phase, in which range measurements are used to estimate the tag's 3D position, employing a simple Gauss-Newton non-linear optimization algorithm.

All network hyperparameters are obtained with an initial random search analysis followed by a grid search exploration to fine-tune them and find a compromise between accuracy and efficiency. Indeed, working at the architectural level is crucial to satisfying the constraints imposed by the studied application. I use $F = 16$ filters and $N = 3$ reduction modules with $r = 8$. As shown in Fig. 6.4, all 1D convolutional operations have a kernel of size 3, except for the first layer and the second branch of the reduction block. The resulting network has an efficient and highly optimized architecture with 6,151 trainable parameters. Finally, to select the optimal number of input features, as shown in Fig. 6.5, I progressively

reduced the input number of dimensions K while annotating the network metrics. All points are the average result of ten consecutive trials. Experimentation shows that eight dimensions are the minimum number of features required for the network to obtain an acceptable range error estimation. Moreover, I empirically find that an input CIR of 152 elements, as suggested by [37], is redundant and could even slightly reduce the model's performance. On the other hand, fewer dimensions of 128 tend to degrade the network's accuracy almost linearly.

The Adam optimization algorithm [53] is employed for training, with momentum parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. The optimal learning rate, $\eta = 3e - 4$, is experimentally derived using the methodology described in [54]. That is kept constant for 30 epochs with a batch size of 32 and the MAE loss function. I utilize the TensorFlow framework to train the network on a PC equipped with 32 GB of RAM and an NVIDIA GeForce RTX 2080 Super GPU. The overall training process can be completed in under 10 minutes.

Model	MAE [NLoS]	MAE [LoS]	R^2 [NLoS]	R^2 [LoS]	σ [NLoS]
Support Vector Machine (SVM)	0,0766	0,0507	0,4444	0,1256	0,1171
Multilayer Perceptron (MLP)	0,0796	0,0466	0,4194	0,2913	0,1170
CNN-1D [37]	0,0890	0,0523	0,2870	0,1089	0,1285
REMNet float32	0,0687	0,0445	0,5607	0,3483	0,1057
Graph Optimization	0,0687	0,0445	0,5607	0,3483	0,1057
Post-training float16 quantization	0,0688	0,0445	0,5607	0,3484	0,1058
Post-training 8-bit quantization	0,0712	0,0455	0,5361	0,3100	0,1082
Full-integer aware quantization	0,0708	0,0449	0,5404	0,3357	0,1079

Table 6.1 Proposed architecture performances after graph optimization and different levels of weight quantization. Initial values of MAE for NLoS and LoS signals are 0.1242 m and 0.0594 m, respectively. It is possible to notice that the different transformations have a minimal impact on the network's range error estimation capability. Moreover, three baseline approaches are tested and compared with the efficient REMNet model and its optimization versions.

6.3.2 Results

The medium room data samples have a starting MAE of 0.1242 m and a standard deviation of $\sigma = 0.1642$ m. On the other hand, the baseline MAE of explicitly labeled LoS samples is 0.0594 m.

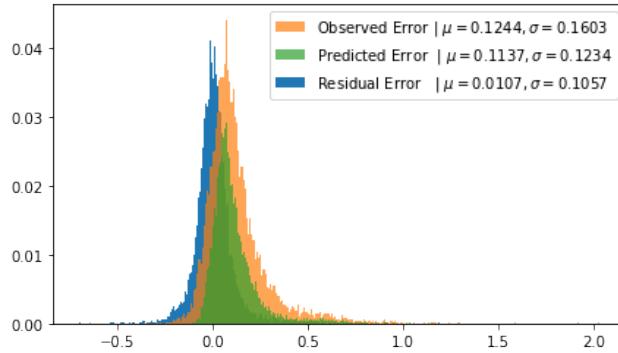


Fig. 6.6 Normalized histograms with 300 bins each. It is possible to notice how the residual range error distribution (blue histogram) is almost Gaussian. That significantly enhances the optimality and simplicity of the subsequent iterative localization algorithm [55].

Fig. 6.6 shows the results obtained by the reference architecture trained with the setting illustrated in Section 6.3.1. It is possible to notice how the network can almost completely compensate for the offset of the original range error and reduce the standard deviation by 34.1%. Moreover, as summarized in Table 6.1, the network can easily detect LoS input signals and apply a small correction factor that considers the multipath effect. The residual error indicates a percentage improvement of 25.1%. On the other hand, MAE for NLoS signals improves by 44.7%, reducing the error to 0.0697 m, which is close to the actual precision of DWM1000 boards [56].

In the upper part of Table 6.1, three simple models (SVM, MLP, and CNN-1D) are included as a reference. For support vector machine (SVM) and MLP, I adopt the six hand-crafted features described in [19, 57]. I use radial basis function as the kernel for our SVM and a 3-layer architecture with 64 hidden neurons for the MLP. Instead, for [37], I feed the network with 152 bins and set the hyperparameters suggested in the article. It is noticeable how REMNet performs better than other literature methodologies, even with a highly efficient architecture. For completeness, Table 6.2 presents the results obtained with cross-validation on the three rooms. REMNet achieves comparable range error mitigation in the three different configurations.

Metric	SR	MR	BR
Training Samples	31,632	36,023	30,811
Test Samples [NLoS]	17,601	13,210	18,422
Test Samples [LoS]	4,691	4,691	4,691
σ_{obs} [NLoS]	0,1508	0,1603	0,1851
σ_{res} [NLoS]	0,1131	0,1057	0,1204
μ_{obs} [NLoS]	0,0881	0,1244	0,1057
μ_{res} [NLoS]	0,0058	0,0153	0,0171
MAE [NLoS]	0,0638	0,0687	0,0702
MAE [LoS]	0,0462	0,0445	0,044
R^2 [NLoS]	0,5793	0,5607	0,5
R^2 [LoS]	0,4005	0,3483	0,444

Table 6.2 REMNet cross-validation results with the three different room sizes, small room (SR), medium room(MR), and big room(BR). Each column presents metrics for NLoS and LoS signals for the room excluded by the training procedure.

Power and Latency

Range error mitigation should be performed directly on the platform where the UWB tag is attached. So, energy consumption and computational power play a decisive role in the applicability of the proposed methodology. However, real-time range mitigation with the whole CIR could be very computationally intensive [58]. Consequently, to comply with cost, energy, size, and computational constraints, I investigate the effects of optimization, detailed in Section 6.2, on the network's accuracy instead of manually extracting a reduced number of features from the CIR.

Graph optimization techniques and different weight quantization levels are examined, starting from the pre-trained reference network. In Table 6.1, the performances of the model after different optimization processes are summarized. Even if the overall metrics are degraded, these changes are mostly negligible. Moreover, it is possible to notice that full-integer quantization, generally producing a size reduction and speed-up of 75%, decreases the NLoS MAE by only 3% if carried out with awareness training. That opens the possibility of achieving effective range mitigation with an almost negligible impact on the overall application. Indeed, extreme weight quantization implies a smaller model size with reduced memory usage, significant latency reduction, and the potential for utilizing highly efficient neural accelerators.

Inference

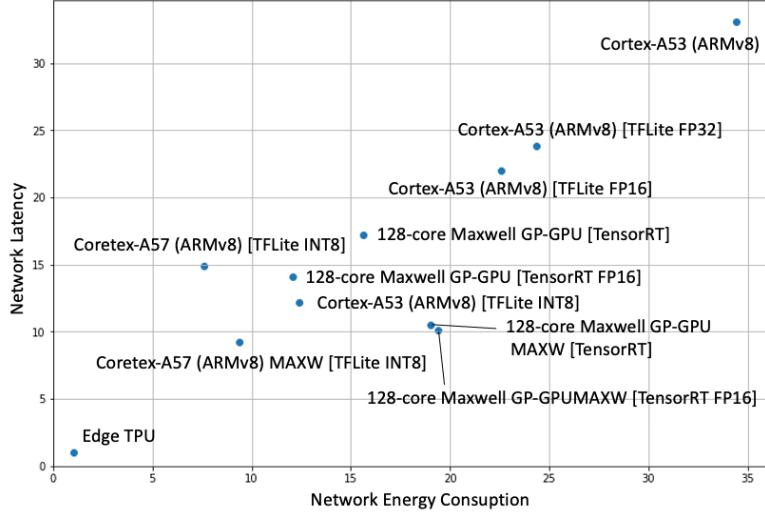


Fig. 6.7 Energy and latency are two important constraints for an effective range error mitigation. Indeed, error correction is performed progressively over all received anchor signals on board the platform connected with the tag. Range error mitigation would not be applicable without an optimized and efficient correction model.

In this section, I test different optimized networks on several devices and hardware accelerators, annotating power and computational requests. The selected microchips are chosen by performing a market evaluation at the time of writing, considering standard computational boards for indoor navigation. Indeed, robotic platforms are typically equipped with Linux-capable Arm Cortex-A CPUs or powerful co-processors and accelerators, such as Nvidia GPUs, Visual Processing Units (VPUs), or Tensor Processing Units (TPUs). I adopt two standard libraries for network deployment, TensorFlow-Lite² and TensorRT³ to produce the optimized models. Both are directly integrated into the TensorFlow framework and are specifically conceived to target different hardware platforms. In particular, I target Cortex-A53, A57 processors, and Edge TPU with TF-Lite, as well as the Nvidia RTX 2080 and 128-core Maxwell GP-GPUs with TensorRT.

Experimentation results are summarized in Table 6.3. It is possible to notice that, due to the high efficiency of the proposed architecture, all configurations satisfy a sufficient inference speed and are compliant with a practical range error

²tensorflow.org/lite

³developer.nvidia.com/tensorrt

Device	G.O.	W.P.	Latency [ms]	Latency _{4 batch} [ms]	V _{al} [V]	I _{idl} [A]	P _{run} [W]	E _{net} [mJ]	Size [KB]
RTX 2080	✓	FP32	19.7 ± 0.23	19.3 ± 0.24	N.A.	N.A.	32	617.6	250.0
	✓	FP32	0.69 ± 0.13	0.69 ± 0.16	N.A.	N.A.	20	138.0	613.0
	✓	FP16	0.54 ± 0.09	0.51 ± 0.02	N.A.	N.A.	18	97.2	615.0
Cortex-A53	✗	FP32	16.9 ± 0.03	17.2 ± 0.05	5.0	0.4	1.0	17.2	250.0
	✓	FP32	12.2 ± 0.03	N.A.	5.0	0.4	1.0	12.2	40.7
	✓	FP16	11.2 ± 0.03	N.A.	5.0	0.4	1.0	11.2	33.9
	✓	INT8	6.23 ± 0.02	N.A.	5.0	0.4	1.0	6.2	32.7
Cortex-A57	✓	INT8	7.63	N.A.	5.0	0.5	0.8	3.81	32.7
	✓	INT8	4.71	N.A.	5.0	0.5	1.0	4.7	32.7
128-core Maxwell	✓	FP32	8.78 ± 0.09	9.03 ± 0.1	5.0	0.67	0.9	7.8	615.0
	✓	FP16	7.22 ± 0.08	7.43 ± 0.05	5.0	0.67	0.9	6.04	613.0
	✓	FP32	5.36 ± 0.05	5.29 ± 0.05	5.0	0.5	1.8	9.7	615.0
	✓	FP16	5.18 ± 0.04	5.39 ± 0.05	5.0	0.5	1.0	4.7	613.0
Edge TPU	✓	INT8	0.51 ± 0.1	N.A.	5.0	0.59	0.7	0.5	70.54

Table 6.3 Comparison between different devices' energy consumption and inference performances. Graph optimization (G.O.) and weight precision (W.P.) reduction further increase the capability of our already efficient neural network design, helping to deal with energy, speed, size, and cost constraints.

mitigation solution. Nevertheless, the various optimization techniques have a significant impact on the energy consumed by the network. Indeed, considering experiments performed with the Cortex-A53, optimization can reduce energy consumption by nearly a factor of three, starting with an initial value of 17.2 mJ and decreasing to barely 6.2 mJ. Moreover, the model size is significantly reduced from 250 KB to 32.7 KB. That implies a smaller storage size and less RAM at runtime, freeing up memory for the main application where UWB localization is needed. Finally, as further highlighted by Fig. 6.7 and the results of the previous subsection, the Edge TPU neural accelerator with a full-integer quantization-aware model is the preferable solution for deployment. With only 0.51 ms of latency and 0.5 mJ of energy consumption, it has a minimal impact on the overall application's performance, allowing for the exploitation of duty cycling and energy-saving techniques. Indeed, as stated in our proposed methodology section, the already efficient design of our architecture, in conjunction with 8-bit weight precision and graph optimization techniques, makes deep learning a feasible solution for an effective range error mitigation for UWB at the edge.

Trilateration

As described in Section 6.3.1, the effect of the proposed method is lastly verified by using the full-integer quantization-aware model for 3D positioning, in which

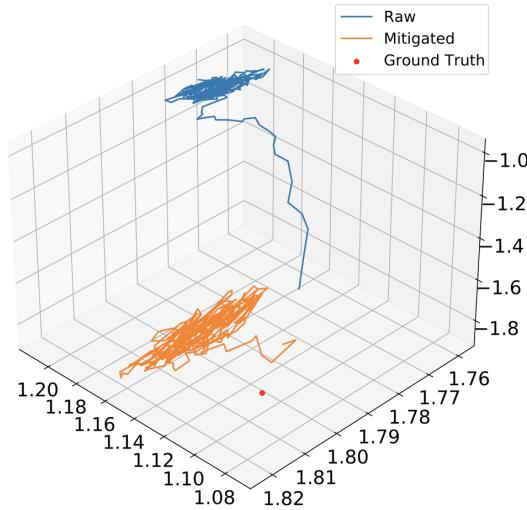


Fig. 6.8 Comparison between position estimations of a fixed tag in NLoS conditions. In light blue, the results obtained from raw range measurements, in orange, the ones achieved with our quantization-aware mitigation model.

the results obtained from raw range estimates are compared to the ones achieved with our mitigation model in the loop. The results are summarized in Table 6.4, while Fig. 6.8 gives a graphical representation of the NLoS results. Regarding the LoS case, the positioning system already achieves good precision on its own, with a very low range MAE and, consequently, a low position MAE. In this case, the mitigation effect is irrelevant, resulting in a slight increase in ranging error but a slight decrease in positioning error. As expected, the model learns to apply subtle corrections to LoS samples, thereby avoiding worsening already good measurements. Instead, the NLoS scenario shows a significant improvement, as the range MAE is more than halved, reaching a value comparable to the LoS case and confirming the results shown in Section 6.3.2. Consequently, the error in position estimation is significantly reduced, decreasing from 57.7 cm to 18.2 cm. Although the final accuracy is still considerably higher than the one found in the LoS case, a reduction of 68% is considered a significant result. Indeed, our approach allows achieving a suitable precision for many indoor robotic applications, showing good generalization to unknown environments.

Results	LOS		NLOS	
	Range MAE [m]	Position MAE [m]	Range MAE [m]	Position MAE [m]
Raw UWB Measurements	0.0388	0.0703	0.1129	0.5772
Full-integer Aware Quantization	0.0465	0.0679	0.0571	0.1817

Table 6.4 Results obtained from the positioning test in the medium room, which is not used for the model's training. For each test, the mean absolute error is reported for both the range estimates and the final position result, highlighting the effect of the former on the latter.

6.4 Conclusion

I introduced REMNet, a novel representation learning model specifically designed to provide an effective solution for range error mitigation. Furthermore, I proposed a set of optimization techniques to further enhance its efficiency and computational results, enabling range error mitigation on ultra-low-power microcontrollers. Extensive experimentation has proven the effectiveness of our methodology and its generality across disparate scenarios, running at a high frequency on microcontrollers and providing enhanced localization in indoor environments. Further work will aim to integrate the deep learning architecture directly on an ultra-low-power microcontroller, which is placed on the UWB device.

The development of REMNet concludes the part of this thesis dedicated to designing efficient deep learning models tailored for perception and localization in resource-constrained robotic systems. From action recognition to image enhancement and depth correction, each contribution demonstrates how tailored architectural choices and optimization techniques can enable real-time performance on edge devices without compromising accuracy.

The following part shifts focus toward a complementary but equally critical dimension of deep learning for robotics: robustness. In real-world environments, autonomous systems must operate reliably in the presence of domain shifts, noisy inputs, and distributional uncertainty. The following chapters address these challenges by exploring methodologies that enhance the generalization and resilience of AI models deployed in the wild.

Bibliography

- [1] S. Angarano, V. Mazzia, F. Salvetti, G. Fantin, and M. Chiaberge, “Robust ultra-wideband range error mitigation with deep learning at the edge,” *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104278, 2021.
- [2] F. Zafari, A. Gkelias, and K. K. Leung, “A survey of indoor localization systems and technologies,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019.
- [3] Q. D. Vo and P. De, “A survey of fingerprint-based outdoor localization,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 491–506, 2015.
- [4] E. Y. Menta, N. Malm, R. Jäntti, K. Ruttik, M. Costa, and K. Leppänen, “On the performance of aoa-based localization in 5g ultra-dense networks,” *IEEE Access*, vol. 7, pp. 33 870–33 880, 2019.
- [5] N. Karlsson, M. E. Munich, L. Goncalves, J. Ostrowski, E. Di Bernardo, and P. Pirjanian, “Core technologies for service robotics,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No. 04CH37566)*, IEEE, vol. 3, 2004, pp. 2979–2984.
- [6] B. Cheng, L. Cui, W. Jia, W. Zhao, and P. H. Gerhard, “Multiple region of interest coverage in camera sensor networks for tele-intensive care units,” *IEEE Transactions on Industrial Informatics*, vol. 12, no. 6, pp. 2331–2341, 2016.
- [7] S. Zorn, R. Rose, A. Goetz, and R. Weigel, “A novel technique for mobile phone localization for search and rescue applications,” in *2010 International Conference on Indoor Positioning and Indoor Navigation*, IEEE, 2010, pp. 1–4.
- [8] K. Jo, K. Chu, and M. Sunwoo, “Gps-bias correction for precise localization of autonomous vehicles,” in *2013 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2013, pp. 636–641.
- [9] M. Stahlke, S. Kram, C. Mutschler, and T. Mahr, “Nlos detection using uwb channel impulse responses and convolutional neural networks,” in *2020 International Conference on Localization and GNSS (ICL-GNSS)*, IEEE, 2020, pp. 1–6.
- [10] W. Wen, G. Zhang, and L.-T. Hsu, “Gnss nlos exclusion based on dynamic object detection using lidar point cloud,” *IEEE Transactions on Intelligent Transportation Systems*, 2019.

- [11] J. Ray, J. Griffiths, X. Collilieux, and P. Rebischung, "Subseasonal gnss positioning errors," *Geophysical Research Letters*, vol. 40, no. 22, pp. 5854–5860, 2013.
- [12] P. Tiwari and P. K. Malik, "Design of uwb antenna for the 5g mobile communication applications: A review," in *2020 International Conference on Computation, Automation and Knowledge Management (ICCAKM)*, IEEE, 2020, pp. 24–30.
- [13] V. Magnago, P. Corbalán, G. P. Picco, L. Palopoli, and D. Fontanelli, "Robot localization via odometry-assisted ultra-wideband ranging with stochastic guarantees.," in *IROS*, 2019, pp. 1607–1613.
- [14] L. Schmid, D. Salido-Monzú, and A. Wieser, "Accuracy assessment and learned error mitigation of uwb tof ranging," in *2019 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, IEEE, 2019, pp. 1–8.
- [15] V. Savic, E. G. Larsson, J. Ferrer-Coll, and P. Stenumgaard, "Kernel methods for accurate uwb-based ranging with reduced complexity," *IEEE Transactions on Wireless Communications*, vol. 15, no. 3, pp. 1783–1793, 2015.
- [16] T. Otim, L. E. Díez, A. Bahillo, P. Lopez-Iturri, and F. Falcone, "Effects of the body wearable sensor position on the uwb localization accuracy," *Electronics*, vol. 8, no. 11, p. 1351, 2019.
- [17] Y.-Y. Chen, S.-P. Huang, T.-W. Wu, W.-T. Tsai, C.-Y. Liou, and S.-G. Mao, "Uwb system for indoor positioning and tracking with arbitrary target orientation, optimal anchor location, and adaptive nlos mitigation," *IEEE Transactions on Vehicular Technology*, 2020.
- [18] B. Silva and G. P. Hancke, "Ir-uwb-based non-line-of-sight identification in harsh environments: Principles and challenges," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 3, pp. 1188–1195, 2016.
- [19] S. Marano, W. M. Gifford, H. Wymeersch, and M. Z. Win, "Nlos identification and mitigation for localization based on uwb experimental data," *IEEE Journal on selected areas in communications*, vol. 28, no. 7, pp. 1026–1035, 2010.
- [20] J. Schroeder, S. Galler, K. Kyamakya, and K. Jobmann, "Nlos detection algorithms for ultra-wideband localization," in *2007 4th Workshop on Positioning, Navigation and Communication*, IEEE, 2007, pp. 159–166.
- [21] V. Barral, C. J. Escudero, and J. A. García-Naya, "Nlos classification based on rss and ranging statistics obtained from low-cost uwb devices," in *2019 27th European Signal Processing Conference (EUSIPCO)*, IEEE, 2019, pp. 1–5.
- [22] A. H. Muqaibel, M. A. Landolsi, and M. N. Mahmood, "Practical evaluation of nlos/los parametric classification in uwb channels," in *2013 1st International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*, IEEE, 2013, pp. 1–6.
- [23] R. Ying, T. Jiang, and Z. Xing, "Classification of transmission environment in uwb communication using a support vector machine," in *2012 IEEE Globecom Workshops*, IEEE, 2012, pp. 1389–1393.

- [24] Z. Xiao, H. Wen, A. Markham, N. Trigoni, P. Blunsom, and J. Frolik, “Non-line-of-sight identification and mitigation using received signal strength,” *IEEE Transactions on Wireless Communications*, vol. 14, no. 3, pp. 1689–1702, 2014.
- [25] C. Jiang, J. Shen, S. Chen, Y. Chen, D. Liu, and Y. Bo, “Uwb nlos/los classification using deep learning method,” *IEEE Communications Letters*, vol. 24, no. 10, pp. 2226–2230, 2020.
- [26] K. Gururaj, A. K. Rajendra, Y. Song, C. L. Law, and G. Cai, “Real-time identification of nlos range measurements for enhanced uwb localization,” in *2017 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, IEEE, 2017, pp. 1–7.
- [27] A. Niitsoo, T. Edelhäußer, and C. Mutschler, “Convolutional neural networks for position estimation in tdoa-based locating systems,” in *2018 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, IEEE, 2018, pp. 1–8.
- [28] C.-H. Hsieh, J.-Y. Chen, and B.-H. Nien, “Deep learning-based indoor localization using received signal strength and channel state information,” *IEEE access*, vol. 7, pp. 33 256–33 267, 2019.
- [29] A. Poulose and D. S. Han, “Uwb indoor localization using deep learning lstm networks,” *Applied Sciences*, vol. 10, no. 18, p. 6290, 2020.
- [30] Z. Zeng, R. Bai, L. Wang, and S. Liu, “Nlos identification and mitigation based on cir with particle filter,” in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, IEEE, 2019, pp. 1–6.
- [31] C. Mao, K. Lin, T. Yu, and Y. Shen, “A probabilistic learning approach to uwb ranging error mitigation,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2018, pp. 1–6.
- [32] H. Wymeersch, S. Maranò, W. M. Gifford, and M. Z. Win, “A machine learning approach to ranging error mitigation for uwb localization,” *IEEE transactions on communications*, vol. 60, no. 6, pp. 1719–1728, 2012.
- [33] B. J. Silva and G. P. Hancke, “Ranging error mitigation for through-the-wall non-line-of-sight conditions,” *IEEE Transactions on Industrial Informatics*, 2020.
- [34] A. Khaliq, V. Mazzia, and M. 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.
- [35] D. Aghi, V. Mazzia, and M. Chiaberge, “Local motion planner for autonomous navigation in vineyards with a rgb-d camera-based algorithm and deep learning synergy,” *Machines*, vol. 8, no. 2, p. 27, 2020.
- [36] F. Salvetti, V. Mazzia, A. Khaliq, and M. Chiaberge, “Multi-image super resolution of remotely sensed images using residual attention deep neural networks,” *Remote Sensing*, vol. 12, no. 14, p. 2207, 2020.

- [37] K. Bregar and M. Mohorčič, “Improving indoor localization using convolutional neural networks on computationally restricted devices,” *IEEE Access*, vol. 6, pp. 17 429–17 441, 2018.
- [38] S. Angarano, F. Salvetti, V. Mazzia, G. Fantin, D. Gandini, and M. Chiaberge, “Ultra-low-power range error mitigation for ultra-wideband precise localization,” in *Science and Information Conference*, Springer, 2022, pp. 814–824.
- [39] S. Angarano, “Deep learning methodologies for uwb ranging error compensation,” Ph.D. dissertation, Politecnico di Torino, 2020.
- [40] W. Wang and J. Shen, “Deep visual attention prediction,” *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2368–2378, 2017.
- [41] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, inception-resnet and the impact of residual connections on learning,” *arxiv: 1602.07261*, 2016.
- [42] B. 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.
- [43] S. Angarano, F. Salvetti, V. Mazzia, G. Fantin, and M. Chiaberge, *Deep UWB: A dataset for UWB ranging error mitigation in indoor environments*.
- [44] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [45] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [46] J. Hu, L. Shen, and G. Sun, “Squeeze-and-excitation networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.
- [47] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “Cbam: Convolutional block attention module,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.
- [48] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [49] Y. Gong, L. Liu, M. Yang, and L. Bourdev, “Compressing deep convolutional networks using vector quantization,” *arxiv: 1412.6115*, 2014.
- [50] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *International Conference on Machine Learning*, 2015, pp. 1737–1746.
- [51] S. Han *et al.*, “Eie: Efficient inference engine on compressed deep neural network,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.
- [52] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arxiv: 1510.00149*, 2015.

- [53] D. P. Kingma, “Adam: A method for stochastic optimization,” *arxiv: 1412.6980*, 2014.
- [54] L. 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.
- [55] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, vol. 3.
- [56] A. R. Jiménez and F. Seco, “Comparing decawave and bespoon uwb location systems: Indoor/outdoor performance analysis,” in *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, IEEE, 2016, pp. 1–8.
- [57] S. Mazuelas, A. Conti, J. C. Allen, and M. Z. Win, “Soft range information for network localization,” *IEEE Transactions on Signal Processing*, vol. 66, no. 12, pp. 3155–3168, 2018.
- [58] Z. Zeng, S. Liu, and L. Wang, “Nlos identification for uwb based on channel impulse response,” in *2018 12th International Conference on Signal Processing and Communication Systems (ICSPCS)*, IEEE, 2018, pp. 1–6.

Part III

Making AI Robust

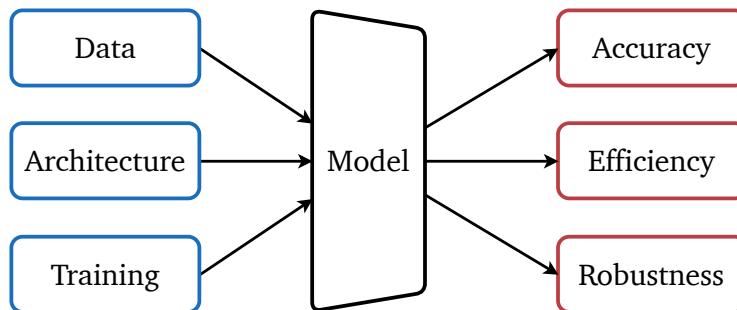


Fig. 6.9 The elements constituting a machine learning model (left) and its properties (right).

In the previous part, I demonstrated that improving inference efficiency is possible without losing accuracy. Returning to the scheme in Fig. 6.9, there is still a property to be discussed: robustness. Whenever an ML model is deployed in a real-world application scenario, multiple factors create risks for its correct functioning. A considerable part of these arises from input data, which often has a different statistical distribution from training and validation sets. This phenomenon poses a problem of generalization. Other issues may be caused by hardware or software faults, which alter the system's functioning and lead to prediction errors.

This part analyzes three principal works that evaluate or improve model generalization and **robustness** to data distribution shifts and hardware faults. Chapter 7 investigates the domain generalization ability of recent convolutional and transformer-based architectures and the impact of several training methods. Chapter 8 proposes a novel domain generalization method based on the knowledge distilled from an ensemble of experts. Finally, Chapter 9 presents a multi-head model for onboard satellite image processing trained with self-supervised learning and optimized for fault tolerance.

Chapter 7

Rediscovering The Role of Backbones in Domain Generalization

Original Paper: *S. Angarano, M. Martini, F. Salvetti, V. Mazzia, and M. Chiaberge, “Back-to-bones: Rediscovering the Role of Backbones in Domain Generalization”, Pattern Recognition, 2024.* [1]

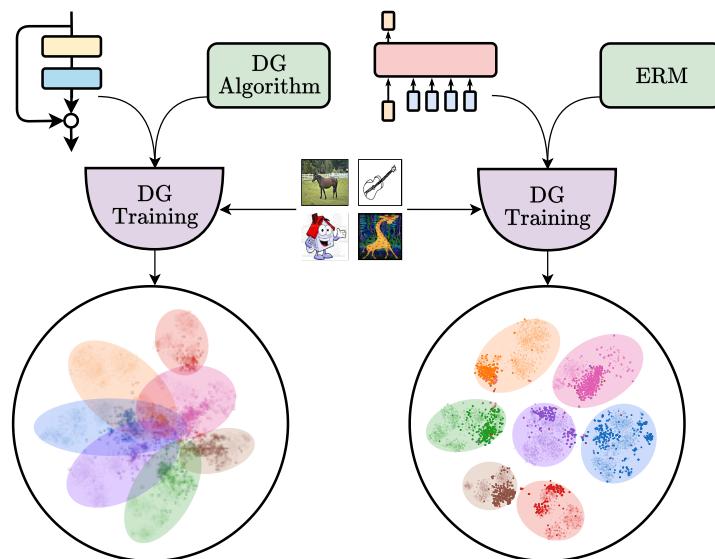


Fig. 7.1 My experimentation proves the importance of backbones in Domain Generalization. I find that novel architectures, such as transformer-based models, lead to a better representation of data, outperforming outdated backbones, such as ResNets, and leaving marginal room for feature mapping improvement through DG algorithms.

The problem of induction has a central role in the learning process. Without generalization, machine learning algorithms would be able to exhibit useful behaviors only in situations identical to those they experienced during training [2]. Deep neural networks are powerful models capable of extracting subtle regularities from training data. Nevertheless, they often fail to generalize to out-of-training scenarios. Even if supervised training methodologies have proved to produce neural networks with remarkable performances, their results are valid only in well-defined settings and do not generalize across tasks, domains, and categories [3].

For object recognition, several literature works have shown that, unlike humans, training frameworks commonly produce networks that are more prone to be biased towards textures and global image statistics in making decisions [4, 5], prioritizing easier-to-fit spurious correlations in favor of invariant shape cues [6]. That prevents scaling on all samples, showing a distribution shift and posing a concrete barrier to deploying models in all critical applications that require true generalization power. For instance, autonomous driving may encounter environments and circumstances not encountered during the training phase, which can be caused by variations in lighting, weather, background, and dynamics of nearby objects. Indeed, disparate independent studies report that neural networks can easily fail without effective generalization capabilities, negatively affecting the overall system's behavior [7, 8]. Similarly, another realistic example of a domain gap is training neural networks in simulation, which has become a standard procedure in the robotics research community. Recently, researchers have faced the Simulation-to-Reality (Sim2Real) gap problem, trying to effectively transfer Deep Neural Networks from virtual scenarios to the real world [9, 10].

Domain Generalization (DG) aims at training models that generalize to out-of-distribution (OoD) data. 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 [11, 12]. As an extension of supervised learning, this approach aims to minimize empirical risk at training time to extrapolate an overall probability distribution from source datasets, enabling accurate classification of OoD data. Over the last decade, aware of the tremendous impact of generalization on computer vision applications, the DG research community has addressed the problem with algorithms that aim to identify invariant features that hold across novel domains. Among

the constellation of proposed approaches, I identify the principal broad strategies adopted for domain generalization in augmenting the source domain [13, 14], aligning domain distributions [15, 16, 17, 18, 19], meta-learning [20, 21, 22], self-supervised learning [23, 24, 25], and regularization strategies [26, 27, 28, 29, 30].

Although methodologies have provided meaningful insights into the nature of DG over the years, recent research contributions have proposed a rigorous testing benchmark to evaluate and compare the advantages supplied by DG algorithms fairly. With DOMAINBED [31], 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 compared to naive empirical risk minimization (ERM). Nevertheless, the study has been carried out uniquely with ResNet50 [32] as a feature extractor. Thus, new DG algorithms are still proposed, overlooking a fundamental aspect of practical deep learning applications: the importance of the backbone.

In past years, several competitive deep learning architectures, characterized by different types of feature extractors, have been proposed to solve classification tasks [33] on popular datasets such as ImageNet [34]. Classical backbones are based on convolutional layers. For instance, AlexNet [35] is a network composed of a small set of convolutions and max-pooling layers, combined with ReLU activation. The VGG architecture [36], in its variations VGG-16 and VGG-19, further explores the convolution-pooling structure by stacking more layers and reaching a deeper design. ResNet [32] first adopted a residual approach to help gradient flow with skip-connections, and it is still a widely adopted backbone for various computer vision tasks. Similarly to VGG, ResNet has been proposed in various forms, with different depths, such as ResNet18, ResNet34, and ResNet50. Other architectures, such as DenseNet [37] or InceptionNet [38], focus on different mechanisms, including dense connections and parallelization of convolutional layers with varying kernel sizes. MobileNet [39] and EfficientNet [40] have been proposed to increase model efficiency, reaching competitive classification results with lightweight architectures and fewer parameters. More recently, self-attention-based models have reached state-of-the-art image classification performance, inspired by the Transformer [41] architecture first proposed for language modeling. In particular, the Vision Transformer (ViT) [42] first adopted a Transformer encoder for vision tasks, while its training methodology has been

refined by the Data Efficient ViT (DeiT) [43]; ConViT [44] combines convolutions with self-attention, and LeViT [45] focuses on a pyramidal architecture of self-attention layers that progressively shrinks spatial dimensions.

This rich literature landscape offers a wide range of choices for researchers when selecting feature extractors for visual applications. However, among the various computer vision tasks, the DG community has substantially neglected the generalization capabilities of existing backbones, instead promoting sophisticated algorithms combined with outdated feature extractors, such as ResNet18 or even AlexNet. Only very few attempts have been made in this direction: Sultana *et al.* [46] proposed the first DG algorithm specifically for Transformer-based models; Guo *et al.* [47] studied how MLP-like models generalize better than CNN by incorporating more global-structure information and proposed a new Mixture-of-Experts architecture; a concurrent work by Li *et al.* [48] has brought valuable insights on the intuition that multi-head attention is a low-pass filter with a shape bias, while convolution is a high-pass filter with a texture bias.

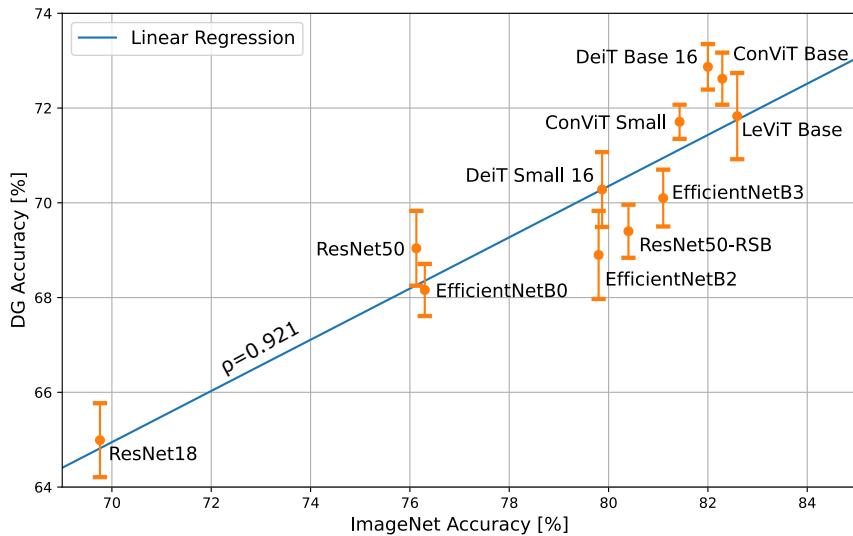


Fig. 7.2 DG accuracy achieved by tested backbones compared with their performance on ImageNet, with error bars. Regardless of the different architectures and priors, I find a strong linear correlation between the two metrics ($\rho = 0.921$). In Section 7.1.1, I also compare DG accuracy with the number of parameters, finding a much weaker correlation.

In this paper [1], I argue that domain gaps in realistic scenarios should be addressed by accurately selecting the model architecture, which is undeniably central in most deep learning applications (Fig. 7.1). I come to similar experimental

conclusions to the concurrent work of [48] on the generalization of transformers and the weaker effect of DG methods. However, I push it further by evaluating multiple backbones with different priors and several DG methodologies and find a strong correlation between ImageNet accuracy and generalization.

In particular, I conduct extensive experimentation on the principal DG datasets and assess a wide variety of backbone architectures, ranging from novel vision transformers to standard convolutional models. My results demonstrate an evident linear correlation between large-scale single-domain classification accuracy and domain generalization performance (Fig. 7.2). Moreover, I achieve state-of-the-art results in DG with naive ERM and simple data augmentation, remarking that, under fair testing conditions, the most promising algorithms presented so far give no substantial advantage.

I reinforce the experimentation with a visual analysis of the feature extractors. Using the t-SNE manifold learning technique [49] on extracted features, I show that novel backbones map same-class samples closer in the feature space and outperform older architectures when trained in a DG framework. I propose a quantitative evaluation of this difference by fitting a k-NN classifier on the extracted features.

This study aims to promote a complete and meaningful approach to the domain generalization problem, avoiding isolated research efforts on DG algorithms and encouraging contributions that target the overall maximization of model generalization. Evidence in the literature suggests that researchers from diverse application fields could significantly benefit from a shift in the DG paradigm towards more realistic circumstances. For instance, data augmentation can automatically be exploited to generate a vast collection of artificial source domains. Domain randomization fully exploits this principle [9], demonstrating its effectiveness in training agents in simulation for controlling manipulators accomplishing visual tasks [50] and autonomous racing drones [51]. That provides further concrete evidence that the success of DG in real-world applications relies on simple ERM techniques, which offer easy implementation and a robust generalization boost.

As an outcome of this work, I release BACK-TO-BONES¹, a testbed to encourage the deep learning community to evaluate and compare the domain generalization performance of newly proposed backbones.

7.1 The Back-to-Bones Benchmark

I set up my experimental benchmark to run a detailed analysis of the role of feature extractors 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, I provide details on my experimental setup.

Backbone	PACS	VLCS	Office-Home	Terra Incognita	Average	ImageNet	Parameters
ResNet18	80.51 ± 0.29	74.64 ± 0.61	63.87 ± 0.36	40.93 ± 1.85	64.99 ± 0.78	69.76	11.69M
ResNet50 [31]	85.50 ± 0.20	77.50 ± 0.40	66.50 ± 0.30	46.10 ± 1.80	68.90 ± 0.68	76.13	25.56M
ResNet50	83.85 ± 0.77	76.21 ± 1.20	68.79 ± 0.21	47.32 ± 0.97	69.04 ± 0.79	76.13	25.56M
ResNet50 A1	84.52 ± 0.68	78.37 ± 0.56	72.47 ± 0.13	42.23 ± 0.87	69.40 ± 0.56	80.40	25.56M
EfficientNetB0	85.46 ± 0.65	75.16 ± 0.34	67.27 ± 0.27	44.76 ± 0.94	68.16 ± 0.55	76.30	5.29M
EfficientNetB2	87.02 ± 1.37	75.44 ± 0.20	69.35 ± 0.24	43.80 ± 1.90	68.90 ± 0.93	79.80	9.11M
EfficientNetB3	86.71 ± 0.30	78.14 ± 0.18	69.84 ± 0.08	45.70 ± 1.84	70.10 ± 0.60	81.10	12.23M
DeiT Small 16	86.22 ± 1.33	79.47 ± 0.41	72.03 ± 0.33	43.40 ± 1.08	70.28 ± 0.79	79.87	22.05M
DeiT Base 16	88.10 ± 0.48	79.80 ± 0.32	76.35 ± 0.36	47.22 ± 0.75	72.87 ± 0.48	82.00	86.57M
ConViT Small	87.10 ± 0.33	80.00 ± 0.34	73.90 ± 0.17	45.83 ± 0.61	71.71 ± 0.36	81.43	27.78M
ConViT Base	87.27 ± 0.40	80.31 ± 0.67	76.51 ± 0.25	46.37 ± 0.89	72.62 ± 0.55	82.29	86.54M
LeViT Base	87.55 ± 1.50	78.91 ± 0.50	75.16 ± 0.13	45.68 ± 1.50	71.83 ± 0.91	82.59	39.13M
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	22.05M
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	88.22M
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	86.57M

Table 7.1 Baselines comparison of different backbones for DG. I report the average accuracy over three runs, along with the associated standard deviation, for each model. I 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.

Backbones To be consistent with previous works, I include ResNet18 and ResNet50 [32] in the benchmark and compare them with some of the most successful architectures proposed in recent image classification research. I also consider the latest ResNet50 A1 [52], trained using the most recent practices in optimization

¹github.com/PIC4SeR/Back-to-Bones

and data augmentation and reaching a remarkable 80.4% top-1 accuracy on Imagenet1K. I include different sizes for each network to glimpse the effects of model dimension on DG accuracy. EfficientNet [40] demonstrated that systematic model scaling and dimension balancing yield remarkable results with fewer parameters. For this reason, I select three network versions, namely B0, B2, and B3. Finally, transformers [41] 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, I choose DeiT (Small and Base) [43], ConViT [44] (both in its Small and Base configurations), and LeViT Base [45]. To provide further insights into the effect of additional pretraining data beyond the standard ImageNet [34], I also include Vision Transformer (ViT) [42] 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. Further information on architectural details can be found in the cited papers. I report the number of parameters for each model in the last column of Table 7.1.

Datasets Among the various datasets created explicitly for DG in recent years, I use four of the most widely adopted ones for my primary experimentation. VLCS [53] considers four previous classification datasets as domains, while PACS [54] and Office-Home [55] focus more on style shifts (e.g., from photos to cartoons, sketches, and paintings). Terra Incognita [56] comprehends several animal photos taken with camera traps placed in different locations by day and night. To this, I add DomainNet [57], a larger and more recent dataset comprising six domains, each divided by style, and 345 classes. I 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. I omit Rotated MNIST [58] and Colored MNIST [6] since I consider them too distant from any practical application. Moreover, from my perspective, simple rotation and colorization do not constitute actual domain shifts.

DG Algorithms I choose some of the most promising DG algorithms in recent research, particularly considering their performance on DOMAINBED [31]. Moreover, I select them to explore different approaches to the DG problem. CORAL [16] and MMD [18], indeed, focus on aligning the extracted features through second-order statistics (covariance). On the other hand, Mixup [59] works directly on input

images, interpolating samples from different domains and considering the loss coming from both precursors. RSC [27], instead, introduces a heuristic that discards dominant features in the label determination, stimulating the model to rely on weaker data correlations. CausIRL [60] (combined with MMD or CORAL) builds from a causal generalization analysis, enforcing soft domain invariance to interventions on the source domain. CAD [61] introduces a contrastive adversarial domain bottleneck to guarantee convergence to target domains that preserve the Bayes predictor. ADDG [62] exploits a double mechanism (Intra-model and Inter-model) to diversify attention between features and suppress domain-related attention.

Data Augmentation Many research works prove that data augmentation plays a fundamental role in DG, as it can partially compensate for specific domain shifts [14]. That is particularly true in the presence of style changes, as popular data augmentation strategies involve altering saturation, hue, and contrast. Since the effect of data augmentation on DG has already been investigated, in this paper, I use a standard setup to keep the focus on backbones. The de facto standard augmentation strategy for DG, which I use in my 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, the input images are further normalized according to the mean and standard deviation of those datasets.

Model Selection To assess the DG capability of the considered pretrained networks, I 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 [31], “a domain generalization algorithm should be responsible for specifying a model selection method” and avoid improper comparisons between results obtained adopting different selection methods. In total agreement with their recommendations, I employ the *training-domain validation set* strategy, which selects the model that maximizes accuracy on a validation split of the training set (in my 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 I conduct a random search for each backbone and dataset to determine the optimal training hyperparameters for the baselines. I 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 experiments, with fixed values of 32 and 30, respectively. Finally, I 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 My benchmarks are developed in Python using PyTorch as the deep learning framework. As the experimentation applies transfer learning to pretrained models, I 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*², EfficientNets from *EfficientNet-PyTorch*³, transformers and ResNet50 A1 from *timm*⁴. The implementations of DG algorithms are taken from DOMAINBED⁵ and adapted to work with the architectures under test. I repeated each training three times with different and randomly generated seeds to give more statistical information about accuracy results. In this way, neither hyperparameter search nor benchmarks can leverage the repeatability of trials, as data splitting, augmentation, and weight initialization vary from one iteration to the next. Therefore, each result of my benchmark is reported as the mean over three repetitions, along with its standard deviation.

7.1.1 Backbones

The first analysis of my work involves a precise and fair benchmark of the deep learning architectures' DG capabilities for image classification, aiming to determine which solutions work best and, possibly, why. Every pretrained backbone,

²pytorch.org/vision/stable/models

³github.com/lukemelas/EfficientNet-PyTorch

⁴github.com/rwightman/pytorch-image-models

⁵github.com/facebookresearch/DomainBed

after a hyperparameter search, is trained following the standard DG *leave-one-domain-out* procedure using the previously described model selection strategy. My benchmark results are reported in Table 7.1 as the mean and standard deviation over three iterations.

Backbone	C	I	P	Q	R	S	Avg
ResNet50 [31]	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 7.2 Baseline comparison of a selection of the best backbones on DomainNet (*Clipart*, *Infograph*, *Painting*, *Quickdraw*, *Real*, and *Sketch* domains). I include the results achieved by DOMAINBED with ResNet50 for reference. The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

Firstly, my benchmark highlights a strong correlation between DG accuracy and ImageNet performance. As depicted in Fig. 7.2, I find a direct proportionality between the two metrics (excluding the ViT models due to their different pre-training). I apply linear least-squares regression and obtain a Pearson correlation coefficient $\rho = 0.921$. Indeed, a quick look at the results reveals how newer and more performing backbones tend to achieve higher DG accuracy on nearly all datasets. That is primarily true for different sizes of the same architecture. ResNet50 achieves better results than ResNet18 for all datasets, and the same holds for EfficientNet, ConViT, and ViT variants. For ResNet50, I also compare my results with those obtained by DOMAINBED and find comparable values. ResNet50 A1 benefits from its stronger pretraining, which largely improves the accuracy obtained by the standard model on VLCS and Office-Home. However, Terra Incognita appears to penalize the network due to its peculiar light conditions, resulting in a slight overall enhancement. Regarding different architectures, EfficientNetB2 performs very similarly to ResNet50, while the B3 version achieves an additional 1%. Transformer-based models bring further improvements by leveraging their self-attention-based feature extraction, even in cases such as DeiT Small and ConViT Small. In particular, they strongly outperform EfficientNet on OfficeHome by over 4% In conclusion, my results indicate that better DG arises from the combination of a suitable feature extractor architecture and optimal pretraining, as neither is sufficient on its own. In Section 7.2, I further discuss

the generalization capability of transformers. I stress the importance of adopting a good model selection strategy by comparing my ResNet18 baseline with recent results obtained using the same backbone.

For an additional comparison, I plot the achieved DG accuracy against the number of parameters of the backbones (Fig. 7.3). Contrary to the graph of Fig. 7.2, in this case, the correlation between model dimension and generalization is much less marked, with a Pearson correlation coefficient $\rho = 0.740$. That confirms the central role of model architecture in DG tasks and my idea of backbone as the union of architecture, training procedure, and data.

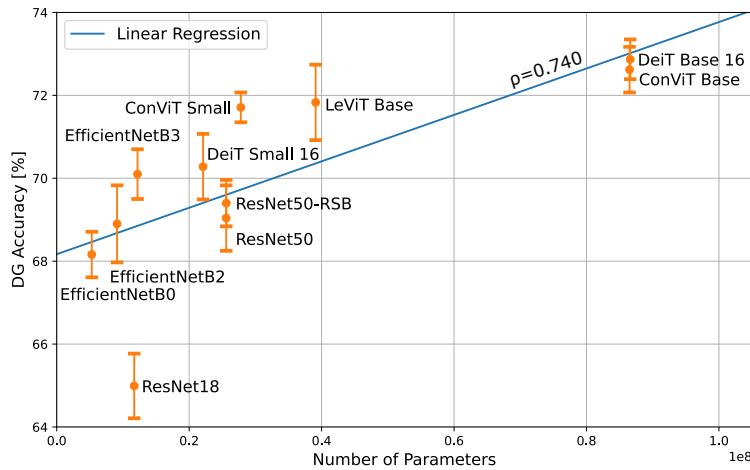


Fig. 7.3 DG accuracy achieved by tested backbones compared with their number of parameters, with error bars. I find a much weaker correlation between the two metrics ($\rho = 0.740$) than the one reported in Fig. 7.2.

Finally, I conduct an additional benchmark on the DomainNet dataset. Although representing a significant challenge for large-scale generalization, I choose to include DomainNet only in this second stage of the study due to its demanding computational nature and strong class unbalancing. Indeed, my primary intention is to establish a practical and accessible benchmark that will become a widely recognized reference for DG. I select only the best three models from the previous tests for this one (DeiT Base 16, ConViT Base, and ViT Base 16). In Table 7.2, I report the results achieved on each test domain, including those obtained by DOMAINBED on ResNet50 for reference. It is evident that the feature extraction capabilities of modern backbones bring substantial improvements across all domains, with an average increase in DG accuracy up to 12.6%. Moreover, ViT further enhances the results by exploiting its stronger pretraining.

7.1.2 Model Introspection

After assessing the DG performance of different backbones, I propose a series of insights on how various architectures utilize training data to create their internal representations. First, I investigate the benefits of ImageNet pretraining for DG with a k-NN classifier, comparing ResNet50 and the best models from my benchmark. Then, I apply t-SNE [49] to the same extracted features to visualize how close same-class and same-domain samples are, as well as the effect of fine-tuning on DG datasets. Finally, I inspect the attention maps of one of the transformer-based models to have a qualitative insight into the region of the images it focuses on.

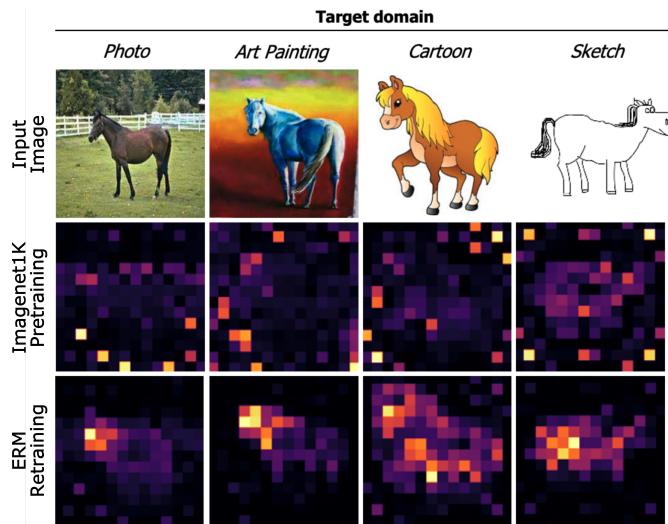


Fig. 7.4 DeiT Base attention maps when using the [CLS] token as a query for the different heads in the last layer. I select the same head for all examples. ERM encourages the backbone to focus on domain-invariant features, highly mitigating pretraining noise.

k-NN Evaluation Firstly, I take ResNet50 and the best-performing models from my benchmark and evaluate their ability to tackle DG without fine-tuning. To do so, I use ImageNet weights to extract features from training domains and a k-NN (with $k = 5$) to fit that data. Then, I use test-domain images for the evaluation. To ensure a fair comparison with my benchmark, I use the same amount of training data, excluding 10% of samples from the source domains. The results in Table 7.3 show an overall difference of about 5% between ResNet50 and transformer-based models pretrained on ImageNet1k. This outcome is consistent with the generalization boost achieved in the standard DG framework (Table 7.1),

although k-NN results tend to oscillate among different datasets. Following this trend, ViT Base 16 achieves 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_{XY}^S is always needed to tackle a substantial domain gap effectively. 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 7.3 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 elucidate the role of backbones in extracting meaningful and invariant features for dealing with DG, I can visualize the distributions in the feature space by projecting them into a two-dimensional space using t-SNE. Fig. 7.5 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, I 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. I also include the silhouette score (S) as a quantitative metric of the separation of classes below each plot.

Fig. 7.5a shows how ResNet50 pretrained on ImageNet tends to map together same-domain samples instead of same-class ones, and is therefore unsuitable for DG without fine-tuning. After the fine-tuning process (Fig. 7.5b), the model achieves a better separation of source domain classes. However, many target domain samples are 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 (Fig. 7.5c), 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 (Fig. 7.5d) distinctly pushes together same-class clusters, resulting in good generalization over the target domain. This analysis suggests that the ConViT backbone is more suited for DG than ResNet50 since it tends to give more similar representations to the same-class samples from different domains.

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 [63]. Indeed, heat maps are presented as evidence of their ability to draw attention to more localized and domain-invariant features. Nevertheless, this section shows that competitive backbones with naive ERM can perfectly localize class-discriminative regions. In particular, Fig. 7.4 displays the attention maps obtained by using the [CLS] token as a query for the different heads in the last layer of the DeiT Base architecture. I 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 redirects attention towards more invariant features.

7.1.3 Domain Generalization Algorithms

Domain generalization research primarily focuses on developing non-trivial algorithms to mitigate the impact of domain shifts on classification accuracy. However, these algorithms are uniquely proposed in combination with outdated backbones such as ResNet50, ResNet18, or even AlexNet. According to the results in Table 7.1, recent backbones can provide significant improvements compared 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 my baselines. To do so, I combine some of the most promising and recent algorithms available on DOMAINBED with three of my best baselines. I evaluate the methods introduced at the beginning of this Section (MMD, CORAL, Mixup, RSC, CAD, CausIRL CORAL, CausIRL MMD, and ADDG) using ViT Base 16, DeiT Base 16, and ConViT Base as backbones, and repeat each training three times. Table 7.5 reports the obtained results, composed of average accuracy and associated standard deviation. Results obtained with ResNet50 are also reported directly from DOMAINBED for the same group of datasets as a reference. The only exception

is the most recent ADDG, which the authors have not tested on VLCS and Terra Incognita, and does not report standard errors.

As highlighted by the values in bold, the overall performance of ERM is equal to or better than that of other DG algorithms for all the considered datasets and backbones. Indeed, even where another methodology slightly outperforms ERM, accuracy results mostly fall in the same confidence interval and differ very little statistically. I can conclude from my experimentation that DG algorithms improve generalization properties marginally or 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. In the next section, I briefly ask ourselves what the reason behind this result is.

7.2 Discussion

7.2.1 Are Transformers Better at Generalizing?

Reflecting on experimental evidence and visual introspection from previous sections, I discuss whether transformer-based backbones are more robust to domain shifts in this paragraph. Undoubtedly, all baseline comparisons of Section 7.1.1 and features visualizations shown in Fig. 7.5 would suggest a positive answer to this intriguing question. Self-attention-based models generalize better to unseen domains in all results and visual representations. This result enforces the finding of [48] that multi-head attention acts as a low-pass filter with a shape bias thanks to its milder prior, while convolution is a high-pass filter with a texture bias.

Nevertheless, exercising caution and critically analyzing all the variables involved in the process is essential. Indeed, such a conclusion only holds leveraging my backbone definition as a function of architecture \mathcal{A} , training procedure \mathcal{T}_B , and data \mathcal{D} . Architecture and training procedure are difficult to disentangle, and there is no guarantee that a training procedure optimal for a specific architecture remains the best for another. Therefore, that implies it is impossible to compare two different architectures directly.

Recent experimentation on residual architectures with current state-of-the-art training procedures has shed light on the contribution of \mathcal{A} to the generalization process. Indeed, in [52], a vanilla ResNet50 is trained using the approach developed by [43], achieving 80.4% top-1 accuracy on ImageNet without the use of extra data or distillation. However, ResNet50 A1 performs only slightly better than the original model on my BACK-TO-BONES testbed, despite a difference of over 4% in ImageNet accuracy. That slightly deviates from the linear correlation described in Section 7.1.1 and suggests that a transformer-based architecture makes a significant contribution to generalization. As further evidence of this trend, ConViT Small has comparable parameters to those of [52] and a similar training procedure, but outperforms it by more than 4% on some datasets. Nonetheless, further experimentation can yield more comprehensive results on this fascinating aspect of vision transformers.

7.2.2 On Baseline Selection in Previous Works

In the last decade, a plethora of algorithms for domain generalization (DG) have been proposed in the literature, trying to tackle the problem with a wide variety of sophisticated methodologies. Nevertheless, my experimentation highlights that the presented baselines often lack proper optimization. Table 7.4 compares the accuracy result obtained in my BACKTOBONES benchmark with those reported by several recent works. I evaluate ResNet18 on PACS, as this is the most common setup, and the baseline I obtain with fair hyperparameter search and validation outperforms all those reported in the latest research works without adding any extra component. Moreover, statistical information is often lacking in past DG works, which have overlooked discussions on proper hyperparameter search and model selection strategies. In accordance with the outcomes of DOMAINBED, I hope to encourage the adoption of rigorous testing procedures, in conjunction with a standard model selection strategy, to ensure transparent research results. This study suggests that new DG algorithms should be analyzed by adopting well-trained backbones. As a matter of fact, an advantage brought to underpowered baselines can be considered meaningless.

Baseline	Average	Std Deviation
TRM [64]	77.13	1.53
MMLD [65]	78.70	-
JiGen [66]	79.05	-
Epi-FCR [67]	79.05	-
MASF [68]	79.23	0.15
SagNet [69]	79.26	-
DDAIG [70]	79.53	0.48
D-SAM [71]	79.55	-
PAdaIN [72]	79.72	-
MetaReg [20]	79.93	0.28
RSC [27]	79.94	-
BACKTOBONES	80.51	0.29

Table 7.4 Comparison between the ResNet18 baseline obtained in my BACKTOBONES benchmark on PACS and those reported by popular DG works. My accuracy result (without any extra component) outperforms all previous ones, which rarely include statistical information from multiple training iterations. Moreover, these works seldom discuss hyperparameter search procedures and model selection strategies.

7.3 Conclusion

In this paper, I have thoroughly investigated the role of backbones in domain generalization, bringing to light the fundamental contribution that a competitive feature extractor provides for generalizing to out-of-distribution data, which has been previously neglected by the community. With this work, I did not add a methodology to the long list, but rather pondered the current situation surrounding DG works, trying to shift towards more effective research. According to my suggested backbone definition, novel architectural solutions, such as DeiT, ConViT, and LeViT, demonstrate remarkable improvements in reducing domain gaps through their intrinsic feature mapping mechanisms. They achieve state-of-the-art results in DG with naive ERM and data augmentation only. Hence, I pointed out that a complete domain generalization study should consider the choice of the backbone as the first step. Moreover, I claimed that the advantage of adopting generalization algorithms should be proved using recent and effectively trained feature extractors.

Building on the foundational insight that the backbone architecture plays a critical role in domain generalization performance, the next chapter moves from

conceptual analysis to methodological innovation. Focusing on semantic segmentation in agriculture, I introduce a practical solution that leverages knowledge distillation and ensemble learning to enhance model robustness across diverse and unseen environments, further advancing the thesis's broader goal of reliable and generalizable AI for real-world autonomous systems.

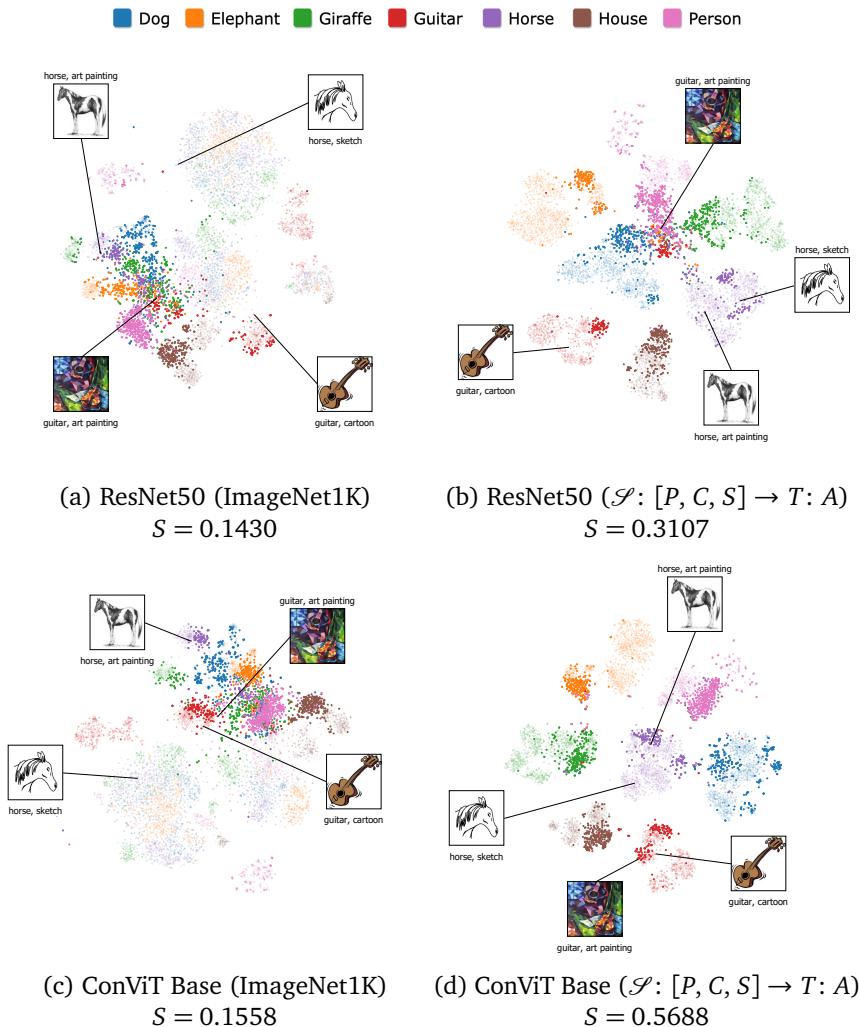


Fig. 7.5 Backbone features visualization with t-SNE on PACS (Photo (P), Art Painting (A), Cartoon (C) and Sketch (S) domains). Target domain samples are highlighted. Some image examples from various domains and classes are visualized to enhance interpretability. After fine-tuning, the ConViT Base architecture achieves better class separation than ResNet50, clustering together same-class samples from different domains.

Backbone	Algorithm	PACS	VLCS	Office-Home	Terra Incognita	Overall
ResNet50 [31]	ERM [73]	85.50 ± 0.20	77.50 ± 0.40	66.50 ± 0.30	46.10 ± 1.80	68.90 ± 0.68
	RSC [27]	85.20 ± 0.90	77.10 ± 0.50	65.50 ± 0.90	46.60 ± 1.00	68.60 ± 0.83
	Mixup [59]	84.60 ± 0.60	77.40 ± 0.60	68.10 ± 0.30	47.90 ± 0.80	69.50 ± 0.58
	CORAL [16]	86.20 ± 0.30	78.80 ± 0.60	68.70 ± 0.30	47.60 ± 1.00	70.33 ± 0.55
	MMD [18]	84.60 ± 0.50	77.50 ± 0.90	66.30 ± 0.10	42.20 ± 1.60	67.65 ± 0.78
	CausIRL CORAL [60]	85.80 ± 0.10	77.50 ± 0.60	68.60 ± 0.30	47.30 ± 0.80	69.80 ± 0.45
	CausIRL MMD [60]	84.00 ± 0.80	77.60 ± 0.40	65.70 ± 0.60	46.30 ± 0.90	68.40 ± 0.68
	CAD [61]	85.20 ± 0.90	78.00 ± 0.50	67.40 ± 0.20	47.30 ± 2.20	69.48 ± 0.95
	ADDG [62]	89.2	-	72.5	-	-
DeiT Base 16	ERM [73]	88.10 ± 0.48	79.80 ± 0.32	76.35 ± 0.36	47.22 ± 0.75	72.87 ± 0.48
	RSC [27]	85.37 ± 1.30	77.27 ± 0.51	76.47 ± 0.28	45.41 ± 1.50	70.97 ± 0.90
	Mixup [59]	85.67 ± 0.61	78.25 ± 0.60	75.96 ± 0.11	46.63 ± 0.49	71.32 ± 0.48
	CORAL [16]	85.13 ± 0.82	78.34 ± 0.86	76.48 ± 0.14	46.33 ± 1.83	71.38 ± 0.93
	MMD [18]	87.22 ± 0.28	78.71 ± 0.22	77.03 ± 0.10	49.35 ± 1.42	73.08 ± 0.50
	CausIRL CORAL [60]	83.86 ± 0.75	77.80 ± 0.40	76.12 ± 0.04	46.73 ± 0.81	71.13 ± 0.50
	CausIRL MMD [60]	85.46 ± 0.68	77.27 ± 0.42	76.53 ± 0.42	45.77 ± 1.66	71.26 ± 0.79
	CAD [61]	87.74 ± 0.62	79.28 ± 0.36	76.61 ± 0.15	47.46 ± 0.64	72.77 ± 0.44
	ADDG [62]	75.30 ± 0.34	78.28 ± 0.77	77.58 ± 0.30	29.14 ± 2.24	65.07 ± 0.91
ConViT Base	ERM [73]	87.27 ± 0.40	80.31 ± 0.67	76.51 ± 0.25	46.37 ± 0.89	72.62 ± 0.55
	RSC [27]	85.73 ± 0.81	79.05 ± 0.61	76.77 ± 0.26	44.94 ± 1.47	71.62 ± 0.79
	Mixup [59]	86.00 ± 0.45	80.00 ± 0.76	76.48 ± 0.16	43.95 ± 0.18	71.61 ± 0.39
	CORAL [16]	86.24 ± 0.24	79.62 ± 0.38	75.33 ± 0.22	44.41 ± 1.33	71.40 ± 0.54
	MMD [18]	86.84 ± 0.63	80.72 ± 0.55	77.94 ± 0.31	46.78 ± 1.22	73.07 ± 0.68
	CausIRL CORAL [60]	84.71 ± 0.31	79.14 ± 0.69	77.05 ± 0.16	45.63 ± 2.03	71.63 ± 0.80
	CausIRL MMD [60]	86.59 ± 0.96	80.30 ± 0.56	77.92 ± 0.35	46.85 ± 0.59	72.92 ± 0.61
	CAD [61]	87.42 ± 0.66	79.99 ± 0.41	77.71 ± 0.09	46.77 ± 3.31	72.97 ± 1.12
	ADDG [62]	86.34 ± 0.76	79.79 ± 0.30	76.29 ± 0.33	43.97 ± 1.75	71.60 ± 0.78
ViT Base 16*	ERM [73]	88.48 ± 1.22	80.05 ± 0.15	81.47 ± 0.21	49.77 ± 1.28	74.94 ± 0.72
	RSC [27]	86.58 ± 2.14	79.59 ± 0.63	78.74 ± 0.64	40.79 ± 1.41	71.42 ± 1.20
	Mixup [59]	88.62 ± 0.54	80.77 ± 1.28	82.93 ± 0.07	48.59 ± 0.92	75.23 ± 0.70
	CORAL [16]	84.60 ± 1.31	80.89 ± 0.49	80.92 ± 0.25	50.58 ± 0.26	74.25 ± 0.58
	MMD [18]	87.99 ± 0.08	79.54 ± 0.37	81.71 ± 0.28	49.40 ± 2.45	74.66 ± 0.79
	CausIRL CORAL [60]	88.26 ± 1.09	80.10 ± 0.91	81.73 ± 0.13	47.29 ± 2.64	74.35 ± 1.19
	CausIRL MMD [60]	86.57 ± 1.13	79.48 ± 1.12	81.62 ± 0.22	49.52 ± 0.58	74.30 ± 0.76
	CAD [61]	87.44 ± 0.53	78.79 ± 2.43	79.80 ± 0.36	39.45 ± 4.15	71.37 ± 1.87
	ADDG [62]	75.33 ± 0.54	77.77 ± 0.32	77.72 ± 0.09	25.60 ± 0.64	64.11 ± 0.40

Table 7.5 Comparison between ERM and three promising DG algorithms on the best-performing backbones of my benchmark. I report the average accuracy over three runs, along with the associated standard deviation, for each model. I highlight in bold the best result for each dataset, including ERM, when its accuracy is in the same confidence interval. I include the results achieved by DOMAINBED with ResNet50 for reference. The model marked with * is pretrained on Imagenet21K instead of ImageNet1K.

Bibliography

- [1] S. Angarano, M. Martini, F. Salvetti, V. Mazzia, and M. Chiaberge, “Back-to-bones: Rediscovering the role of backbones in domain generalization,” *Pattern Recognition*, vol. 156, p. 110762, 2024.
- [2] L. Valiant, *Probably Approximately Correct: Nature’s Algorithms for Learning and Prospering in a Complex World*. Basic Books (AZ), 2013.
- [3] G. Csurka, *Domain adaptation in computer vision applications*. Springer, 2017.
- [4] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness.,” in *International Conference on Learning Representations*, 2019.
- [5] L. Gatys, A. S. Ecker, and M. Bethge, “Texture synthesis using convolutional neural networks,” in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015.
- [6] M. Arjovsky, L. Bottou, I. Gulrajani, and D. Lopez-Paz, “Invariant risk minimization,” *arxiv: 1907.02893*, 2019.
- [7] D. Dai and L. Van Gool, “Dark model adaptation: Semantic image segmentation from daytime to nighttime,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 3819–3824.
- [8] G. Volk, S. Müller, A. von Bernuth, D. Hospach, and O. Bringmann, “Towards robust cnn-based object detection through augmentation with synthetic rain variations,” in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, IEEE, 2019, pp. 285–292.
- [9] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “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.
- [10] M. Mozifian, A. Zhang, J. Pineau, and D. Meger, “Intervention design for effective sim2real transfer,” *arxiv: 2012.02055*, 2020.
- [11] G. Blanchard, G. Lee, and C. Scott, “Generalizing from several related classification tasks to a new unlabeled sample,” *Advances in neural information processing systems*, vol. 24, pp. 2178–2186, 2011.

- [12] K. Muandet, D. Balduzzi, and B. Schölkopf, “Domain generalization via invariant feature representation,” in *International Conference on Machine Learning*, PMLR, 2013, pp. 10–18.
- [13] S. Shankar, V. Piratla, S. Chakrabarti, S. Chaudhuri, P. Jyothi, and S. Sarawagi, “Generalizing across domains via cross-gradient training,” in *International Conference on Learning Representations*, 2018.
- [14] R. Volpi, H. Namkoong, O. Sener, J. C. Duchi, V. Murino, and S. Savarese, “Generalizing to unseen domains via adversarial data augmentation,” in *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [15] Y. Ganin *et al.*, “Domain-adversarial training of neural networks,” *The journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.
- [16] B. Sun and K. Saenko, “Deep CORAL: Correlation alignment for deep domain adaptation,” in *European conference on computer vision*, Springer, 2016, pp. 443–450.
- [17] S. Motian, M. Piccirilli, D. A. Adjeroh, and G. Doretto, “Unified deep supervised domain adaptation and generalization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5715–5725.
- [18] H. Li, S. J. Pan, S. Wang, and A. C. Kot, “Domain generalization with adversarial feature learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5400–5409.
- [19] S. Chen, L. Wang, Z. Hong, and X. Yang, “Domain generalization by joint-product distribution alignment,” *Pattern Recognition*, vol. 134, p. 109 086, 2023, ISSN: 0031-3203.
- [20] Y. Balaji, S. Sankaranarayanan, and R. Chellappa, “Metareg: Towards domain generalization using meta-regularization,” *Advances in Neural Information Processing Systems*, vol. 31, pp. 998–1008, 2018.
- [21] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, “Learning to generalize: Meta-learning for domain generalization,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [22] M. M. Zhang, H. Marklund, N. Dhawan, A. Gupta, S. Levine, and C. Finn, “Adaptive risk minimization: A meta-learning approach for tackling group shift,” in *International Conference on Learning Representations*, 2020.
- [23] S. Bucci, A. D’Innocente, Y. Liao, F. M. Carlucci, B. Caputo, and T. Tommasi, “Self-supervised learning across domains,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5516–5528, 2022.
- [24] I. Albuquerque, N. Naik, J. Li, N. Keskar, and R. Socher, “Improving out-of-distribution generalization via multi-task self-supervised pretraining,” *arxiv: 2003.13525*, 2020.
- [25] M. M. Rahman, C. Fookes, M. Baktashmotagh, and S. Sridharan, “Correlation-aware adversarial domain adaptation and generalization,” *Pattern Recognition*, vol. 100, p. 107 124, 2020, ISSN: 0031-3203.

- [26] S. Sagawa, P. W. Koh, T. B. Hashimoto, and P. Liang, “Distributionally robust neural networks,” in *International Conference on Learning Representations*, 2020.
- [27] Z. Huang, H. Wang, E. P. Xing, and D. Huang, “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.
- [28] S. Shahtalebi, J.-C. Gagnon-Audet, T. Laleh, M. Faramarzi, K. Ahuja, and I. Rish, “Sandmask: An enhanced gradient masking strategy for the discovery of invariances in domain generalization,” *arxiv: 2106.02266*, 2021.
- [29] D. Kim, Y. Yoo, S. Park, J. Kim, and J. Lee, “Selfreg: Self-supervised contrastive regularization for domain generalization,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 9619–9628.
- [30] M. Segu, A. Tonioni, and F. Tombari, “Batch normalization embeddings for deep domain generalization,” *Pattern Recognition*, vol. 135, p. 109 115, 2023, ISSN: 0031-3203.
- [31] I. Gulrajani and D. Lopez-Paz, “In search of lost domain generalization,” in *International Conference on Learning Representations*, Computer Vision Foundation, 2021.
- [32] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [33] O. Elharrouss, Y. Akbari, N. Almaadeed, and S. Al-Maadeed, “Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches,” *arxiv: 2206.08016*, 2022.
- [34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [36] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.
- [37] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [38] C. Szegedy *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [39] A. G. Howard *et al.*, “Mobilennets: Efficient convolutional neural networks for mobile vision applications,” *arxiv: 1704.04861*, 2017.

- [40] M. Tan and Q. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” in *International Conference on Machine Learning*, PMLR, 2019, pp. 6105–6114.
- [41] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [42] A. Dosovitskiy *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations*, 2021.
- [43] H. Tuvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, “Training data-efficient image transformers & distillation through attention,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 10347–10357.
- [44] S. D’Ascoli, H. Tuvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun, “Convit: Improving vision transformers with soft convolutional inductive biases,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 2286–2296.
- [45] B. Graham *et al.*, “Levit: A vision transformer in convnet’s clothing for faster inference,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12259–12269.
- [46] M. Sultana, M. Naseer, M. H. Khan, S. Khan, and F. S. Khan, “Self-distilled vision transformer for domain generalization,” in *Proceedings of the Asian Conference on Computer Vision*, 2022, pp. 3068–3085.
- [47] J. Guo, N. Wang, L. Qi, and Y. Shi, “Aloft: A lightweight mlp-like architecture with dynamic low-frequency transform for domain generalization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 24132–24141.
- [48] B. Li *et al.*, “Sparse mixture-of-experts are domain generalizable learners,” in *The Eleventh International Conference on Learning Representations*, 2022.
- [49] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [50] J. Tobin *et al.*, “Domain randomization and generative models for robotic grasping,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3482–3489.
- [51] A. Loquercio, E. Kaufmann, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, “Deep drone racing: From simulation to reality with domain randomization,” *IEEE Transactions on Robotics*, vol. 36, no. 1, pp. 1–14, 2019.
- [52] R. Wightman, H. Tuvron, and H. Jegou, “Resnet strikes back: An improved training procedure in timm,” in *NeurIPS 2021 Workshop on ImageNet: Past, Present, and Future*, 2021.

- [53] C. Fang, Y. Xu, and D. 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*, 2013, pp. 1657–1664.
- [54] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, “Deeper, broader and artier domain generalization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5542–5550.
- [55] H. Venkateswara, J. Eusebio, S. Chakraborty, and S. Panchanathan, “Deep hashing network for unsupervised domain adaptation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5018–5027.
- [56] S. Beery, G. Van Horn, and P. Perona, “Recognition in terra incognita,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 456–473.
- [57] X. Peng, Q. Bai, X. Xia, Z. Huang, K. Saenko, and B. Wang, “Moment matching for multi-source domain adaptation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1406–1415.
- [58] M. Ghifary, W. B. Kleijn, M. Zhang, and D. Balduzzi, “Domain generalization for object recognition with multi-task autoencoders,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2551–2559.
- [59] S. Yan, H. Song, N. Li, L. Zou, and L. Ren, “Improve unsupervised domain adaptation with mixup training,” *arxiv: 2001.00677*, 2020.
- [60] M. Chevalley, C. Bunne, A. Krause, and S. Bauer, “Invariant causal mechanisms through distribution matching,” *arxiv: 2206.11646*, 2022.
- [61] Y. Ruan, Y. Dubois, and C. J. Maddison, “Optimal representations for covariate shift,” in *International Conference on Learning Representations*, 2022.
- [62] R. Meng *et al.*, “Attention diversification for domain generalization,” in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIV*, Springer, 2022, pp. 322–340.
- [63] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “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.
- [64] Y. Xu and T. Jaakkola, “Learning representations that support robust transfer of predictors,” *arxiv: 2110.09940*, 2021.
- [65] T. Matsuura and T. Harada, “Domain generalization using a mixture of multiple latent domains,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 11749–11756.
- [66] F. M. Carlucci, A. D’Innocente, S. Bucci, B. Caputo, and T. Tommasi, “Domain generalization by solving jigsaw puzzles,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2229–2238.

- [67] D. Li, J. Zhang, Y. Yang, C. Liu, Y.-Z. Song, and T. M. Hospedales, “Episodic training for domain generalization,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.
- [68] Q. Dou, D. Coelho de Castro, K. Kamnitsas, and B. Glocker, “Domain generalization via model-agnostic learning of semantic features,” *Advances in Neural Information Processing Systems*, vol. 32, pp. 6450–6461, 2019.
- [69] H. Nam, H. Lee, J. Park, W. Yoon, and D. Yoo, “Reducing domain gap by reducing style bias,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8690–8699.
- [70] K. Zhou, Y. Yang, T. Hospedales, and T. Xiang, “Deep domain-adversarial image generation for domain generalisation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 13 025–13 032.
- [71] A. D’Innocente and B. Caputo, “Domain generalization with domain-specific aggregation modules,” in *German Conference on Pattern Recognition*, Springer, 2018, pp. 187–198.
- [72] O. Nuriel, S. Benaim, and L. Wolf, “Permuted adain: Reducing the bias towards global statistics in image classification,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9482–9491.
- [73] V. N. Vapnik, “An overview of statistical learning theory,” *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.

Chapter 8

Domain Generalization with Standardized Ensemble Knowledge Distillation

Original Paper: *S. Angarano, M. Martini, A. Navone, and M. Chiaberge, “Domain Generalization for Crop Segmentation with Standardized Ensemble Knowledge Distillation”, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024.* [1]

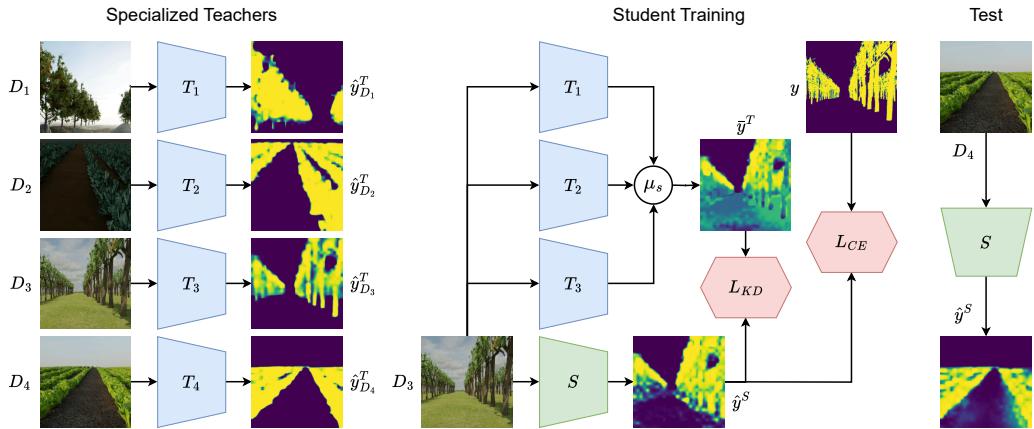


Fig. 8.1 Schematic representation of the proposed distillation methodology for crop segmentation. Ensembled specialized teachers allow the student to obtain a standardized distillation mask (\bar{y}^T) that is much more informative than the hard label (y) for robust student training. μ_s represents standardized ensembling.



Fig. 8.2 From left to right: examples of synthetic 3D crop models used to build the AGRISEG Dataset (*Generic Tree 2, Zucchini, Lettuce, Vineyard*); examples of resulting dataset images (*Vineyard, Chard*); examples of real-world test images (*Vineyard, Miscellaneous*).

In the last two decades, scientific research in precision agriculture has significantly evolved its automatic and self-managed processes. Automation has been analyzed through four essential requirements: increasing productivity, allocating resources reasonably, adapting to climate change, and avoiding food waste [2]. Recently, deep learning solutions led to new technological trends in all these tasks, providing competitive advantages for crop monitoring and managing [3]. Autonomous robots equipped with perception systems can assist or replace human operators in agricultural tasks such as harvesting [4], spraying [5], and vegetative assessment [6], reducing human labor and enhancing operational safety. Various computer vision methods have been proposed for navigating and monitoring row crops, most of which are based on semantic segmentation [7]. Real-time crop segmentation can be used to identify objects on different scales: detailed leaf disease [8], single fruits or branches [9], crop rows [10], and entire fields [11]. It has also been exploited for autonomous navigation [10], combined with waypoint generation [12] or sensorimotor agents [13].

However, crop segmentation presents two main challenges. First, changing weather, lighting, terrain, and crop types significantly hinder generalization. Supervised training methodologies typically yield remarkable results in well-defined experimental settings but struggle to achieve good results when the data distribution changes [14]. However, robustness in realistic scenarios can be enhanced using frameworks like domain generalization (DG). DG is a set of representation learning techniques that aims to train models capable of generalizing to unseen domains, i.e., out-of-distribution data. Several DG methodologies have been

presented in recent years, although often limiting their scope to classification on toy datasets [15]. Applying generalization methods to realistic tasks is still limited to a few attempts [16, 17, 18]. Moreover, the considered domains are often limited to stylistic changes, overlooking more radical correlation shifts [19]. For instance, in specific scenarios, brown and green colors are positively correlated with terrain and vegetation. However, other domains present brown tree trunks and grass on the ground, inverting the correlation.

A second challenge is data availability, as no comprehensive dataset exists for crop segmentation across multiple scenarios. The reason is that on-field data collection and labeling are highly time-demanding. Hence, the only publicly available datasets focus on specific scenarios and usually include a modest number of samples. The scarcity of task-specific labeled data has recently led to the practice of synthetic data generation, creating an additional Simulation-to-Reality (Sim2Real) gap and further compromising generalization [20].

Generalization to Out-of-domain (OoD) data distributions is one of the most critical requirements for real-world computer vision applications, such as crop segmentation. Recently, rigorous validation benchmarks have been proposed to compare the advantages of different approaches and backbones for classification [21, 22]. In the meantime, segmentation across multiple scenarios has been studied, either designing massive foundation models [23] or creating new DG methods. As I aim to push the limits of generalization for efficient and easily deployable architectures, I focus on the latter approach. In particular, [24] proposed an Instance Batch Normalization (IBN) block for residual modules to avoid bias toward low-level domain-specific features like color, contrast, and texture. [25], on the same line, proposed a permuted Adaptive Instance Normalization (PAdaIN) block, which works at both low-level and high-level features, randomly swapping second-order statistics between source domains and hence regularizing the network towards invariant features. [16] proposed RobustNet, a model incorporating an Instance Selective Whitening (ISW) loss, disentangling and removing the domain-specific style in feature covariance. [17] proposed to extract domain-generalized features by leveraging a variety of contents and styles using a “wild” dataset. Most recently, [26] has been the first attempt to apply KD in the DG framework for classification tasks, proposing a gradient filtering approach. [27] proposed Cross-domain Ensemble Distillation (XDED) to extract the knowledge from domain-specific teachers and obtain a general student. However, this setup

was only applied to classification, while the authors used a different approach for segmentation distilling from a single teacher. Standard DG benchmarks primarily focus on domestic environments or autonomous driving [28, 29], and generalization for crop segmentation has been addressed only recently. In particular, [30] proposes a style transfer method for robust weed segmentation, focusing on a single crop type. [31] proposes supervised Domain Adaptation for row crop segmentation, requiring target-domain labeled data. I push the generalization concept further, including not only weather and lighting conditions but also aiming to generalize to unseen crop types without prior knowledge about the target data distribution.

This work [1] aims to effectively enhance DG in crop segmentation, working towards having a single model that can generalize across different crop types and environmental conditions. It is well known that supervised neural networks exploit spurious correlations to find shortcuts in data and efficiently minimize the loss function [32]. In agricultural scenarios, these correlations can be easily observed in the color of a specific species, low-level terrain textures, or the background. I apply the DG framework to encourage models to learn deep, robust features without prior knowledge of the target data distribution. Moreover, recent findings have given a theoretical interpretation of the efficacy of model ensembling and knowledge distillation (KD) for robust representation learning [33]. Multiple features exist in data samples that can be used to classify them correctly, and this multi-view structure constitutes the “dark knowledge” that ensembles and KD exploit, explaining the efficacy of these methods. I investigate whether such property enhances domain and Sim2Real generalization, particularly for crop segmentation.

The proposed method distills knowledge from an ensemble of models, individually trained on source domains, into a student model that can adapt to unseen target domains, as depicted in Fig. 8.1. To effectively balance the contributions of the teachers, I standardize their output logits, thereby avoiding overconfident predictions and facilitating effective knowledge transfer. To properly validate the proposed method, I present the synthetic multi-domain dataset for crop segmentation AGRISeg, which comprises 11 crop types and covers various terrain styles, weather conditions, and lighting scenarios, totaling more than 70,000 samples. I conduct thorough experiments on AGRISeg and additional real-world datasets to verify the effectiveness of my method compared to other state-of-the-art solu-

tions. The code¹ used for the experiments and the AGRISEG dataset² are publicly available.

8.1 Methodology

I propose a simple yet effective training procedure based on model ensemble, KD, and logit standardization to encourage the model to learn domain-invariant features. I chose ensemble KD, inspired by the recent theory presented in [33] on multi-view extraction from data. Ensemble KD has been previously applied to classification in XDED [27], leveraging the separate pretraining of a teacher for each source domain and distilling the ensembled predicted logits. I aim to apply the same intuition to crop semantic segmentation, considering the differences between the two tasks and the additional challenges given by the agricultural setting and the Sim2Real gap. Another critical challenge of this application scenario is that domain shifts are not only provided by style transfer but also by the presence of completely different crop types.

In my proposed method, I train a teacher for each source domain and ensemble them to create the distilled knowledge:

$$\bar{y}^T(\mathbf{x}) = \frac{1}{D} \sum_{d=1}^D \hat{y}_d^T(\mathbf{x}) \quad (8.1)$$

where $\hat{y}_d^T(\mathbf{x})$ is the predicted logits tensor for the source domain d , \bar{y}^T is the ensembled teacher logits tensor, and D is the number of source domains. The motivation behind this choice is that by averaging the predictions of different specialized models, the resulting map is much more informative than the ground-truth label. As depicted in Fig. 8.3, the teacher’s segmentation is less confident and often assigns non-zero probabilities to disturbing elements such as grass and background vegetation. This spurious information guides the student towards implicitly recognizing which features are more likely to be confounding at test time. This information does not overcome label supervision, as the distillation loss has a relatively low weight in the optimization process. On the contrary, if the distillation mask is very confident, the student is guided toward being more

¹github.com/PIC4SeR/AgriSeg

²pic4ser.polito.it/agriseg

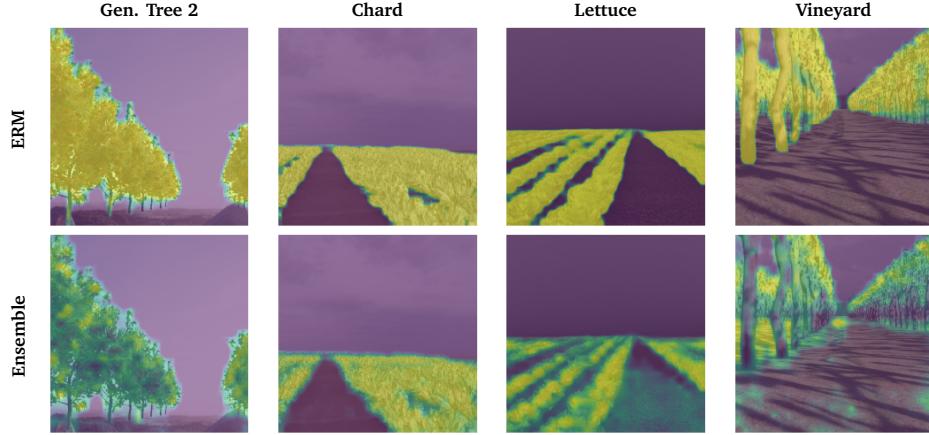


Fig. 8.3 Comparison of ERM predictions with my ensemble of specialized teachers. For simpler domains, the predictions of the specialized teachers agree and return a high-confidence mask. In contrast, for challenging ones, the teachers provide an uncertain but more informative mask that can be distilled for the student.

confident and implicitly incorporates the information that a particular domain is easier to segment. This effect can be tuned using a temperature factor and a weight loss.

I train the student using the standard ERM DG framework with an additional distillation loss based on the distance between the student's logits and the ensemble of the teacher's logits. To improve the effectiveness of distillation, I designed a simple mechanism to prevent students from being biased by teachers' overconfidence. Indeed, each teacher is trained on a single domain and can therefore fall prey to spurious correlations in the training data (e.g., color bias). I thus propose to standardize teacher and student logits [34] before distilling as follows:

$$\tilde{y}^T = \frac{y^T - \bar{y}^T}{\sigma(y^T) \cdot \tau}, \quad \tilde{y}^S = \frac{\hat{y}^S - \bar{y}^S}{\sigma(\hat{y}^S) \cdot \tau} \quad (8.2)$$

where \bar{y} is the mean, $\sigma(y)$ is the standard deviation of the logits, and τ is the temperature. The intuition behind this choice is that optimal τ could vary across domains due to the teachers being more or less confident about their predictions. Standardization enables adaptive calibration of logit temperature, facilitating effective domain knowledge transfer.

The distillation loss is calculated as the Kullback-Leibler divergence between teacher and student logits:

$$L_{\text{KD}}(\tilde{y}^T, \tilde{y}^S) = \frac{\tau^2}{C} \sum_{c=1}^C \sum_{i=1}^{W \cdot H} \phi(\tilde{y}_{c,i}^T) \cdot \log\left(\frac{\phi(\tilde{y}_{c,i}^T)}{\phi(\tilde{y}_{c,i}^S)}\right) \quad (8.3)$$

where C is the number of classes and τ is the temperature.

In combination with the distillation loss, I optimize the standard cross-entropy loss between student logits and ground-truth labels y :

$$L_{\text{CE}}(y, \hat{y}^S) = - \sum_{i=1}^C y_i \cdot \log(\hat{y}_i^S) \quad (8.4)$$

which for binary segmentation becomes a simple binary cross-entropy loss. The overall loss can be written as follows:

$$L(y, \bar{y}^T, \hat{y}^S) = L_{\text{CE}}(\bar{y}, \hat{y}^S) + \lambda L_{\text{KD}}(\tilde{y}^T, \tilde{y}^S) \quad (8.5)$$

where λ is a weighting parameter to balance the loss components. I remark that my method adds no overhead at test time. I provide a thorough ablation of the various components of my method in Section 8.2.4 to highlight the strong improvement on previous solutions.

8.2 Experiments

This section describes the details of the proposed synthetic AGRISEG segmentation dataset and the procedure I followed to validate the effectiveness of my DG methodology. In Section 8.2.1, I review the procedure followed to generate the AGRISEG dataset, while in Section 8.2.2, details on the training framework and implementation are given.

Crop	Samples	Type	Category↓	Height [m]
<i>Lettuce</i>	4,800	Synthetic	Low	0.22
<i>Chard</i>	4,800	Synthetic	Low	0.25
<i>Lavender</i>	5,260	Synthetic	Low	0.3
<i>Zucchini</i>	19,200	Synthetic	Medium	0.6
<i>Cotton</i>	4,800	Synthetic	Medium	0.6
<i>Vineyard</i>	4,800	Synthetic	Tall	1.5-2.5
<i>Pergola Vineyard</i>	4,800	Synthetic	Tall	3.2
<i>Apple Tree</i>	9,600	Synthetic	Tall	2.7
<i>Pear Tree</i>	4,800	Synthetic	Tall	3.0
<i>Generic Tree 1</i>	4,800	Synthetic	Tall	4.5
<i>Generic Tree 2</i>	2,785	Synthetic	Tall	4.5
<i>Vineyard</i> [10]	500	Real	Tall	2.5
<i>Miscellaneous</i>	100	Real	Any	Any
<i>VegAnn</i> [35]	3,775	Real	Any	Any

Table 8.1 Detailed properties for each domain of the AGRISEG dataset. The section on the top reports the synthetic crop datasets generated in simulation, while the section on the bottom reports the real-world ones.

8.2.1 Dataset

High-quality 3D plant models have been created using Blender³ to generate the synthetic crop dataset with realistic plant textures and measurements. A wide variety of crops have been included in the dataset to validate the segmentation performance of the model trained with the proposed DG method. Depending on the plant’s height, three primary macro-categories of crops have been identified. Low crops, such as *Lettuce* and *Chard*, have an average height of 20-25 cm; Medium crops, such as zucchini, grow to 60 cm; Tall crops, which include vineyards and trees, can grow up to 2.5-4.5 m. Some examples of 3D plant models are shown in Fig. 8.2.

Various terrains and sky models have been utilized to create realistic backgrounds and lighting conditions. Afterward, Blender’s Python scripting functionality was used to automatically separate plants from the rest of the frame and generate a dataset of RGB images and corresponding segmentation masks. This work presents the AGRISEG dataset, composed of samples from low crops (e.g., chard and lettuce), medium crops (e.g., zucchini), and tall crops (e.g., vineyard,

³blender.org

Method	Gen. Tree 2	Chard	Lettuce	Vineyard	Average
Teacher	84.52 ± 0.62	95.09 ± 0.11	95.37 ± 0.10	85.51 ± 1.08	90.12 ± 0.48
ERM[36]	76.31 ± 1.53	87.63 ± 0.86	80.64 ± 5.00	67.34 ± 2.18	77.98 ± 1.62
IBN[24]	79.15 ± 1.57	88.92 ± 1.07	57.85 ± 6.00	69.11 ± 2.88	73.76 ± 2.36
ISW[16]	77.14 ± 1.77	89.44 ± 0.47	53.86 ± 8.45	68.76 ± 3.06	72.30 ± 2.30
pAdaIN[25]	75.17 ± 2.03	86.65 ± 1.75	78.05 ± 4.90	69.79 ± 1.41	77.41 ± 0.77
WildNet[17]	82.34 ± 1.55	93.68 ± 0.08	43.55 ± 6.91	72.83 ± 0.76	73.10 ± 2.70
CWD[37]	64.20 ± 8.08	84.70 ± 1.92	83.84 ± 2.79	62.88 ± 2.49	73.90 ± 2.17
WCTA[30]	75.09 ± 0.94	86.66 ± 1.93	70.73 ± 10.85	66.57 ± 3.20	74.76 ± 2.25
KDDG[26]	80.13 ± 1.61	87.67 ± 1.66	74.16 ± 3.37	65.55 ± 1.18	76.88 ± 0.68
XDED[27]	77.18 ± 2.20	88.62 ± 0.71	75.82 ± 5.12	70.48 ± 1.00	78.03 ± 1.85
Ours	78.84 ± 1.24	88.35 ± 1.27	78.04 ± 3.46	72.21 ± 1.02	$\underline{\textbf{79.36}} \pm \underline{\textbf{1.04}}$

Table 8.2 Comparison between the proposed methodology and other state-of-the-art DG algorithms for semantic segmentation adopting the leave-one-out DG validation procedure described in Section 8.2.2. I report the Intersection-over-Union (IoU) metric (in %) for each result as mean and standard deviation. The best and second-best overall results are highlighted and underlined, respectively.

pear tree, and generic tall tree). Each dataset comprises four sub-datasets that differ in terms of background and terrain. Cloudy and sunny skies, as well as diverse lighting and shadow conditions, are included. The camera position and orientation have been adjusted to capture diverse image samples across the entire field for each sub-dataset. Overall, the AGRISEG dataset contains more than 70,000 samples. In the bottom rows, I also include three additional domains to validate the considered solutions on real-world data as a final test. The *Real Vineyard* dataset was initially presented in [10], but the proposed labels were coarse. Hence, I re-label the samples using the *SALT* labeling tool⁴ based on Segment Anything [23]. I also add *Miscellaneous*, containing 100 samples from disparate crop types, and label it similarly. Finally, I include *VegAnn* [35], a multi-crop dataset acquired under diverse conditions for vegetation segmentation. This domain constitutes a highly different setting from the training domains, so I use it to evaluate generalization in extreme domain shifts. Details for each dataset are listed in Table 8.1.

⁴github.com/anuragxel/salt

8.2.2 Experimental Setting

In this section, I report all relevant information regarding the experimental settings for model training and testing, including data preprocessing, hyperparameter search, and implementation. I repeat each training with different and randomly generated seeds five times to obtain statistically relevant metrics. Each of my benchmark results is reported with its mean and standard deviation.

Data Preprocessing

I preprocess the input images using the ImageNet standard normalization [38] to utilize pre-trained weights. I apply the same data augmentation to all experiments, consisting of random cropping with a factor in the range of $[0.5, 1]$ and flipping with a probability of 50%. I don't use random jitter, contrast, and grayscale, which are standard in DG. I instead draw inspiration for the WCTA stylization method proposed in [30] for weed segmentation. I apply it randomly with probability $p = 0.001$ and refer to my version as pWCTA. The reason is that I don't tackle just a shift in style, but also in context (the change in crop type), and stronger stylization could overregularize training. Experiments confirm that my choice leads to enhanced generalization on the proposed dataset.

Hyper-parameters

I conduct a random search to determine the optimal training hyperparameters for the ERM DG baseline. I define a range of values for continuous arguments and a set of choices for discrete ones and select the best combination via the *training-domain validation set* strategy proposed in [21]. It consists of choosing the model that maximizes the metric (in my case, IoU) on a validation split of the training set (in my case, 10%, uniformly distributed across domains) at the end of each epoch.

I choose a batch size $B = 64$ and set the number of training epochs to 50. I choose temperature $\tau = 2$ and weight $\lambda = 0.1$ for L_{KD} . I use AdamW [39] as the optimizer with a weight decay of 10^{-5} . The learning rate is scheduled to decay polynomially between 5×10^{-5} and 5×10^{-6} . I compare to state-of-the-art methodologies, running the same hyperparameter search when tuning is

Method	Pear Tree	Zucchini	Real Vineyard	Real Misc.	VegAnn	Average
Teacher	90.84 ± 0.30	90.42 ± 0.11	69.20 ± 2.86	54.39 ± 3.71	85.70 ± 0.88	78.11 ± 1.57
ERM[36]	82.11 ± 0.93	86.11 ± 0.15	51.51 ± 7.27	67.48 ± 0.97	61.92 ± 0.93	69.83 ± 1.33
IBN[24]	82.24 ± 0.64	86.06 ± 0.07	52.04 ± 3.98	67.97 ± 2.21	63.06 ± 1.51	70.27 ± 1.68
ISW[16]	82.31 ± 0.83	86.03 ± 0.11	55.46 ± 2.83	67.68 ± 2.04	63.29 ± 1.42	70.95 ± 1.45
pAdaIN[25]	82.67 ± 0.77	85.96 ± 0.20	53.02 ± 6.26	63.39 ± 2.00	62.56 ± 1.26	69.52 ± 2.10
WildNet[17]	88.50 ± 0.31	86.32 ± 0.10	37.30 ± 0.61	72.15 ± 0.27	42.62 ± 1.36	65.38 ± 0.53
CWD[37]	79.52 ± 0.54	85.84 ± 0.07	50.83 ± 4.37	65.27 ± 3.28	61.18 ± 1.21	68.53 ± 1.89
WCTA[30]	81.80 ± 0.82	85.87 ± 0.32	54.83 ± 1.50	64.81 ± 3.99	63.22 ± 2.17	70.11 ± 1.76
KDDG[26]	81.69 ± 0.50	86.22 ± 0.13	55.99 ± 2.61	62.60 ± 4.23	63.49 ± 0.63	70.00 ± 1.62
XDED[27]	82.04 ± 0.56	86.11 ± 0.25	56.10 ± 7.30	66.92 ± 1.93	64.09 ± 0.80	71.05 ± 1.41
Ours	83.83 ± 0.11	86.39 ± 0.04	57.21 ± 3.49	69.84 ± 1.34	65.00 ± 1.55	$\underline{\underline{72.45 \pm 1.31}}$

Table 8.3 Comparison between the proposed methodology and other state-of-the-art DG algorithms on additional target domains. I train the models on all four domains chosen for the previous benchmark. I report IoU (in %) on the unseen domains as mean and standard deviation. The best and second-best results are highlighted and underlined, respectively.

necessary. I apply IBN [24] and ISW[16] to the first two blocks of the backbone, while pAdaIN [25] is applied to all the layers with a probability of 10^{-3} . The ISW loss is weighted by a factor of 10^{-2} , while XDED [27] is applied with a weight of 10^{-3} , a τ of 2, and in combination with UniStyle feature whitening.

Implementation

To tackle a realistic real-time application and following previous work on crop segmentation [10], I choose MobileNetV3 [40] with an LR-ASPP segmentation head [40] as the model architecture. This choice provides an optimal trade-off between performance and efficiency, exploiting effective modules such as depth-wise convolutions, channel-wise attention, and residual skip connections. I train models using ImageNet-pretrained weights, so the input size is fixed to (224, 224). The considered state-of-the-art DG methodologies are taken from the official repositories when available or reimplemented. All the training runs are performed on a single Nvidia RTX 3090 GPU.

Method	Teacher	pWCTA	Logit Std	Vineyard Real	Misc. Real	VegAnn	Average
ERM	\times	\times	\times	51.51 ± 7.27	67.48 ± 0.97	61.92 ± 0.93	60.30 ± 3.06
KD	ERM	\times	\times	61.27 ± 1.03	63.70 ± 1.72	64.99 ± 0.47	63.32 ± 1.07
XDED [27]	Ens.	\times	\times	56.10 ± 7.30	66.92 ± 1.93	64.09 ± 0.80	62.37 ± 3.34
Ours	Ens.	0.001	\times	57.01 ± 2.53	69.07 ± 0.69	65.36 ± 1.12	63.81 ± 1.45
	Ens.	0.001	\checkmark	57.21 ± 3.49	69.84 ± 1.34	65.00 ± 1.55	64.02 ± 2.13

Table 8.4 Ablation study highlighting the contribution of different design choices. I evaluate the effect of KD, domain-expert teachers, pWCTA, and logit standardization. I report IoU (in %) for each result as mean and standard deviation. The best and second-best results are highlighted and underlined, respectively.

8.2.3 Domain Generalization Results

I compare my distillation-based approach with recent and promising DG and semantic segmentation alternatives. Inspired by popular datasets for image classification, I select four domains (*Generic Tree 2*, *Chard*, *Lettuce*, and *Vineyard*) and evaluate all the methodologies by training on three domains and testing on the fourth. The domains are selected to cover various crop dimensions and visual characteristics, ensuring a challenging benchmark for generalization. Then, I perform an additional evaluation by training models on all four datasets and testing on five additional target domains (*Pear Tree*, *Zucchini*, *Real Vineyard*, *Real Miscellaneous*, and *VegAnn*). I run the leave-one-out DG benchmark described in Section 8.2.2 and report the results, along with their mean and standard deviation, in Table 8.2. On average, my ensemble distillation methodology is 1.3% better than the second-best compared solution (XDED), which also distills from a set of specialized teachers. This strategy, therefore, provides students with insightful information and reduces bias towards domain-specific features. The results for ERM are quite balanced across domains, confirming the strong validity of this method despite its simplicity. Other DG methods, even though obtaining superior results in specific domains, are, on average, suboptimal. This failure could be due to the methods focusing on features that are highly beneficial to some particular scenarios but useless to others. My method, on the other hand, consistently delivers good performance in all domains, thanks to the insights provided by the ensembled teachers.

To further validate the generalization capability of my method, I construct a more challenging benchmark by using five unseen test domains (*Pear Tree*,

Zucchini, Real Vineyard, Real Miscellaneous, and VegAnn). The models are trained and validated on all four datasets used for the previous benchmark. In this test, I also investigate the Sim-to-Real gap. The results are reported in Table 8.3, where I also include the teachers' performance as an upper bound. Teacher IoU is lower for real datasets as they are more challenging and contain fewer samples. On average, my ensemble distillation methodology is 1.4% better than the second-best compared solution (XDED), confirming the outcome of the previous benchmark. Moreover, my method retains the best performance on all real domains. This result enforces earlier considerations on the generalization ability of KD without any additional layers or computation at inference time. ERM achieves acceptable results in all domains but is surpassed by a significant margin by other methods, such as IBN and ISW. These results suggest ERM and state-of-the-art DG methods suffer Sim2Real more than ours. Indeed, the change from synthetic to real crops further widens the domain gap between different crops and backgrounds. Another interesting insight can be found in the standard deviations, as my method obtains one of the smallest values. WildNet performs poorly on *Real Vineyard* and *VegAnn* while obtaining satisfactory results on synthetic ones. Its small standard deviation suggests that the multiple training losses applied during training could have an over-regularizing effect on the process. On the contrary, my approach strikes the optimal balance between regularization and learning. In the next section, I examine the contributions of various elements to this result.

8.2.4 Ablation Study

I conduct an ablation study to investigate the effect of different components on the generalization capability of my methodology. I also highlight the main differences between my approach and XDED [27] in terms of methodological components and performance. In particular, I consider the distillation strategy, logit standardization, and pWCTA. I also try substituting the specialized teachers with an ensemble of ERM models trained on all source domains. The results are reported in Table 8.4, where ERM is included as a baseline.

First, the results confirm that applying distillation improves simple ERM without requiring additional computation at inference time. Moreover, plain ensemble distillation cannot bridge the substantial Sym2Real gap. This failure is

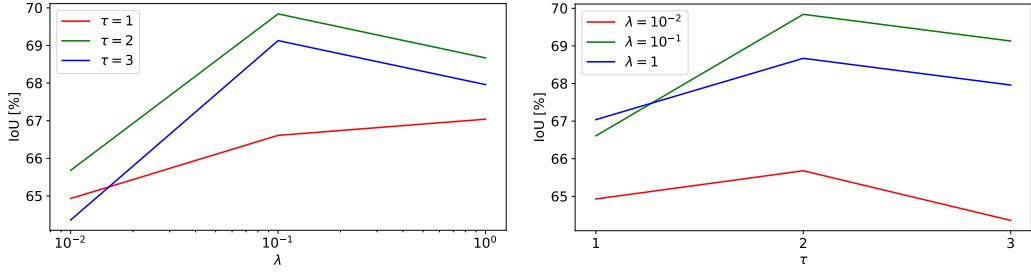


Fig. 8.4 Ablation study on the hyper-parameters λ and τ . The reported IoU value is relative to the *Real Miscellaneous* domain and is averaged on three runs. I represent two views of the results for better readability.

likely due to the unbalanced contributions of different teachers, which can lead to the transfer of domain-dependent biases to the student. To contrast this risk, I standardize distillation logits and apply pWCTA with a low probability to avoid overfitting. My methodology outperforms ERM distillation, leveraging the best from specialized teachers. As depicted in Fig. 8.3, the distillation masks are less confident, providing the student with a better understanding of which parts of the image are more likely to confound the predictor.

I further inspect the effect of the method’s hyperparameters on generalization capabilities. I vary the distillation loss weight λ and the temperature τ and report the results on the *Real Miscellaneous* domain in Fig. 8.4. The graphs show that my choice ($\lambda = 10^{-1}$, $\tau = 2$) is the optimal balance that ensures regularization without constraining the student. As reported in my benchmarks, this yields good generalization across various synthetic and real domains.

8.3 Conclusion

In this work, I proposed a novel method to tackle the problem of DG for crop semantic segmentation in realistic scenarios. I demonstrated that my distillation method represents a competitive approach for transferring domain-specific knowledge learned from multiple teacher models to a single student, with no overhead at inference time. Moreover, I proposed logit standardization to adapt ensembled knowledge to the student, balancing overconfident predictions and penalizing spurious correlations. Each teacher must be trained only once, and the method can be extended to more domains by just training a new teacher and then

distilling. I conceived a solution to enhance the robustness and generalization properties of segmentation models for unseen environmental conditions or crops. Extensive experimentation has been conducted on the novel multi-crop synthetic dataset AGRISEG and on real test data to demonstrate the overall generalization boost provided by my training method. Moreover, I conducted an ablation study to highlight the role of different components in my solution. The superior results provided by my method show how pairing ensembled KD and DG can lead to robust perception models for realistic tasks in precision agriculture. Future works will progressively add more domains and DG methods to the AGRISEG benchmark. I will also include more real-world labeled data to guarantee a deeper investigation of the use of synthetic data for robust generalization in agriculture.

After addressing domain generalization in semantic segmentation through ensemble distillation for agricultural robotics, the next chapter broadens the scope to perception from overhead imagery. Here, I present a multi-head model designed for onboard satellite image processing, trained using self-supervised learning and optimized for fault tolerance. This work extends the pursuit of robust and generalizable AI by enabling efficient, resilient perception directly at the edge, which is essential for autonomous systems operating in dynamic and resource-constrained environments, such as remote sensing platforms.

Bibliography

- [1] S. Angarano, M. Martini, A. Navone, and M. Chiaberge, “Domain generalization for crop segmentation with standardized ensemble knowledge distillation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5450–5459.
- [2] Z. Zhai, J. F. Martínez, V. Beltran, and N. L. Martínez, “Decision support systems for agriculture 4.0: Survey and challenges,” *Computers and Electronics in Agriculture*, vol. 170, p. 105 256, 2020.
- [3] C. Ren, D.-K. Kim, and D. Jeong, “A survey of deep learning in agriculture: Techniques and their applications,” *Journal of Information Processing Systems*, vol. 16, no. 5, pp. 1015–1033, 2020.
- [4] C. W. Bac, E. J. van Henten, J. Hemming, and Y. Edan, “Harvesting robots for high-value crops: State-of-the-art review and challenges ahead,” *Journal of Field Robotics*, vol. 31, no. 6, pp. 888–911, 2014.
- [5] D. Deshmukh, D. K. Pratihar, A. K. Deb, H. Ray, and N. Bhattacharyya, “Design and development of intelligent pesticide spraying system for agricultural robot,” in *Hybrid Intelligent Systems: 20th International Conference on Hybrid Intelligent Systems (HIS 2020), December 14-16, 2020*, Springer, 2021, pp. 157–170.
- [6] A. Feng, J. Zhou, E. D. Vories, K. A. Sudduth, and M. Zhang, “Yield estimation in cotton using uav-based multi-sensor imagery,” *Biosystems Engineering*, vol. 193, pp. 101–114, 2020.
- [7] Z. Luo, W. Yang, Y. Yuan, R. Gou, and X. Li, “Semantic segmentation of agricultural images: A survey,” *Information Processing in Agriculture*, 2023.
- [8] S. Mukhopadhyay, M. Paul, R. Pal, and D. De, “Tea leaf disease detection using multi-objective image segmentation,” *Multimedia Tools and Applications*, vol. 80, pp. 753–771, 2021.
- [9] H. Peng *et al.*, “Semantic segmentation of litchi branches using deeplabv3+ model,” *IEEE Access*, vol. 8, pp. 164 546–164 555, 2020.
- [10] D. Aggi, S. Cerrato, V. Mazzia, and M. Chiaberge, “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.

- [11] E. Raei, A. A. Asanjan, M. R. Nikoo, M. Sadegh, S. Pourshahabi, and J. F. Adamowski, “A deep learning image segmentation model for agricultural irrigation system classification,” *Computers and Electronics in Agriculture*, vol. 198, p. 106 977, 2022.
- [12] F. Salvetti, S. Angarano, M. Martini, S. Cerrato, and M. Chiaberge, “Waypoint generation in row-based crops with deep learning and contrastive clustering,” in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part VI*, Springer, 2023, pp. 203–218.
- [13] M. Martini, S. Cerrato, F. Salvetti, S. Angarano, and M. Chiaberge, “Position-agnostic autonomous navigation in vineyards with deep reinforcement learning,” in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2022, pp. 477–484.
- [14] G. Csurka, *Domain adaptation in computer vision applications*. Springer, 2017.
- [15] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy, “Domain generalization: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [16] S. Choi, S. Jung, H. Yun, J. T. Kim, S. Kim, and J. Choo, “Robustnet: Improving domain generalization in urban-scene segmentation via instance selective whitening,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 11 580–11 590.
- [17] S. Lee, H. Seong, S. Lee, and E. Kim, “Wildnet: Learning domain generalized semantic segmentation from the wild,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 9936–9946.
- [18] M. Martini, V. Mazzia, A. Khaliq, and M. Chiaberge, “Domain-adversarial training of self-attention-based networks for land cover classification using multi-temporal sentinel-2 satellite imagery,” *Remote Sensing*, vol. 13, no. 13, p. 2564, 2021.
- [19] N. Ye *et al.*, “Ood-bench: Quantifying and understanding two dimensions of out-of-distribution generalization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 7947–7958.
- [20] R. Barth, J. IJsselmuider, J. Hemming, and E. J. Van Henten, “Data synthesis methods for semantic segmentation in agriculture: A capsicum annum dataset,” *Computers and electronics in agriculture*, vol. 144, pp. 284–296, 2018.
- [21] I. Gulrajani and D. Lopez-Paz, “In search of lost domain generalization,” in *International Conference on Learning Representations*, Computer Vision Foundation, 2021.
- [22] S. Angarano, M. Martini, F. Salvetti, V. Mazzia, and M. Chiaberge, “Back-to-bones: Rediscovering the role of backbones in domain generalization,” *Pattern Recognition*, vol. 156, p. 110 762, 2024.
- [23] A. Kirillov *et al.*, “Segment anything,” *arxiv: 2304.02643*, 2023.
- [24] X. Pan, P. Luo, J. Shi, and X. Tang, “Two at once: Enhancing learning and generalization capacities via ibn-net,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 464–479.

- [25] O. Nuriel, S. Benaim, and L. Wolf, “Permuted adain: Reducing the bias towards global statistics in image classification,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9482–9491.
- [26] Y. Wang, H. Li, L.-p. Chau, and A. C. Kot, “Embracing the dark knowledge: Domain generalization using regularized knowledge distillation,” in *Proceedings of the 29th ACM International Conference on Multimedia*, 2021, pp. 2595–2604.
- [27] K. Lee, S. Kim, and S. Kwak, “Cross-domain ensemble distillation for domain generalization,” in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXV*, Springer, 2022, pp. 1–20.
- [28] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*, 2012.
- [29] M. 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.
- [30] J. Weyler, T. Läbe, F. Magistri, J. Behley, and C. Stachniss, “Towards domain generalization in crop and weed segmentation for precision farming robots,” *IEEE robotics and automation letters*, vol. 8, no. 6, pp. 3310–3317, 2023.
- [31] S. K. Panda, Y. Lee, and M. K. Jawed, “Agronav: Autonomous navigation framework for agricultural robots and vehicles using semantic segmentation and semantic line detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2023, pp. 6271–6280.
- [32] R. Geirhos *et al.*, “Shortcut learning in deep neural networks,” *Nature Machine Intelligence*, vol. 2, no. 11, pp. 665–673, 2020.
- [33] Z. Allen-Zhu and Y. Li, “Towards understanding ensemble, knowledge distillation and self-distillation in deep learning,” in *The Eleventh International Conference on Learning Representations*, 2022.
- [34] S. Sun, W. Ren, J. Li, R. Wang, and X. Cao, “Logit standardization in knowledge distillation,” *arxiv: 2403.01427*, 2024.
- [35] S. Madec *et al.*, “Vegann, vegetation annotation of multi-crop rgb images acquired under diverse conditions for segmentation,” *Scientific Data*, vol. 10, no. 1, p. 302, 2023.
- [36] V. N. Vapnik, “An overview of statistical learning theory,” *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 988–999, 1999.
- [37] C. Shu, Y. Liu, J. Gao, Z. Yan, and C. Shen, “Channel-wise knowledge distillation for dense prediction,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5311–5320.
- [38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

- [39] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2019.
- [40] A. Howard *et al.*, “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

Chapter 9

A Multi-service Edge AI Architecture based on Self-supervised Learning

Original Paper: *E. Magli, S. Angarano, S. Bassetti, T. Bianchi, P. Boccardo, S. Bucci, M. Chiaberge, G. Inzerillo, D. Lisi, G. Mascetti, M. Mergè, C. Monaco, M. Pasturensi, D. Piccinini, D. Valsesia, G. Zema, “A Multi-service Edge-AI Architecture based on Self-supervised Learning,” in Proc. of 75th International Astronautical Congress, IAF, 2024.* [1]

Conventional satellite imaging systems typically capture space data and transmit it to ground stations for subsequent processing and analysis. This workflow often delays significantly the availability of the processed image to the final user, sometimes extending it by several days. Such delays are especially problematic in emergency scenarios, such as natural disasters, where immediate access to data is critical. To mitigate this issue, a new approach has gained traction: shifting some of the image processing workload onboard the satellite itself. By processing data in real-time, satellites can detect critical events early and send prioritized alerts, ensuring that essential information reaches ground stations more quickly.

This emerging trend, known as edge AI, leverages recent advancements in artificial intelligence, particularly deep learning, to enable advanced image analysis directly in orbit. Edge AI enables satellites to process imagery immediately after capture, filtering out irrelevant data, such as cloud-covered images, and detecting high-priority events, like floods or fires, with minimal delay. However, the adoption of edge AI has been hindered by several challenges, including the difficulty of deploying machine learning models on field-programmable gate arrays

(FPGAs), high power consumption, and the limited availability of labelled data for training purposes. This paper aims to establish a framework for developing an edge AI system that manages multiple onboard tasks while addressing existing technological barriers. The proposed system centers on a deep neural network for satellites equipped with multispectral sensors. The central component of this system is a feature extractor, which generates semantic representations of the captured multispectral imagery that various onboard applications can utilize. This extractor is trained using a self-supervised learning (SSL) technique, enabling it to generate general-purpose features without requiring large quantities of labelled data. Individual application-specific heads are developed for specific tasks and utilize these shared features, minimizing both computational demands and the need for extensive annotated datasets.

Additionally, multitasking can enhance overall system efficiency; for example, cloud detection can optimize data transmission by excluding regions with dense clouds, enabling adjustments to compression algorithms for improved data management and faster data transmission. While deep learning significantly enhances detection capabilities in such applications, designing an AI system for onboard real-time multitasking requires careful consideration of satellite platform constraints, including limited power and processing resources. This paper proposes developing such an architecture, emphasizing a lightweight neural network architecture tailored for feature extraction from multispectral images with varying spatial resolutions. The feature extractor is trained in a self-supervised manner using large volumes of unlabelled data. These extracted features are available to third-party developers, who can design task-specific application heads, such as image segmentation or classification, each running independently of the backbone architecture. That ensures multiple applications can operate concurrently without overwhelming the onboard computational resources. Furthermore, the modular system design supports in-flight updates and the gradual integration of new tasks, improving the satellite's adaptability and long-term efficiency.

Another crucial aspect for embedded systems that operate outside the atmosphere is the effect of ionizing radiation. Digital circuits operating in the presence of radiation can undergo permanent or transient disturbances: the former appear only after prolonged radiation exposure and are caused by the trapping of charges inside the oxide, the latter can instead appear due to the impact of a single particle with an electric charge against a sensitive area of the

integrated circuit. However, these events remain relatively rare, although the progressive miniaturization of computational systems has dramatically increased their probability [2]. Radiation can cause errors of two types in the functioning of a machine learning model [3]. The first consists of variations in the value of the neural network weights, stored in RAM. That causes a permanent and systematic error in network predictions, which can only be resolved by updating the weights with those stored in ROM, thereby making the system safer. The second problem concerns network activations, i.e., the intermediate results of operations within the model. These problems are transient, as the activations are overwritten at the next iteration of the network, and therefore have less impact in the long run, considering that the probability of an event is relatively low. Despite this, it may be interesting to detect these errors (Fault Detection) and possibly discard the prediction concerned, or repeat it. In addition, several methodologies have been proposed to make neural networks more resilient to these phenomena, either during (Fault-aware Training) or after training (Post-training Fault Mitigation) [4].

This paper [1] presents an implementation of the whole architecture, comprising the feature extractor and three specific application heads, with results demonstrating its effectiveness in three different tasks (cloud segmentation, flood segmentation, and fire segmentation), highlighting the potential of this architecture for onboard satellite image processing. To perform these tasks, I selected five spectral bands for training: red, green, blue, near-infrared (NIR), and short-wave infrared (SWIR). They offer a strong balance between informational richness and computational efficiency. These bands provide sufficient spectral diversity to capture key features for my tasks while maintaining efficiency, as I only use five channels rather than the dozens or hundreds typically found in hyperspectral imagery. I also analyze a selection of simple methodologies for detecting and rejecting errors caused by radiation.

9.1 Dataset

To train my AI architecture, I utilized four different datasets, all containing multi-spectral images captured by the European Space Agency's Sentinel-2 mission:

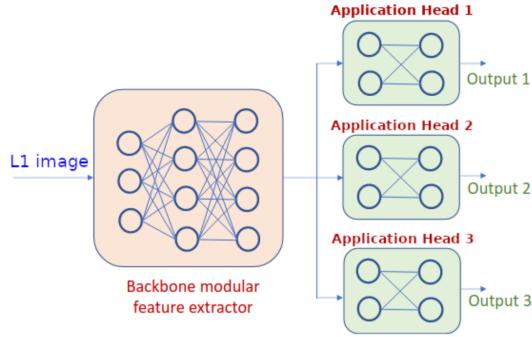


Fig. 9.1 The backbone feature extractor creates meaningful feature representations shared with the three application heads.

- A subset of approximately 40,000 images from the majorTom dataset [5] was used for self-supervised learning to train the backbone feature extractor.
- The CloudSen12 dataset [6] was employed for supervised training of the application head dedicated to cloud segmentation. This dataset contains around 49,000 images with corresponding segmentation masks indicating the presence of cloudy or clear pixels (distinguishing between *thin cloud*, *thick cloud*, *cloud shadow*, and *background*) and was explicitly created as an extensive benchmark for cloud cover segmentation tasks.
- I specifically created the training datasets for flood and fire segmentation heads for these two tasks. For the flood segmentation dataset, I created segmentation masks with four classes: *no event*, *flood*, *flood trace*, and *permanent water*. Similarly, for the fire segmentation dataset, I generated masks with three classes: *no event*, *burnt area*, and *active flame*. To generate the datasets, shapefiles obtained from the Rapid Mapping activations of the Copernicus Emergency Management Service were also used; these shapefiles were refined automatically and manually to meet my needs¹.

9.2 Methodology

The overall architecture of my AI system is illustrated in Fig. 9.1, consisting of two primary components: a backbone feature extractor and multiple application

¹[Copernicus Emergency Management Service](#)

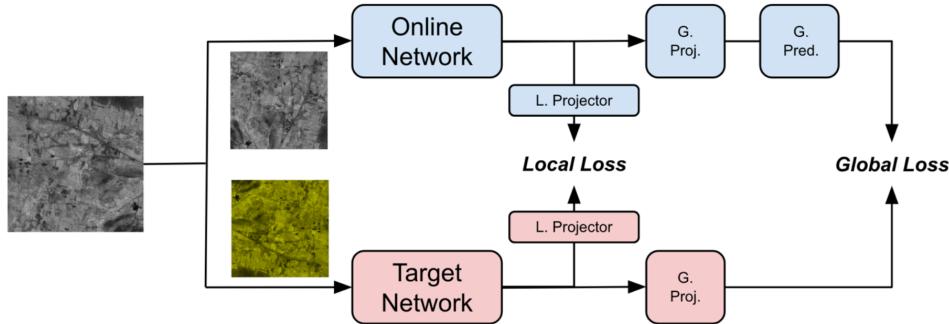


Fig. 9.2 SSL training of the backbone feature extractor using both global and local contrastive loss. The input images are obtained by applying two different augmentation sets to the same source image: spatial transformations (upper branch) and spatial and color transformations (lower branch).

heads. This modular design enables the use of various independent application heads for different tasks, with the flexibility to add more as needed. The backbone serves as a universal feature extractor, trained using a self-supervised learning approach and developed independently of specific application heads. Its primary requirement is to generate broadly applicable features across various tasks. Once extracted, these features are passed to task-specific heads, smaller neural networks tailored to individual tasks, and then fine-tuned on specific datasets for the task they are designed to solve. To maintain the reusability of the extracted features for all tasks, I freeze backbone weights during fine-tuning.

The input to the architecture, as shown in Fig. 9.1, is a raw Sentinel-2 Level-1C image received directly from the multispectral sensor. These images have undergone radiometric and geometric corrections, and an orthorectification process has been applied to them. It is important to note that once the backbone has extracted the features, they are simultaneously passed to the application heads, which process these representations in parallel. That allows the system to perform n tasks ($n = 3$ for my specific demonstration) concurrently, effectively reducing latency.

Moreover, I applied post-training quantization to meet the stringent requirements of low latency and computational efficiency for effective onboard processing, thereby reducing the model's weight precision from 32-bit floating-point (FP) to 8-bit integer (INT). This approach minimizes the model's complexity, reducing memory occupation with only a minimal loss in accuracy, and optimizes it for deployment on resource-constrained hardware. The shift to INT also allows

for faster inference times, making the model suitable for real-time operations in space environments. In the following subsections, I will go into details and explain each module of the architecture.

9.2.1 Feature Extractor

As mentioned in Section 2, the backbone feature extractor aims to generate feature representations that must be shared among all application heads. Sharing the feature extractor across multiple information extraction applications is a highly effective concept for enabling the execution of several tasks within the same architecture. The most computationally intensive part is performed only once, significantly improving efficiency. To this aim, the feature extractor must be trained independently of any specific application, as it needs to be adaptable to a wide range of tasks. For this purpose, I trained the feature extractor using self-supervised learning techniques, specifically contrastive learning.

In contrast to conventional supervised training, self-supervised techniques do not rely on explicit annotations and labels from the training dataset. To perform the self-supervised training of my feature extractor, I employed the contrastive learning method BYOL [7]. Contrastive learning is a self-supervised approach that aims to learn useful representations by contrasting positive pairs (i.e., similar data points, such as different views of the same image) against negative pairs (i.e., dissimilar data points). Typically, these architectures utilize two neural networks, an online network and a target network, to align their output representations effectively. The architecture of my backbone and the training process using the SSL framework are illustrated in Fig. 9.2. As depicted in the figure, the input to the SSL architecture consists of two distinct augmentations of the same base image. These augmentations play a crucial role, as they introduce variations of the same input that both the online and target networks process. Despite receiving different augmented versions, the networks are trained to produce similar feature representations.

Specifically, for the online network (upper branch), the input image is subjected solely to spatial augmentations (e.g., random crop, horizontal and vertical flip). In contrast, the target network (lower branch) receives the same source image augmented with both spatial and color transformations (e.g., solarization,

color jittering). Then, to learn informative feature representations of the input images, a global contrastive loss is computed by aligning the feature representations output by the global projector and global predictor of the online network with those from the target network. The goal is to minimize the difference between the online network’s predicted output and the target network’s projected output. It is essential to highlight that most state-of-the-art SSL methods, including BYOL, are primarily designed for image-level tasks, such as classification. However, in this work, I aim to pre-train a model that can effectively support a variety of application heads, including those responsible for pixel-level tasks like semantic segmentation and object detection. To achieve this, I introduced a local contrastive loss, inspired by [8], into my SSL training. This loss is designed to better capture and model dense features, which are crucial in solving fine-grained tasks. The overall SSL loss is computed as:

$$L = \lambda L_{LC} + (1 - \lambda) L_{GC} \quad (9.1)$$

where λ is a trade-off parameter to balance between local contrastive loss (L_{LC}) and the classical global contrastive loss (L_{GC}). The local contrastive loss is essentially a Negative Log-Likelihood (NLL) to capture fine-grained, pixel-level correspondences between two differently augmented views of the same input image. To do so, I first define a set of key points in an augmented image and then map these points to their corresponding locations in the second image based on the known spatial transformations. These key points in the first image p_c are matched with corresponding points in the second image p_{sc} . For each key point in the first image, the model computes the similarity between its feature representation and the feature representation of the corresponding point in the second image. That is achieved using a dense correspondence map, which calculates the similarity score between points. Lastly, the NLL quantifies the likelihood that the corresponding points (p_c, p_{sc}) match due to feature similarity.

Specifically, it measures how well the feature representation of a point in the first image aligns with the feature representation of the corresponding point in the second image, compared to other possible points in the second image. On the other hand, the global contrastive loss utilizes cosine similarity to assess the degree of alignment between the global representations of the two augmented

views. I apply cosine similarity to the feature tensors from the online and target networks.

Regarding the architecture of the backbone feature extractor, given that the objective is to develop a model capable of running within the constraints of a satellite payload, I opted for a pre-existing SOTA efficient and low-complexity architecture [9]. Nevertheless, it is worth noting that the choice of this model as the backbone is not restrictive, as the SSL training procedure and the entire framework would remain unchanged with any other neural network capable of extracting image features.

9.2.2 Application Heads

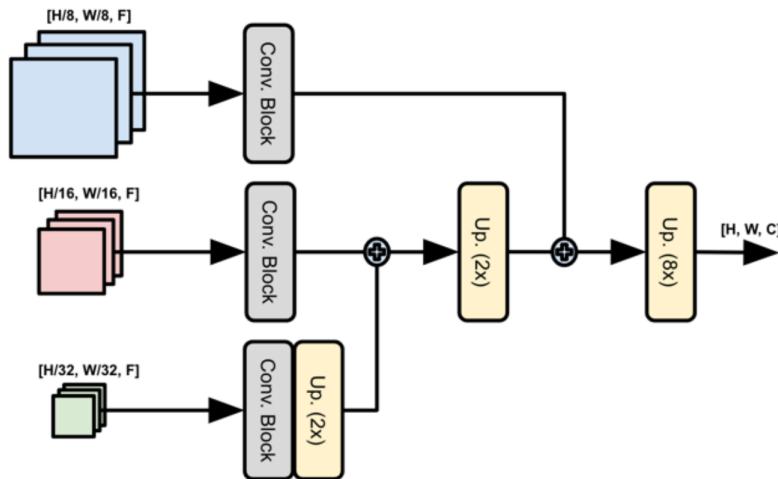


Fig. 9.3 Architecture of the segmentation head, which takes in input multi-resolution features. The shape of the different features is marked in square brackets. F identifies the number of channels computed by the feature extractor, C the number of output classes in the segmentation map, and H and W are the height and width of the image/features, respectively.

I tested the entire model on the three tasks described in Section 9.2.1 to evaluate my AI system. Consequently, I required three parallel, task-specific heads. Since all tasks involve segmentation (cloud, fire, and flood), I designed a single-head architecture. As shown in Fig. 9.3, the segmentation head is a neural network that takes three different feature maps at various resolutions from the feature extractor and aggregates them before performing a set of upsampling and lightweight convolution to obtain the segmentation map.

Each head, tailored to specific tasks, only requires fine-tuning on datasets pertinent to its particular function. When introducing a new task, the process involves simply creating and fine-tuning an additional application head. Importantly, training the entire architecture is avoided to maintain its modularity and generality. Instead, I selectively fine-tune only the application heads while keeping the backbone's weights frozen.

9.2.3 Fault Robustness

I have addressed the two problems separately, starting with weights, as they have a greater impact on performance and are easier to identify.

Permanent Errors on Weights

The most straightforward approach is to update the parameters of the networks in the memory used for calculations at regular intervals with those in the non-volatile memory. This approach, however, requires heavy access to both the ROM and RAM of the system, and therefore, the update cannot be done too often. Additionally, updates are applied even when they are unnecessary.

Therefore, I studied an alternative blind method based on observing the network's outputs rather than its parameters. This method uses a test input and the corresponding known output to detect the presence of corruption in the weights. Only if a discrepancy is detected are the weights reloaded from the ROM. This second method is more efficient as it puts less strain on the system at the computational level. Making an inference on a test input is less expensive than reloading all the weights into memory. At the same time, some errors may not be detected solely by checking the output. To make it more robust, one could also check the intermediate outputs of the layers or use multiple test inputs, which would increase resource expenditure.

This system further reduces memory access and eliminates the need for periodic test input usage. Additionally, its simplicity allows for no overhead in system predictions. More sophisticated and precise detection methods may be studied in the future, considering the need for higher computational costs.

Transient Errors on Activations

Because errors on activations are transient, the solution must be in place during regular system operation without periodic checks or test inputs. At the same time, the chosen approach should not introduce significant computational overhead compared to the system's nominal operation, thereby avoiding slowdowns. Finally, I want to avoid introducing fault-aware network training methods [10] for two reasons:

- The robustness that can be obtained architecturally with special training is never perfect or guaranteed.
- Networks developed in this way tend to pay for the acquired robustness with a deterioration in performance.

I therefore focus on post-training methods. Two approaches can be implemented:

- **Fault Mitigation**, which consists of changing the already trained network to absorb or reduce the effect of errors caused by radiation.
- **Fault Detection**, which consists of detecting and discarding measurements affected by errors.

For the former, I have replaced the ReLU activation function, used during training, with the more robust ReLU6 [11]. That allows for saturating values that are too large during intermediate computations, thereby limiting the effect of possible anomalies. The advantage of this approach is that there is not always the need to repeat radiation-affected predictions because some errors are automatically “absorbed” by the network. On the other hand, errors that cannot be absorbed must be detected, and the corresponding predictions discarded. This system further reduces memory access and eliminates the need for periodic test input usage. Additionally, its simplicity allows for no overhead in system predictions. In the next section, I report the results for different values of the detection range, which has also been tested in combination with the fault mitigation method. More sophisticated and precise detection methods may be studied in the future, considering the need for higher computational costs.

9.3 Results

I conducted a series of extensive experiments with the proposed architecture. The tests were performed by first training the backbone using the SSL approach explained in Section 9.2.1. Then, the three segmentation heads for cloud, fire, and flood segmentation were trained, maintaining the feature extractor’s weights frozen (i.e., without updating them), in a classical supervised manner, using the labels of the three datasets. Table 9.1 summarizes the computational complexity of each component of the proposed architecture, computed by forwarding an input image of shape (512, 512, 5). It is easily noticed that the bottleneck of my architecture is found, as expected, in the backbone feature extractor, where the computational complexity is most concentrated. At the same time, the segmentation head design is significantly more efficient, with a small number of floating-point operations (FLOPs) and parameters.

To evaluate the performance of my model across cloud, fire, and flood segmentation, I adopted the mean intersection over union (mIoU) metric. It is calculated by determining the intersection over union (IoU) for each class in the segmentation task. The IoU measures the overlap between the predicted segmentation and the ground truth for each class, where the intersection is the common area between the predicted and ground truth masks, and the union is the total area covered by both the predicted and ground truth masks combined. The mIoU extends this concept by averaging the IoU values across all the classes, providing a single performance measure. Mathematically, for N classes, the mIoU is expressed as:

$$mIoU = \frac{1}{N} \sum_{i=1}^N \omega_i \left| \frac{P_i \cap G_i}{P_i \cup G_i} \right| \quad (9.2)$$

where P_i is the predicted set of classes for class i , G_i is the ground truth set of pixels for class i , and ω_i is the weight assigned to class i , based on the proportion of pixels in the ground truth for that class.

The mIoU metric is particularly advantageous in my case since it balances the contributions of all classes in the segmentation task, ensuring that performance is not disproportionately affected by classes that occupy larger areas (e.g., background), furthermore, using the weight factor ω ensures that larger classes (in terms of

	FLOPs	Params
Feature Extractor	4.50×10^9	9.55×10^6
Segmentation Head	1.87×10^8	4.02×10^5

Table 9.1 Computational complexity of each component of the proposed Architecture.

pixels) have a proportionally higher weight in the final mIoU calculation. Using mIoU as my evaluation metric, I can comprehensively assess how well my model performs across different segmentation tasks and ensure a fair comparison of its efficacy on cloud, fire, and flood segmentation tasks. This metric also facilitates straightforward interpretation: a value closer to 1 indicates a higher degree of overlap between the predicted and true segmentation, whereas a value closer to 0 indicates poor overlap, making it a clear indicator of model performance.

In the following subsections, I discuss the evaluation settings and results for each task. As a term of comparison, I evaluate my architecture under two configurations:

- SSL: the backbone feature extractor is pretrained using the aforementioned SSL approach, after which its weights are frozen. The three task-specific heads are then fine-tuned independently for each task. This configuration allows for parallel inference across all three heads.
- SL: the backbone feature extractor and a single segmentation head are treated as one single network and trained using a classical supervised learning (SL) approach. In this setup, parallel inference is impossible; only one task can be addressed at a time. However, the backbone is fine-tuned specifically for that task, producing task-specific rather than general features shared across multiple tasks.

9.3.1 Cloud Segmentation

To evaluate my architecture on the cloud segmentation task, I utilized only the high-quality images (i.e., images with a corresponding high-quality segmentation mask as ground truth) from the CloudSen12 dataset [6], specifically those manually labelled. I excluded the low-quality images, as they are either automatically annotated or contain labelling inaccuracies, which could introduce noise and

negatively impact the model’s training and evaluation process. Focusing on the high-quality subset ensured more reliable ground-truth data, allowing for a more accurate assessment of the model’s segmentation performance.

Moreover, since there is essentially no difference between the *thin cloud* and *thick cloud* classes, I aggregated the two classes into one, resulting in a 3-class dataset (*cloud*, *cloud shadow*, and *background*). Table 9.2 presents quantitative performance comparisons for the cloud segmentation task using both the SSL and SL configurations. As anticipated, the SL architecture exhibits superior performance since the backbone feature extractor and the segmentation head are specifically trained and fine-tuned for the cloud segmentation task. However, this highest performance comes at a cost: it precludes the ability to perform other tasks in parallel, strongly increasing latency in the multitask approach. In Fig. 9.4 (first column) are reported the confusion matrices computed for the cloud segmentation task.

For cloud segmentation, the SSL model performs reasonably well in detecting the background (47.48%), but shows notable confusion between clouds (6.5%) and cloud shadows (2.95%). Cloud shadow detection in the SSL model is relatively weak, with significant overlap with the cloud and background classes. In contrast, the SL model performs better, with a background detection rate of 49.01%. Furthermore, cloud shadows are more readily recognized (6.72%) with reduced confusion. The better performance of the SL model in this task can be attributed to the direct supervision provided during training, which likely allows the model to learn more precise boundary definitions between clouds, shadows, and background, minimizing ambiguity.

Configuration	mIoU
SSL	70.65
SL	82.00

Table 9.2 Cloud Segmentation Performance Comparison.

9.3.2 Flood Segmentation

The dataset used for flood segmentation, as detailed in Section 9.1, initially included four distinct classes: *no event*, *flood*, *flood trace*, and *permanent wa-*

ter. However, differentiating between flood water and permanent water using multispectral imagery proved extremely challenging, if not nearly impossible. Consequently, I opted to aggregate these two classes into a single *water* class to simplify the segmentation task and improve the model’s performance. Globally, the class distribution of pixels after the merge of the two water classes is as follows:

- No event: 75.12%.
- Water: 21.45%.
- Flood trace: 3.43%.

I used 14,805 images for the training process of the application heads and 4,643 images for testing. Table 9.3 presents quantitative performance comparisons for the flood segmentation task using both the SSL and SL configurations. Fig. 9.4 (second column) reports the confusion matrices computed for the flood segmentation task.

Configuration	mIoU
SSL	91.83
SL	92.96

Table 9.3 Flood Segmentation Performance Comparison.

In the flood segmentation task, the SSL model demonstrates strong performance in detecting the *no event* class (74.69%), but struggles with water detection, achieving 20.72% and exhibiting significant confusion with the *no event* class (0.64%). Flood trace detection is notably weak at 2.53%, indicating that the model struggles to distinguish flood traces from other classes. The SL model offers a slight improvement, maintaining high accuracy for the *no event* class (74.53%) and increasing water detection to 21.06%. Flood trace detection, however, remains low at 2.09%. The marginal improvement in the SL model’s performance may be attributed to the availability of labeled data, which enables the model to differentiate between subtle features of water and flood traces. However, the intrinsic complexity of flood patterns still poses a challenge.

9.3.3 Fire Segmentation

For the fire segmentation task, I utilized the dataset in its original form, without any class aggregation or modifications. However, it is crucial to highlight the significant class imbalance present in this dataset: *no event* covers 80.53% of the pixels, *burnt area* 19.43%, and *active flames* 0.04%. This imbalance arises because capturing remotely sensed images directly over areas with active flames is extremely rare. Additionally, due to their intrinsic nature, accurately recognizing and labeling active flames is challenging, leading to their minimal representation in Sentinel-2 imagery.

I used 14,876 images for the training process and 4,233 images for testing. Table 9.4 presents quantitative performance comparisons for the cloud segmentation task using both the SSL and SL configurations. The confusion matrices computed for the fire segmentation task are reported in Fig. 9.4 (third column). As shown from the matrices, the SSL model performs well in classifying the *no event* class (76.2%) but shows confusion with the *burnt area* class (4.32%). Burnt area detection reaches 13.44%, though much of it is misclassified as *no event*.

The model is almost entirely ineffective at detecting active flames, with an accuracy of just 0.01%. The SL model slightly improves performance, with a *no event* detection rate of 76.89% and a *burnt area* classification accuracy of 17.53%. However, active flame detection remains extremely poor (0.01%). The difficulty both models face in distinguishing burned areas and active flames stems from the visual similarities between these features and the background, as well as the challenge of capturing small regions of dynamic flames, which are extremely rare in the dataset, as discussed at the beginning of this subsection. The scarcity of features related to active flames is readily apparent, as only 0.04% of pixels in the entire dataset contain active flames.

It is also plausible that during the SSL training of the backbone feature extractor, very few images (if any) contain active flames, making it challenging to learn semantically representative features of the phenomenon, which, even in the case of SL configuration, is extremely difficult to detect correctly.

	Background	Clouds	Cl. Shadows	No Event	Water	Flood Trace	No Event	Burnt Area	Active Flames
Background	47.48	3.96	1.1	74.69	0.35	0.08	76.2	4.32	0.01
Clouds	6.5	30.69	0.76	0.64	20.72	0.09	5.98	13.44	0.01
Cl. Shadows	2.95	1.96	4.61	2.53	0.58	0.32	0.02	0.01	0.02
	Background	Clouds	Cl. Shadows	No Event	Water	Flood Trace	No Event	Burnt Area	Active Flames
Background	49.01	2.49	1.04	74.53	0.44	0.15	76.89	3.62	0.01
Clouds	2.84	34.23	0.88	0.28	21.06	0.1	1.88	17.53	0.01
Cl. Shadows	1.47	1.32	6.72	2.09	0.64	0.71	0.01	0.01	0.03

Fig. 9.4 Confusion matrices of segmentation tasks reporting the absolute percentage value of true and predicted classes. Columns from left to right indicate cloud segmentation, flood segmentation, and fire segmentation. Rows indicate SSL configuration and SL training.

Configuration	mIoU
SSL	81.93
SL	89.91

Table 9.4 Fire Segmentation Performance Comparison.

9.3.4 Discussion

Examining the results presented in the previous section, I observe that, in general, the SL model outperforms the SSL model across all tasks, particularly in detecting clouds and burned areas. As I might expect, the primary reason for this improvement is the presence of labelled data in the SL configuration, which enables the model to learn more apparent class distinctions during the end-to-end training of the whole architecture. In contrast, the SSL model, lacking explicit supervision (except for the fine-tuning of task-specific heads), struggles with overlapping or visually ambiguous features, resulting in increased confusion between classes. However, neither model performs optimally in detecting smaller or more complex classes, such as flood traces or active flames. That suggests both configurations could benefit from more advanced techniques or, even better, by training on datasets that better represent these features.

Configuration	mIoU	Latency [ms]
SSL	81.47	114.41
SL	88.29	37.47

Table 9.5 Average quality metric and latency comparison in performing the three tasks.

However, it is worth noting that, even though the SSL configuration performs slightly worse than the SL in segmentation accuracy, it successfully addresses my first objective: performing all three tasks with overall high accuracy while reducing the computational complexity. The SSL model enables parallel inference, allowing for the simultaneous resolution of all three tasks (clouds, floods, and fire segmentation). This parallel approach offers a significant advantage in terms of efficiency, as it minimizes the need for sequential processing.

In Table 9.5, I provide an analysis of the latencies for computing the three tasks using both configurations (quantized, INT8 precision) on a 7W low-power edge device with an accelerator specifically tailored for DL models, highlighting how the SSL setup offers reduced latency. Additionally, I present a comprehensive comparison of the energy-latency-quality trade-off, demonstrating that while the SL configuration slightly edges out in quality, the SSL model excels in energy efficiency and latency. The SSL configuration maintains a stable latency even when adding a finite number of task-specific heads. That is because the tasks are executed in parallel, allowing the system to handle multiple tasks simultaneously without significantly increasing processing time. On the other hand, in the SL configuration, latency increases linearly with the addition of task-specific heads. Each time a new head is added, it must be executed sequentially, leading to two main drawbacks: increased latency due to the serial execution of tasks and higher memory consumption on board, as each task requires its own model instance (backbone and head). That creates inefficiencies, particularly in systems with limited computational resources, where managing multiple models simultaneously can lead to performance bottlenecks. Therefore, while the SL model might offer slight improvements in accuracy, the SSL configuration proves more scalable and efficient, especially when dealing with multiple tasks in real-time applications.

9.3.5 Fault Robustness

A framework for simulating errors generated by radiation at the software level was created to perform all the reported experiments. The developed framework is primarily based on the studies presented in two existing articles [12, 10].

The problem of transient errors on intermediate activations was initially analyzed by testing the performance of the trained model for different failure probabilities. In particular, the accuracy of the cloud segmentation task on the CloudSEN12 dataset was tested [6]. The error probability p used as a parameter corresponds to the likelihood that an error will perturb a batch sample. However, errors are randomly inserted at each convolutional, linear, or batch normalization layer; therefore, the actual probability is much higher. I report the percentage of samples corrupted in the rightmost column of Table 9.6. In addition, I report loss, accuracy, balanced accuracy, and mean intersection over union (mIoU). The first interesting result is that for low probabilities ($< 1 \times 10^{-3}$), the effect of errors is almost negligible, whereas for high values ($> 1 \times 10^{-2}$), it is disastrous, preventing the network from functioning correctly. As a result, I have focused on the middle probability band that may have room for improvement. A second aspect to underline is that some errors injected into the network are intrinsically absorbed and do not impact performance. Note the small amount of accuracy loss compared to the size of the loss or the percentage of corrupt samples.

p	Loss	Accuracy	Balanced Accuracy	mIoU	Faulty Samples
0	0.38	85%	78%	68%	0%
10^{-5}	0.39	85%	78%	68%	1%
10^{-4}	0.39	85%	78%	68%	3%
10^{-3}	8×10^4	84%	76%	65%	26%
10^{-2}	4×10^{12}	73%	66%	50%	95%
10^{-1}	+inf	34%	34%	18%	100%

Table 9.6 Effect of fault injection on the cloud segmentation task at increasing fault probabilities.

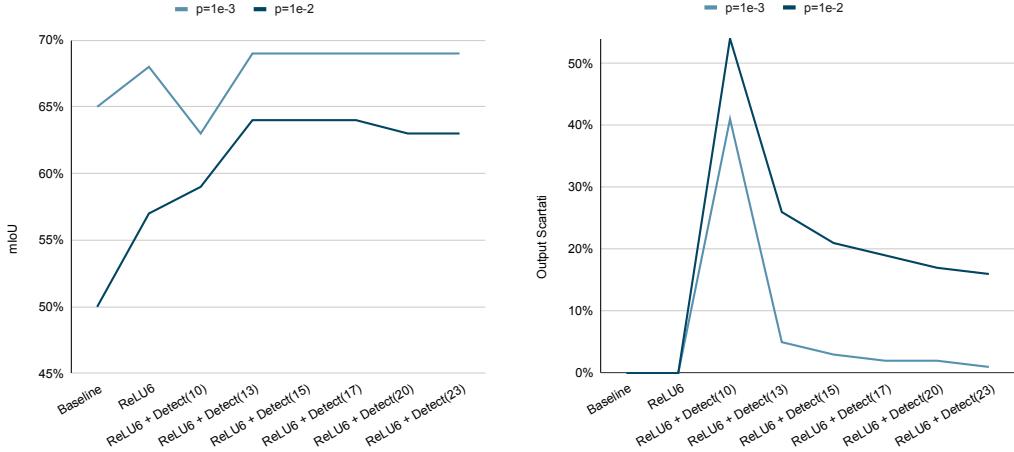


Fig. 9.5 mIoU and discarded samples with different mitigation and detection approaches.

Fault Mitigation

I report the results obtained by replacing the ReLU activation function with the more robust ReLU6. This simple trick already reduces the effects of errors, obtaining +7% of mIoU for $p = 1 \times 10^{-2}$ and +3% for $p = 1 \times 10^{-3}$. This result was so positive that I chose to retrain the network using ReLU6 from the beginning, further improving the results, and using it as the default choice in subsequent experiments.

Fault Detection

As described in the previous section, I employ a statistical approach to detect errors, specifically by verifying the range of network outputs. In the first experiment conducted, the range $[-10, 10]$ is chosen, and all outputs outside this range are discarded. I immediately noticed that the detection system significantly reduces the effect of corruption. This result is justified by the fact that, while fault mitigation aims to absorb the error by keeping all the outputs, fault detection predictions with considerable error are completely discarded. The number of discarded predictions, however, is not insignificant, as each discarded prediction implies making the inference again with the relative use of time and computational power. For this reason, I carry out a study with increasing ranges to find the optimal trade-off between precision and computational overhead (quantifiable

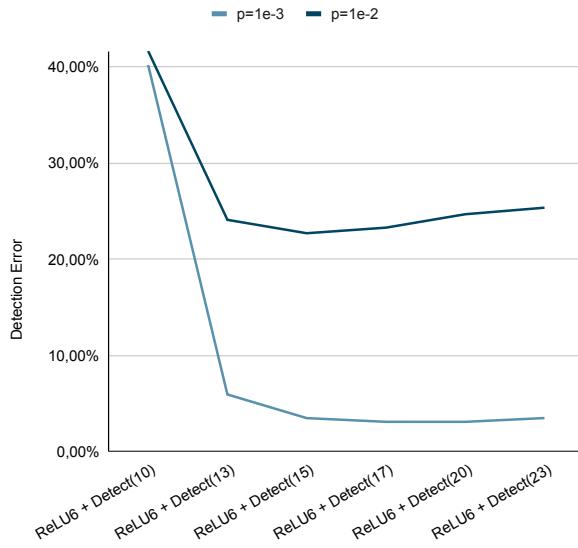


Fig. 9.6 Detection error measured with different mitigation and detection approaches.

with the number of discarded outputs). In particular, I studied output ranges under nominal conditions and tested the detector for range values up to $[-23, 23]$ (Fig. 9.5).

I can see that widening the range significantly reduces the number of discarded predictions (from 54% to 16% for $p = 1 \times 10^{-2}$) and thus saves a substantial amount of calculation. The optimal trade-off should be chosen at the time of the first tests on the integrated system, taking into account the probability of fault measured under realistic conditions. However, at this stage, I can make assumptions that allow us to propose a threshold a priori. The two probabilities of error analyzed ($p = 1 \times 10^{-3}$ and $p = 1 \times 10^{-2}$) allow us to have an estimate of the behavior of the system in a more realistic case of low probability of error ($p = 1 \times 10^{-3}$) and in an exceptionally high situation of errors caused by the passage of the satellite in a specific risk zone (such as, for example, the Van Allen belts [13]). To find a trade-off that is optimal for both risk bands, from the previous charts, a threshold between ± 15 and ± 17 is suitable. These values guarantee the maximum recovery in terms of accuracy (mIoU) with an acceptable number of discarded samples (well beyond the elbow of Fig. 9.5, which is located in ± 13). For further verification, I conducted a validation of the fault detector by comparing the rejected samples with those actually affected by the errors. To make the results more interpretable, I chose to divide the predictions affected

by error into two categories: critical ($>5\%$ of the pixels predicted by an image are altered by faults) and non-critical ($<5\%$ of pixels are changed). The 5% threshold was evaluated by analyzing the impact of faults on outputs in terms of the percentage of pixels affected. The results in Fig. 9.6 therefore confirm that the threshold values for which the detector can identify and filter errors optimally are ± 15 and ± 17 .

9.4 Conclusion

I presented the design of an onboard edge AI system capable of performing inference of multiple tasks in parallel, demonstrating its potential through three segmentation tasks, two of which were tested on custom novel datasets. Combining my self-supervised learning technique and a modular design creates an efficient system that can address complex tasks with low latency and minimal computational demands. The ability to run multiple tasks simultaneously highlights the advantage of the SSL configuration, particularly in resource-constrained environments where parallel processing is essential. Regarding fault robustness, the implemented solutions were chosen to enhance the system's performance with minimal computational overhead. The focus was first of all on the permanent effects of radiation on the weights of the network, which are quantitatively more problematic but at the same time easier to identify. The two solutions for this type of error are deterministic and involve more computation at regular intervals determined by the probability of an event occurring. Regarding faults in activations, which are less problematic but difficult to detect, a simple fault detection system has been designed that can be performed at any inference point in the network with minimal overhead. Despite its simplicity, this system already enables the detection of the vast majority of errors and discards the related predictions.

Bibliography

- [1] E. Magli *et al.*, “A multi-service edge-ai architecture based on self-supervised learning,” in *Proc. of 75th International Astronautical Congress*, IAF, 2024, pp. 1–9.
- [2] H. Heeb, A. Ruehli, J. Janak, and S. Daijavad, “Simulating electromagnetic radiation of printed circuit boards,” in *1990 IEEE International Conference on Computer-Aided Design*, IEEE Computer Society, 1990, pp. 392–393.
- [3] P. Rech, “Artificial neural networks for space and safety-critical applications: Reliability issues and potential solutions,” *IEEE Transactions on Nuclear Science*, vol. 71, no. 4, pp. 377–404, 2024.
- [4] Z. Chen, G. Li, and K. Pattabiraman, “A low-cost fault corrector for deep neural networks through range restriction,” in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, IEEE, 2021, pp. 1–13.
- [5] A. Francis and M. Czerkawski, “Major tom: Expandable datasets for earth observation,” in *IGARSS 2024-2024 IEEE International Geoscience and Remote Sensing Symposium*, IEEE, 2024, pp. 2935–2940.
- [6] C. Aybar *et al.*, “Cloudsen12, a global dataset for semantic understanding of cloud and cloud shadow in sentinel-2,” *Scientific data*, vol. 9, no. 1, p. 782, 2022.
- [7] J.-B. Grill *et al.*, “Bootstrap your own latent-a new approach to self-supervised learning,” *Advances in neural information processing systems*, vol. 33, pp. 21 271–21 284, 2020.
- [8] A. Islam, B. Lundell, H. Sawhney, S. N. Sinha, P. Morales, and R. J. Radke, “Self-supervised learning with local contrastive loss for detection and semantic segmentation,” in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 5624–5633.
- [9] Y. Tang, K. Han, J. Guo, C. Xu, C. Xu, and Y. Wang, “Ghostnetv2: Enhance cheap operation with long-range attention,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 9969–9982, 2022.
- [10] F. F. dos Santos *et al.*, “Improving deep neural network reliability via transient-fault-aware design and training,” *IEEE Transactions on Emerging Topics in Computing*, 2025.
- [11] A. Krizhevsky, G. Hinton, *et al.*, “Convolutional deep belief networks on cifar-10,” *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

- [12] N. Cavagnero, F. Dos Santos, M. Ciccone, G. Averta, T. Tommasi, and P. Rech, “Transient-fault-aware design and training to enhance dnns reliability with zero-overhead,” in *2022 IEEE 28th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, IEEE, 2022, pp. 1–7.
- [13] J. A. Van Allen, G. H. Ludwig, E. C. Ray, and C. E. McIlwain, “Observation of high intensity radiation by satellites 1958 alpha and gamma,” *Journal of Jet Propulsion*, vol. 28, no. 9, pp. 588–592, 1958.

Part IV

Efficient and Robust AI for Precision Agriculture

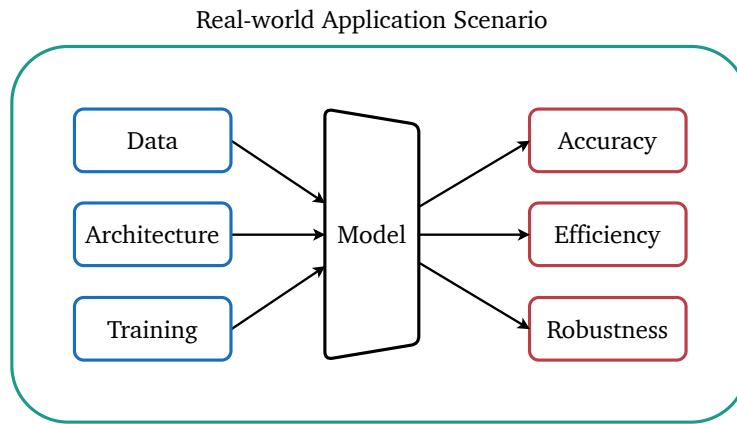


Fig. 9.7 The elements constituting a machine learning model (left) and its properties (right) finally find their purpose in solving complex tasks in unstructured application contexts.

In the previous chapters, I explored how novel machine learning methodologies can enhance the accuracy, efficiency, and robustness of autonomous systems. Now, it is the moment when the properties I ensured for machine learning models come to good use in serving a realistic application (Fig. 9.7). That is when technology can truly push the boundaries of what is possible and, hopefully, help build a better and more just society for all humankind. In the words of Einstein, already quoted at the beginning of the thesis:

Concern for man himself and his fate must always form the chief interest of all technical endeavors, concern for the great unsolved problems of the organization of labor and the distribution of goods in order that the creations of our mind shall be a blessing and not a curse to mankind. Never forget this in the midst of your diagrams and equations.

As global food demand rises due to population growth, traditional agricultural methods are increasingly insufficient to meet production needs while minimizing ecological impacts. Precision agriculture represents a transformative approach to farming that integrates advanced technologies, data analytics, and resource optimization to address the growing challenges of food security, environmental sustainability, and economic efficiency. It leverages tools like GPS, sensors, satellite imagery, and predictive analytics to tailor farming practices to the specific needs of crops and livestock, thereby reducing waste and maximizing yields.

Research in this field is vital not only for enhancing productivity but also for mitigating environmental degradation through the efficient use of water, fertilizers, and pesticides. Furthermore, it fosters resilience against climate change by improving resource management and reducing dependency on new land acquisition. Investing in precision agriculture research can lead to sustainable farming practices that strike a balance between profitability and ecological preservation, ultimately paving the way for a more secure and sustainable future in agriculture.

For this reason, the final part of the thesis focuses on **precision agriculture** applications, showcasing that it is possible to combine multiple AI-based solutions to perform autonomous tasks in unstructured environments. In particular, Chapter 10 presents a deep learning-driven algorithmic pipeline for autonomous navigation in row-based crops, seamlessly integrating global and local path planning.

Chapter 10

A Deep Learning Driven Pipeline for Autonomous Navigation in Crops

Original Paper: *S. Cerrato, V. Mazzia, F. Salvetti, M. Martini, S. Angarano, A. Navone, M. Chiaberge, “A deep learning driven algorithmic pipeline for autonomous navigation in row-based crops”, IEEE Access, 2024.* [1]



Fig. 10.1 Field tests with the Jackal platform in different seasonal periods for the same crop. Lush vegetation and thick canopies greatly reduce GPS precision, affecting its reliability and the overall navigation pipeline. Nevertheless, my proposed segmentation-based algorithm exploits semantic segmentation to provide a proportional controller that drives the robot along the whole row.

Building on the latest deep learning research for computer perception and exclusively using low-range sensors, I present a complete algorithmic pipeline for autonomous navigation in row-based crops [1]. The proposed robust solution is explicitly designed to adapt to seasonal variation, as shown in Fig. 10.1, ensuring complete field coverage in different situations. Moreover, the low price of my methodology significantly reduces maintenance and production costs, enabling a large-scale adoption of fully autonomous machines for precision agriculture.

Firstly, I build on a robust data-driven methodology to generate a viable path for the autonomous machine, covering the crop's full extension with only the field's occupancy grid map. Successively, depending on the vegetation growth status of the crop, I adopt a purely Global Navigation Satellite System (GNSS) local path planning or a vision-based algorithm that exclusively uses a low-cost RGB-D camera to navigate inside the inter-row space [2, 3]. Indeed, meteorological conditions, especially lush vegetation and thick canopies, significantly affect GNSS reliability, degrading its precision and consequently the overall navigation pipeline [4, 5]. Conversely, my vision-based system leverages semantic information to navigate between rows and depth information to refine the underlying control smoothness, thereby disentangling itself from the necessity of precise localization [6]. Moreover, I exclusively use synthetic data and domain randomization [7] to enable affordable supervised learning and simultaneously bridge the domain gap between simulation and reality. Such a technique allows us to construct and train a deep learning model efficiently, which can generalize effectively to various row-based crops.

The proposed methodology ensures autonomous navigation throughout row-based crops, eliminating the need for expensive sensors and accommodating every seasonal variation. The entire algorithmic pipeline has been developed in ROS to simplify communication among different software modules and to facilitate easy deployment on my developing platform, the Jackal Unmanned Ground Vehicle (UGV) by Clearpath Robotics¹. Extensive experimentation and simulations in computer-generated environments, as well as diverse real-world crops, demonstrated the robustness and intrinsic generality of my solution.

Over the past years, autonomous systems designed to navigate row-crop fields have primarily used high-precision GNSS receivers combined with laser-based

¹clearpathrobotics.com

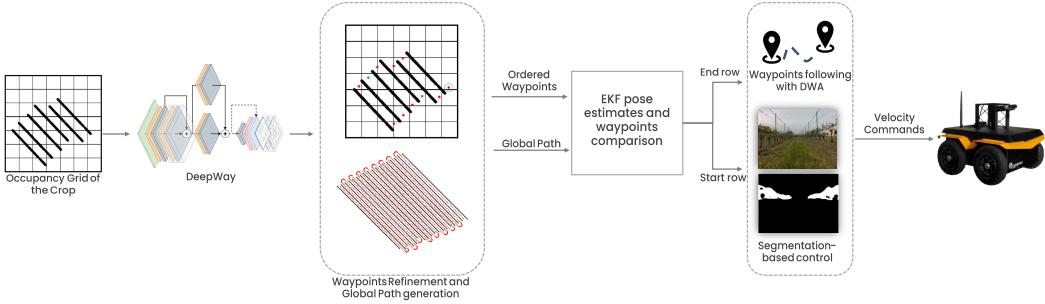


Fig. 10.2 A representation of the overall pipeline. An occupancy grid of the crop is provided as input to the DeepWay model, which estimates the start and end waypoints of the vineyard row. Then, a custom algorithm is responsible for ordering the generated waypoints and computing a global path, maintaining a safe distance from crops. Finally, a local path planner selects the most suitable navigation algorithm based on the UGV's position relative to a waypoint, guiding the mobile platform along the path.

sensors [8, 9]. Nevertheless, the canopies on the sides of the row reduce the GNSS accuracy, affecting its reliability and forcing the adoption of more expensive sensors [4]. Indeed, more robust solutions integrate multiple sensor information (GNSS, inertial navigation systems, wheel encoders) to obtain a more accurate estimation of the mobile platform's location, particularly in the presence of thick canopies and adverse meteorological conditions [10, 11]. However, high adoption of high-range sensors leads to higher system and maintenance costs, preventing a large-scale adoption of self-driving agricultural machinery.

On the other hand, visual odometry (VO) [12, 13] and computer vision-based solutions [14, 15] have been proposed as more affordable approaches. Nonetheless, VO systems exhibit poor performance over long distances due to accumulating errors and struggle with highly similar patterns [16] and unpredictable lighting conditions [17]. Moreover, purely vision-based solutions cannot effectively address seasonal variations, and generalizing to different crops is challenging.

In this state-of-the-art landscape, my team has begun developing a comprehensive, fully functional pipeline to autonomously navigate vineyard rows without relying on multiple expensive sensors.

As already introduced, computer vision and deep learning based algorithms have demonstrated particular robustness in solving problems with noisy signals [18]. Moreover, optimization and edge AI techniques have progressively

made inference computationally affordable, opening up the use of deep learning methodologies to diverse practical applications [19]. Consequently, [20, 21] addressed navigation in vineyard rows using a simple RGB-D camera with a deep learning algorithm. Furthermore, starting with [22], the global path generation automation problem, which the research community has commonly neglected, has been addressed. However, a suitable path generator is crucial for autonomous navigation, and its absence prevents the control of the platform in the field. For that reason, DeepWay [23] was presented, moving from the clustering solution proposed in [22] to detect the rows of the vineyards. It is a robust data-driven approach for global path generation. Besides being more robust and easier to adopt, DeepWay is a highly general approach that can be extended to any row-based crop.

Compared to other autonomous navigation algorithms, my solution utilizes low-cost sensors, including a cost-effective Global Navigation Satellite System (GNSS) receiver with Real-Time Kinematic (RTK) corrections, an RGB-D camera, an Inertial Measurement Unit (IMU), and wheel encoders. My approach aims to bridge the gap between the use of low-cost sensors and the robustness of autonomous navigation in precision agriculture, leveraging the collaboration between AI and standard navigation algorithms. In contrast, existing solutions utilize high-cost sensors, such as 3D LiDARs and expensive RTK-GNSS receivers, to achieve a reliable and robust, yet comprehensive, autonomous solution.

10.1 System Overview

The proposed work aims to present a comprehensive autonomous navigation system for general row-based crops. The designed system is organized into several software modules that should collaborate to obtain adequate and reliable autonomous navigation throughout fields. Fig. 10.2 shows a visualization of the complete pipeline.

Firstly, the system takes a georeferenced occupancy grid map of the considered crop to compute a global path. In particular, it exploits the DeepWay network [23] to estimate each row's start and end waypoints. Then, a custom global path planner [24] computes the desired path, maintaining a safe distance from crops.

Secondly, according to the season period and the amount of vegetation, it is possible to choose the kind of navigation to perform: GNSS-only or AI-assisted.

Suppose a good view of the sky is available both outside and inside the row space. In that case, the system exploits only Real-time Kinematic (RTK) corrections, GNSS signals, and inertial data to guide the mobile platform throughout the crops. On the other hand, when lush vegetation is present, the proposed navigation system utilizes RTK corrections, GNSS signals, and inertial data to switch rows, as a clear sky view is available outside the row space. Within rows, it leverages the camera, deep neural network, and segmentation-based control to mitigate the unreliability of the GNSS signal. In both scenarios, the system strongly relies on estimated global positions of the UGV to follow the provided global path.

My approach addresses the localization problem by combining the positioning information from an RTK-enabled GNSS receiver with the inertial data provided by an IMU. All the data is loosely fused exploiting an Extended Kalman Filter (EKF), which uses an omnidirectional model for prediction and outputs position estimations (x, y, ψ) . x and y are represented in the East-North-Up (ENU) reference frame, and ψ is the robot's yaw relative to the magnetic north, corrected with the actual magnetic declination.

The algorithm selection occurs by comparing the estimated UGV global positions and an ordered list of waypoints that DeepWay computed and successively refined. Finally, it is essential to underline that the DWA navigation scheme is only one possible solution. It is adopted in the presented pipeline for its simplicity, flexibility, and online collision avoidance capability. Further experimentation with even simpler algorithms has been conducted; however, the results have not been reported for brevity. For instance, the Pure Pursuit controller [25] demonstrated promising results between and outside rows, potentially offering advantages in the presence of more densely packed rows or larger vehicles.

10.2 Methodology

In this section, each block of the navigation system is presented, detailing all the aspects of the path planning and navigation processes. My methodology requires

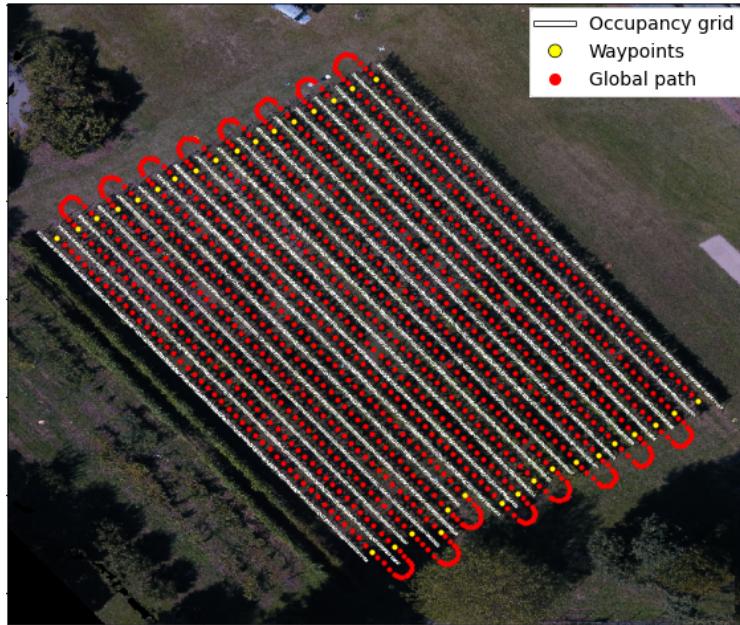


Fig. 10.3 Aerial view of a row crop field, together with the occupancy grid (white), the estimated waypoints (yellow), and the global path (red). The resolution is 0.1 m/px, the end-row distance margin $d_{er} = 20$ px, which results in a real-world margin of 2 m.

as input a georeferenced occupancy grid of the target field $X_{occ} \sim (H, W)$, obtained by segmenting an aerial view of the environment. The system is developed and tested on satellite imagery, but the same methodology can be applied to images obtained by drones flying over the target field. The global path is computed by the first two blocks of the pipeline and is represented as an ordered set of points $\mathcal{P} = \{(x, y) | x, y \in \mathbb{R}\}$ to be followed by the UGV to reach full coverage of the target field. Fig. 10.3 shows an example of an occupancy grid with the predicted global path in red superimposed on the aerial image of the field. Once the global path \mathcal{P} has been generated, the system employs a local path planning policy to navigate the considered crop, selecting the appropriate control algorithm based on the seasonal period and the presence of thick canopies on crops. Indeed, full autonomous navigation exploits the GNSS information and the semantic information obtained by the segmentation network.

10.2.1 Waypoint Estimation

The first block aims to predict the list of l waypoints $\mathbf{W} \sim (l, 2)$ in the occupancy grid reference frame, which represent the beginning and end of each row in the target field. Since classical clustering methods fail under real-world conditions such as rows of varying lengths, holes, and outliers, I adopt the DeepWay framework [23], which frames the waypoint prediction as a regression problem. DeepWay is a fully convolutional neural network that takes as input the occupancy grid \mathbf{X}_{occ} of dimension (H, W) and outputs a map $\hat{\mathbf{Y}} \sim (U_H, U_W, 3)$:

$$\hat{\mathbf{Y}} = f_{DeepWay}(\mathbf{X}_{occ}) \quad (10.1)$$

The first channel of $\hat{\mathbf{Y}}$ is a confidence map that outputs for each cell u the probability $P(u)$ that a waypoint falls in that cell. The output map dimensions are obtained by subsampling the input space of a factor k :

$$U_H = H/k \quad U_W = W/k \quad (10.2)$$

Thus, each cell u represents a square region of $k \times k$ pixels of the original occupancy grid. The other two channels of the output map $\hat{\mathbf{Y}}$ predict for each cell u two compensation factors Δx and Δy used to localize the waypoint inside the u cell, as shown in Fig. 10.4. Those factors are normalized to the $[-1, 1]$ range to represent a distance from the center of the cell. Thus, a factor of 1 means a positive deviation on the corresponding axis of half the length of the cell. Eventually, the final location of the predicted waypoints in the input space can be recovered as:

$$\hat{\mathbf{y}}_O = k \left(\hat{\mathbf{y}}_U + \frac{\Delta + 1}{2} \right) \quad (10.3)$$

where $\hat{\mathbf{y}}_O$ and $\hat{\mathbf{y}}_U$ are the vectors of (x, y) coordinates of a generic waypoint in the input and output reference systems, respectively; Δ is the vector of the $(\Delta x, \Delta y)$ normalized compensation factors.

The prediction confidences stored in the first channel of the output map \hat{Y} are compared to a confidence threshold c_{thr} , and all the positions with $P(u) > c_{thr}$ are selected and projected in the input space as in Eq. (10.3). Furthermore, a suppression mechanism is adopted, similar to standard object detection algorithms, to avoid multiple predictions of the same waypoint. All points falling within a distance threshold d_{thr} of each other are replaced with the one having the maximum confidence $P(u)$. The final waypoints are stored in the list W .

The network is characterized by a stack of N residual reduction modules, which are based on 2D convolutions with *mish* activation [26] and implement both channel and spatial attention [27]. Each module halves the spatial dimension using a reduction block based on 2D convolutions with a stride of two. After N modules, a 2D Transpose Convolution increases the spatial dimension by a factor of two. The last 2D convolution projects a concatenation of the features from the previous two modules into the 3D output space. Since the first and last convolutional layers also have strides of two, the output tensor spatial dimension is reduced by a factor $k = (N + 1)^2$ compared to the input. The final layer uses sigmoid activation for the first channel, which encodes the waypoint probability, and tanh activation for the other two channels, which encode the normalized compensation factors. All code related to DeepWay is open-source².

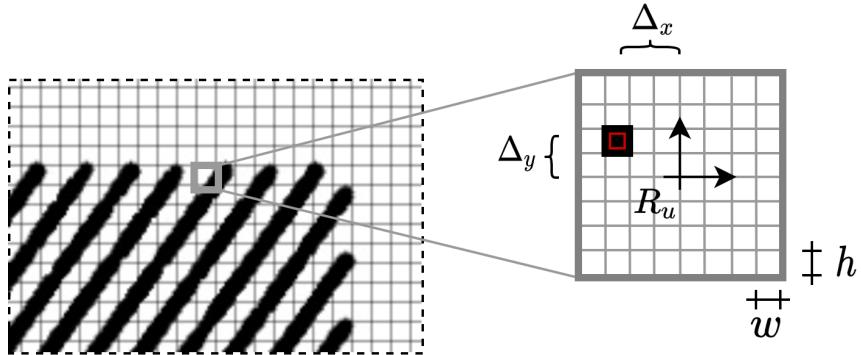


Fig. 10.4 Example of an output map \hat{Y} of DeepWay [23]. Since $k = 8$, each cell u represents a square area of 8×8 pixels of the original occupancy grid. The local reference system R_u is at the centre of the cell, and the compensation factors Δx and Δy are normalized to the $[-1, 1]$ range, as a fraction of the semi-cell length.

²github.com/fsalv/DeepWay

10.2.2 Global Path Planning

The output list of waypoints \mathbf{W} should be ordered to plan a global path that reaches full coverage of the field. I adopt the same post-processing as in [23] with the following steps:

1. The row crops orientation is estimated from the occupancy grid \mathbf{X}_{occ} with the progressive probabilistic Hough transform [28].
2. The waypoints are clustered using the density-based algorithm DBSCAN [29] that creates a variable number of clusters depending on the space density of the waypoints.
3. The points in each cluster are ordered by projecting them along the normal to the direction estimated in step 1.
4. The clusters are merged using a heuristic approach based on their position and size, until two main groups representing the two sides of the field are formed.
5. The final ordered list of waypoints $\mathbf{W}_{ord} \sim (l, 2)$ is obtained by selecting the points from the two main clusters following an A-B-B-A scheme.

The global path is generated from the ordered list \mathbf{W}_{ord} in two steps. The intra-row paths are obtained with [24], which exploits a gradient-based planner between each row's starting and ending waypoints. On the other hand, the inter-row paths are generated with a circular pattern to maintain a safe margin from the end of the rows and avoid collisions during turns. Considering an end-row waypoint \mathbf{p}_i and the successive point that starts the following row \mathbf{p}_{i+1} , the waypoints are firstly moved along the row direction to get an end-row margin d_{er} :

$$\begin{aligned}\mathbf{p}_i^{shifted} &= \mathbf{p}_i + d_{er} \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \\ \mathbf{p}_{i+1}^{shifted} &= \mathbf{p}_{i+1} + (d_{er} + \Delta d) \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix}\end{aligned}\tag{10.4}$$

where α is the angle estimated during the post-processing steps and Δd is the distance between the two points along the row direction:

$$\Delta d = (\mathbf{p}_i - \mathbf{p}_{i+1}) \cdot \begin{bmatrix} \cos\alpha \\ \sin\alpha \end{bmatrix} \quad (10.5)$$

The end-row margin d_{er} can be selected depending on the resolution of the occupancy grid to have a target margin in meters in the real environment. A circular interpolation is employed to connect the shifted points by linearly interpolating the angles, with the mean point serving as the center of the circumference. The whole sequence of points obtained by the intra-row and inter-row planning creates the global path $\mathcal{P} = \{(x, y) | x, y \in \mathbb{R}\}$, defined in the reference system of the occupancy grid 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 the UGV motion. In Fig. 10.3, an aerial view of a field is shown, with the predicted waypoints in yellow and the global path in red.

10.2.3 Segmentation Network

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 $\mathbf{X}_{RGB} \sim (h, w, c)$ and produces a binary map, $\hat{\mathbf{X}}_{seg} \sim (h, w)$ with h , w and c as height, width and channels, respectively. The output positive class segments the crops and the foliage in the camera view. Ideally, it should be evenly distributed on both sides of the frame for a perfectly centered path. Successively, the semantic information of the row, $\hat{\mathbf{X}}_{seg}$, is used in conjunction with its corresponding depth map to control all movements of the platform inside the crop rows.

Among all recent real-time semantic segmentation models, I carefully select an architecture that guarantees high accuracy levels while also considering hardware costs, optimization simplicity, and computational load. Indeed, the segmentation-based control does not significantly benefit from fine-grained predictions and elaborated encoder-decoder networks [30] or two-pathway backbones [31], which do not bring about any substantial improvement. Therefore, I adopt a very lightweight backbone, MobileNetV3 [32], followed by a reduced version

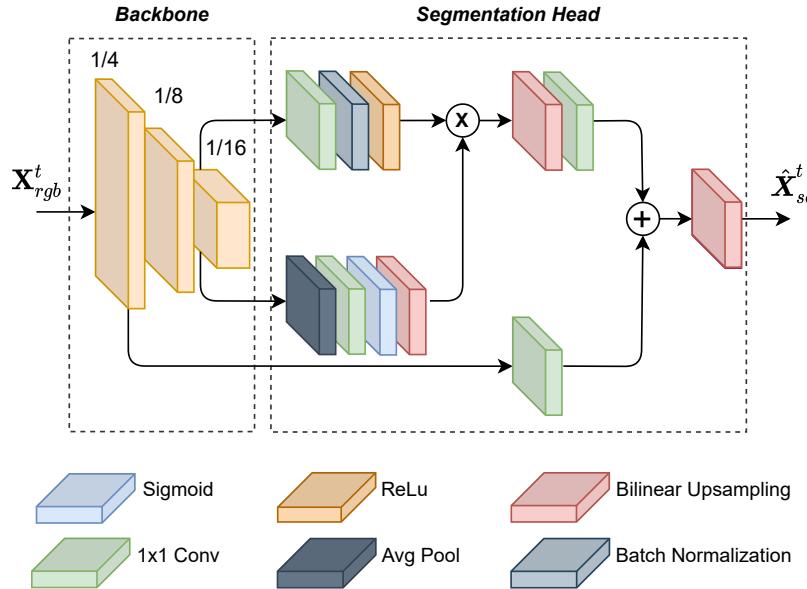


Fig. 10.5 Graphical representation of the architecture of the segmentation network. Features at different resolutions are fed into the segmentation head, which combines them to produce the output binary map, \hat{X}_{seg} .

of the Atrous Spatial Pyramid Pooling module [33] 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 Fig. 10.5. The backbone extracts contextual information through repeated spatial reduction, and two of its branches, operating at different resolutions, feed the segmentation head. One layer applies an atrous convolution at $1/16$ resolution to extract denser features, and the other one is used to add a skip connection at $1/4$ resolution to work with more detailed information. Finally, I employ a 224×224 input to maintain real-time performance even without hardware accelerators. Therefore, I rescale the global average pooling layer, setting the kernel size to 12×12 with strides of 4 and 5. Additionally, to ensure equal input and output dimensions, I apply a final bilinear upsampling with a factor of 8 to the end of the segmentation head.

10.2.4 Segmentation-based Control

The segmentation masks $\hat{X}_{seg} \sim (h, w)$ provided by the deep neural network are post-processed and fed into a custom control algorithm to generate consistent velocity commands to drive the UGV inside the inter-row space and maintain as much as possible the inter-row centrality. As in [21], I compute a sum of S segmentation maps along with an intersection with depth information provided by an RGB-D camera to obtain a more stable control. First, I pick S consecutive segmentation maps at times $\{t - S, \dots, t\}$ and I fuse them

$$\hat{X}_{cumSeg}^t = \sum_{n=0}^S \hat{X}_{seg}^{t-n} \quad (10.6)$$

then, I join the depth information $X_{depth}^t \sim (h, w)$ to reduce the line of sight of the actual scene and remove some background noise. The line of sight is limited to a fixed value generating a binary map $X_{depthT}^t \sim (h, w)$, as follows:

$$X_{depthT}^t_{i=0:h, j=0:w}(i, j) = \begin{cases} 0, & \text{if } (X_{depth}^t)_{i,j} \geq d_{depth} \\ 1, & \text{if } (X_{depth}^t)_{i,j} < d_{depth} \end{cases} \quad (10.7)$$

where d_{depth} is a fixed experimental scalar. Finally, exploiting an interception operation between the cumulative output \hat{X}_{cumSeg}^t , computed in Eq. (10.6) and the binary map X_{depthT}^t previously generated, I obtain the pre-processed input $X_{ctrl}^t \sim (h \times w)$ for the control algorithm:

$$X_{ctrl}^t = \hat{X}_{cumSeg}^t \cap X_{depthT}^t \quad (10.8)$$

In the X_{ctrl}^t binary map, 1 stands for obstacles and 0 for free space. The segmentation-based control algorithm is developed by building upon the SPC algorithm presented in [21]. Indeed, I propose a simplified version of the control function to eliminate unnecessary conditional blocks, thereby achieving a more real-time control algorithm.

Algorithm 1 contains the pseudo-code of the proposed custom control, which, starting from a pre-processed segmented image X_{ctrl}^t , is responsible for computing the driving velocity commands. First, a noise reduction function removes

Algorithm 1 Segmentation-based local path planner

Require: X_{ctrl}^t : Pre-processed segmented image
Ensure: v_x, ω_z : Continuous control commands

```

1: noise_reduction_function()
2: for  $i=0, \dots, w$  do
3:    $c \leftarrow \text{sum\_columns}(X_{ctrl}^t)$ 
4: end for
5: zeros  $\leftarrow \text{list\_zero\_clusters}(c)$ 
6: max_cluster  $\leftarrow \text{find\_max\_cluster}(\text{zeros})$ 
7: if cluster_length(max_cluster)  $\geq anomaly_{th}$  then
8:    $v_x, \omega_z \leftarrow 0, 0$ 
9: else
10:  compute_cluster_center()
11:   $v_x, \omega_z \leftarrow \text{control\_function}()$ 
12: end if
```

undesired noise in the bottom part of the cumulative segmentation mask X_{ctrl}^t , which is caused by grass on the terrain and may be incorrectly segmented by the neural network. To perform such operation, I compute the sum over rows of X_{ctrl}^t obtaining an array $g_{noise} \sim (h)$, then I set $X_{ctrl(indices,:)}^t = 0$, where $indices$ contains the matrix-row indices such that $g_{noise} < th_{noise}$, with $th_{noise} = 0.03 \cdot max(g_{noise})$ as threshold. I perform this operation because, in an ideal segmentation mask, there are no ones at the top or bottom of the image, while the majority are supposed to be in the central belt. After the noise reduction phase, I store the sum over columns of the obtained matrix X_{ctrl}^t in the array $c \sim (w)$, which contains the number of segmented trees for each column. Therefore, every zero in c represents a potential space where the mobile platform can be routed. Then, I select the clusters of zeros in c , which are the groups of consecutive zeros, to store them in the list **zeros**. Next, I look for the largest cluster of zeros **max_cluster** and in case of the length of such cluster is over an empirically chosen threshold, $anomaly_{th} = 0.8 \cdot w$, the driving commands are set to zero value, because it means the provided cumulative segmentation mask X_{ctrl}^t has more zeros than ones, that is an anomaly, so for safety reason the mobile platform is stopped. In the absence of anomalies, I compute the cluster center, which is provided as input to the control function. The identified cluster contains obstacle-free space information that can be utilized to guide 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:

$$\omega_z = -\omega_{z, gain} \cdot d \quad (10.9)$$

$$v_x = v_{x, max} \cdot \left[1 - \left[\frac{d^2}{(\frac{w}{2})^2} \right] \right] \quad (10.10)$$

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 X_{ctrl}^t and d is defined as:

$$d = x_c - \frac{w}{2} \quad (10.11)$$

with x_c center coordinate of the selected cluster. Eq. (10.9) represents the angular velocity control function, while Eq. (10.10) has been used to compute the linear velocity, as in [34] and [21]. Eventually, the control velocity commands sent to the actuators are smoothed using the exponential moving average (EMA), as formalized in Eq. (10.12), to prevent the mobile platform from experiencing sharp motion.

$$EMA_t = EMA_{t-1} \cdot (1 - \alpha_{EMA}) + \begin{bmatrix} v_x \\ \omega_z \end{bmatrix} \cdot \alpha_{EMA} \quad (10.12)$$

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

10.2.5 Local Path Planning Policy

Once the global path P and the row's start and end waypoints W_{ord} have been correctly generated, I utilize the provided information to navigate locally throughout the entire field. In case of lush vegetation and thick canopies that may distort GNSS signals inside the inter-row space, the local navigation problem is solved

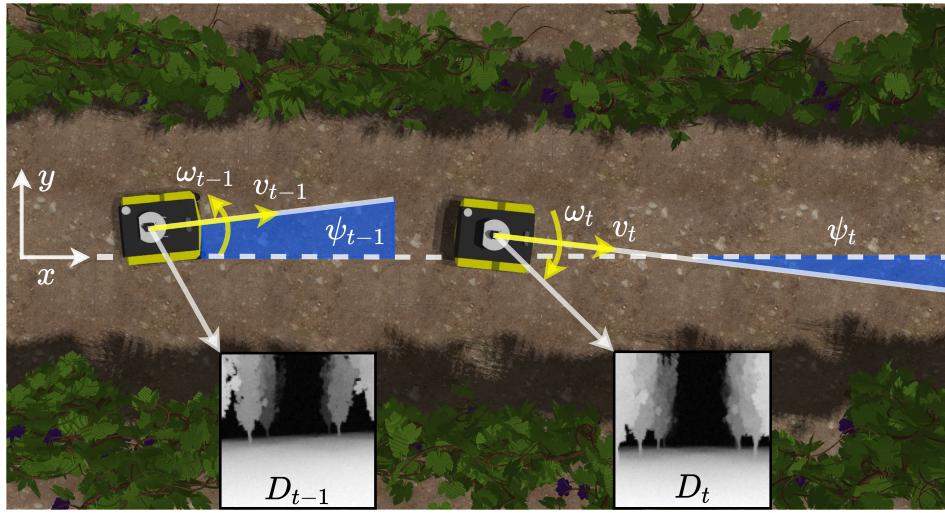


Fig. 10.6 At each time instant t , the proposed agent receives as inputs a raw noisy depth image D_t , the previous velocity commands $[v_{t-1}, \omega_{t-1}]$ and the yaw ψ_t , to generate the new commands $[v_t, \omega_t]$. Since no explicit localization is required, the agent performs a positioning-independent navigation in the vineyard row. During training, the reward at each time step t is computed as a function of the distances from the end of the row (EoR) d_t and d_{t-1} , and the angle ϕ_t between the robot orientation and the shortest path to EoR.

using the synergy of two different algorithms according to the kind of navigation requested in a determined place of the considered crop:

1. Inside the inter-row space, I exploit the custom control algorithm, described in Section 10.2.4 and based on the segmentation information provided by the deep neural network, to overcome localization inaccuracies due to blocked GNSS signals by overgrown plant vegetation.
2. Switching between different rows, I use the standard DWA [35] with fine-tuned parameters to follow the circular path generated in Section 10.2.2, since a clear view of satellites and sky should be available.

The choice of a suitable algorithm is made by comparing the estimated position from the EKF localization filter with the provided row waypoints. In the case of start row recognition, the local path planning policy selects the segmentation-based control; otherwise, it uses the DWA, as shown in Fig. 10.2. The comparison happens by computing a simple Euclidean distance between the row waypoint and the estimated position. If the distance is lower than a threshold (i.e., $d_w = 0.5$),

the algorithm considers the waypoint as reached and selects the right local controller.

On the other hand, during specific year periods when plant vegetation is not dense and overgrown, the local path planning policy utilizes only DWA to navigate throughout the entire field, thanks to good satellite reception, following the complete global path generated in Section 10.2.2.

10.2.6 An Alternative Approach: Reinforcement Learning

As an alternative to the local path planner proposed in the pipeline, I mention an end-to-end approach based on reinforcement learning presented in [36]. The method is position-agnostic and can reach the end of the row without global localization. The vision and control systems are coupled, and the policy learns to directly map sensor state vectors and motion during training, resulting in a sensorimotor agent. The controller is environment-independent, since there is no need for adaptations to deal with unseen scenarios (e.g., curved rows). The system is computationally efficient and optimized to run directly onboard the robotic platform in real-time. I report here only the methodology; the curious reader is encouraged to consult the paper for further details.

Task Formulation

I model the navigation problem within a reinforcement learning framework. Therefore, the problem is formulated as a Markov Decision Process (MDP) described by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma)$ [37]. An agent starts its interaction with the environment in an initial state s_0 , drawn from a pre-fixed distribution $p(s_0)$ and then cyclically select an action $a_t \in \mathcal{A}$ from a generic state $s_t \in \mathcal{S}$ to move into a new state s_{t+1} with the transition probability $\mathcal{P}(s_{t+1}|s_t, a_t)$, receiving a reward $r_t = R(s_t, a_t)$.

A reinforcement learning process aims to optimize a parametric policy π_θ , which defines the agent's behavior once trained. In the context of autonomous navigation, I model the MDP with an episodic structure with maximum time steps T , hence the agent is trained to maximize the cumulative expected reward $\mathbb{E}_{\tau \sim \pi} \sum_{t=0}^T \gamma^t r_t$ over each episode, where $\gamma \in [0, 1]$ is the discount factor. More

in detail, I use a stochastic agent policy in an entropy-regularized reinforcement learning setting, in which the optimal policy π_θ^* with parameters θ is obtained by maximizing a modified discounted term:

$$\pi_\theta^* = \arg \max_{\pi} \mathbb{E}_{\tau \sim \pi} \sum_{t=0}^T \gamma^t [r_t + \alpha \mathcal{H}(\pi(\cdot|s_t))] \quad (10.13)$$

Where $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy term which increases robustness to noise through exploration, and α is the temperature parameter which regulates the trade-off between reward optimization and policy stochasticity.

Reward

The potential outcome of the vineyard navigation task can be easily summarized in a binary output: the robot successfully arrives at the end of the row, or it does not. However, this sparse reward feedback can be assigned to the agent only in the case of a completed successful episode, which is an improbable event during the initial learning phases. According to this, reward shaping is the typical process that leads researchers to analytically specify the desired behaviour to the agent, thanks to a dense reward signal assigned at each time step. Moreover, this approach enables the expression of secondary desired behaviors. In this application scenario, I identify three key features of an ideal optimal policy: a complete collision-free travel in the vineyard row, a centered trajectory, and a proper orientation. Nonetheless, the reward can be computed while training in a simulated environment, exploiting positioning data that is not needed by the agent at test time. To this end, I first define a reward contribution r_h to keep the robot oriented towards the end of the row:

$$r_h = \left(1 - 2 \sqrt{\left| \frac{\phi}{\pi} \right|} \right) \quad (10.14)$$

where ϕ is the heading angle of the robot, namely the angle between its linear velocity and the end of the row. I consider r_h to be a fundamental feedback mechanism that enables the agent to understand how to counteract the sudden angular

deviations imposed by the irregular terrain, which is realistically generated in the simulated environment. Then, to obtain a central trajectory, I consider the possibility of directly scoring the distance of the robot from the center of the row. However, this approach requires comparing the robot pose with the mean line of each specific row at each step. Nonetheless, it results in a slow and inefficient policy when combined with the heading reward r_h . For this reason, I prefer a distance-based reward to strongly encourage the agent to reach the end of the row:

$$r_d = d_{t-1} - d_t \quad (10.15)$$

where d_{t-1} and d_t are Euclidean distances between the robot and the end of the row (EoR) at successive time steps, as shown in Fig. 10.6. Robot pose information is uniquely used for reward computation during training and is not included as agent input, as better specified in the following Section 10.2.6.

I finally include a sparse reward contribution for end-of-episode states, assigning $r_s = 1,000$ for the successful completion of the task, $r_c = -500$ if a collision occurs, and $r_\psi = -500$ if the robot overcomes a $\pm 85^\circ$ yaw limit. Stopping the episode when the robot exits the vineyard row or reverses its motion direction is crucial to maintaining meaningful sample transitions for the task.

The final reward signal results in the following:

$$r = a \cdot r_h + b \cdot r_d + \begin{cases} r_s & \text{if success} \\ r_c & \text{if collision} \\ r_\psi & \text{if reverse} \end{cases} \quad (10.16)$$

Where $a = 0.6$ and $b = 35$ are numerical coefficients to integrate the diverse reward contributions in the final signal efficiently.

Policy Network

I define the parametrized agent policy with a deep neural network. I train the agent using the Soft Actor-Critic (SAC) algorithm presented in [38], which enables a continuous action space. In particular, I instantiate a stochastic Gaussian policy for the actor and two Q-networks for the critics.

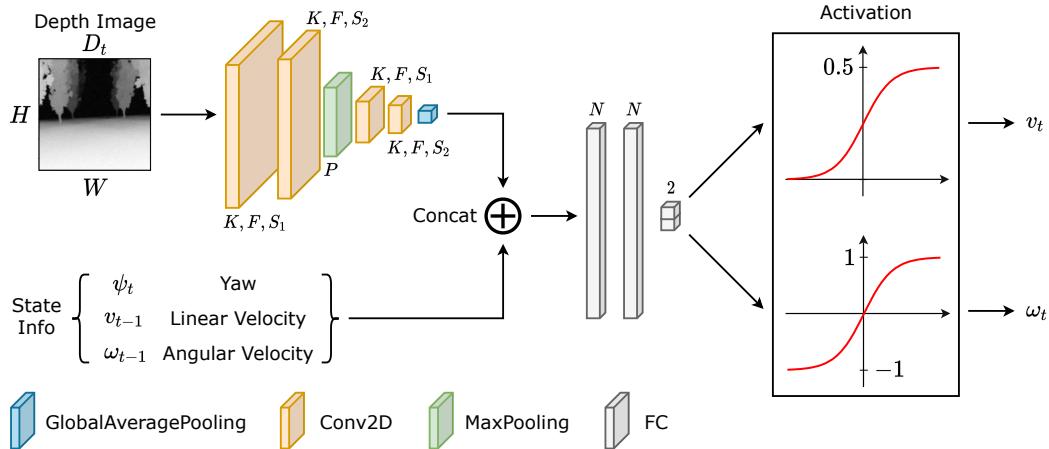


Fig. 10.7 Architecture of the actor policy network. A convolutional backbone extracts features from the depth image D_t . The features are then concatenated to the input vector $[v_{t-1}, \omega_{t-1}, \psi_t]$ and two fully connected layers output the action vector $[v_t, \omega_t]$ with specific activation functions.

Input features I select the input features of the policy network, only considering the odometric and perception data available during the vineyard navigation task. To this end, I identify a set of input features that enable localization-independent navigation and an affordable perception system, such as a simple depth camera. Several iterations led us to select three key elements as input for the agent.

1. The previous set of velocity commands $[v_{t-1}, \omega_{t-1}]$ to provide information about its current motion and temporal continuity to the agent and avoid strongly oscillating or disentangled commands.
2. The yaw of the robot ψ_t measured at the current time instant t , which is always available thanks to an IMU sensor on the robot. The simple yaw angle does not represent the required optimal heading angle ϕ_t for the task. Indeed, such an angle depends on the robot's position, which is not available at test time in GPS-denied conditions. However, the yaw ψ_t provides awareness about the actual orientation of the robot and helps in generating smooth, collision-free velocity commands.
3. A raw noisy depth image of size 112×112 . Each pixel of the image contains a distance in the range of $[0.0, 5.0]$ m, and it is the only perception data the agent exploits to guide the robot through the vineyard. Processing the image, the agent acquires the necessary knowledge on obstacles, and it can also visually infer its current orientation relative to the end of the

visible row. A reduced image size enables the acquisition of a compact latent representation comprising only 32 features. The choice of depth images is also motivated by the aim of reducing the simulation-to-reality gap. Indeed, depth images are only marginally affected by visual features. On the other hand, real depth images may present peculiar noisy behaviors, so I add to each raw image two different noises: a first uniform random noise with values in $[-0.5, 0.5]$ m, and a second random noise proportional to the depth values in the image, also in the range $[-0.5, 0.5]$ m. The second noise aims to disturb the points at higher distances more heavily. Overall, each pixel presents at most a perturbation of ± 1 m.

Output actions The policy network predicts an action $a_t = [v_t, \omega_t]$ at each time step, which directly represents the required linear and angular velocity command to control the robot. This choice is primarily determined by the differential-drive steering model of the robotic platforms I use, and it also enables us to integrate the system with standard ROS messages.

Network architecture The architecture of the policy network presents a multi-input structure, represented in Fig. 10.7. A convolutional feature extractor is designed to efficiently map the depth image in a compact latent representation, inspired by the work proposed for visual SAC in [39]. However, I avoid adopting an encoder-decoder structure and an additional reconstruction loss. My solution enables the agent to autonomously learn how to extract relevant features for the task at hand. Firstly, the feature extractor takes the depth image $D_t \sim (H, W)$ as input. It consists of two convolutional layers (kernel size $K = 3$, $F = 32$ filters, strides $S_1 = 2$ and $S_2 = 1$ respectively) with ReLU activations, followed by a max pooling layer (pool size $P = 2$), two more convolutional layers (with the same structure as the previous ones), and a global average pooling layer. The features are then concatenated to the position-agnostic robot state input vector, which includes the measured yaw ψ_t and the previous action command, consisting of the linear velocity v_{t-1} and the angular velocity ω_{t-1} . The resulting vector is processed by two fully connected layers with $N = 256$ neurons and ReLU activation. Finally, an additional output layer predicts the action vector $[v_t, \omega_t]$, using \tanh as activation function. The linear velocity v_t is further squashed to $(0, 0.5)$ to match the velocity profile of the robot.

As I use a stochastic Gaussian policy, the network predicts the mean and the standard deviation of each action distribution, which are used to sample a value from the derived distribution while training. Instead, the mean value is directly used as output at test time. The critic network structure is identical to the actor one, except that it also takes the predicted action vector as input and outputs an estimate of the Q value, and is trained according to the SAC algorithm.

Random Initialization Strategy A critical condition of the vineyard environment is its constrained geometry. As a consequence, the agent typically collides in a few steps during the early stage of DRL training, exploring a drastically reduced number of states in the environment. This behavior negatively impacts the generalization properties of the agent and, more broadly, the likelihood of the training algorithm achieving stable convergence. For this reason, I identify a set of counteractions to train the policy effectively:

- The vineyard training stage comprises different vineyard rows where the robot travels during the same training simulation to encourage better generalization derived from a higher number of visited states.
- The robot is initialized in a new random pose in the vineyard every 10 episodes, varying its $[x_0, y_0]$ position coordinates and its initial yaw ψ_0 , enabling the agent to travel the rows in both directions. Consequently, the agent can visit the final sections of rows from the beginning of the simulation, significantly speeding up the convergence of the training with better generalization results.
- An initial exploration phase is combined with an additional ε -greedy policy, which samples random values in the action spaces with a probability that is exponentially reduced as the number of episodes increases, to maintain a proper level of exploration throughout the entire training.

10.3 Experiments

In this section, I describe the main experiments conducted in both simulation and real environments to better validate the algorithms. First, I discuss the creation of synthetic datasets for the training of the two deep neural networks. I then illustrate the training process and the optimization techniques employed to

minimize inference costs. Finally, I conclude with the simulation and evaluations in a real environment.

All the code is compatible with ROS to exploit some of its most used packages in robotics research (e.g., *move_base*³ and *robot_localization*⁴, that offer ready-to-use local planners and basic localization methods, respectively). All tests have been performed using Ubuntu 18.04 and ROS Melodic.

10.3.1 Dataset

Regarding the two presented deep neural networks, they require training on specific datasets tailored to the desired final application. Supervised learning training algorithms are the easiest to adopt, with usually the best final results. However, they all require supervision through a labeled training dataset. That significantly impacts costs and complicates data collection, making it a time-consuming process. Therefore, I exclusively use synthetic data and domain randomization [7] to enable affordable supervised learning and simultaneously bridge the domain gap between simulation and reality.

For DeepWay, a dataset of randomly generated occupancy grids is created. All parameters, such as number, orientation, depth, and length of the rows, are randomly selected. The coordinates of the starting and ending points for each row are generated through geometrical reasoning, and the occupancy grid is obtained by placing circles of different radii at each location between the two corresponding points. Random holes are also created in the rows to increase the variability of the images. Ground truth waypoints are obtained to always lie within the rows, as I have experimentally found that this helps the network's prediction. Given a starting point a and an ending point b from two adjacent rows, I compute the target waypoint as follows:

$$\mathbf{p}_{a,b} = \begin{bmatrix} 0 & \mp 1 \\ \pm 1 & 0 \end{bmatrix} \frac{\mathbf{a} - \mathbf{b}}{2} + \frac{\mathbf{a} + \mathbf{b}}{2} \quad (10.17)$$

³wiki.ros.org/move_base

⁴docs.ros.org/robot_localization

that corresponds to a $\pm 90^\circ$ rotation around the mean point, where the sign is selected to make the waypoint inside the row. I train DeepWay using a total of 3,000 synthetic images and validate it with 100 manually annotated satellite images taken from the Google Maps database.



Fig. 10.8 An example of a synthetic RGB image (a) and the corresponding segmentation mask (b).

On the other hand, the segmentation neural network requires a set of RGB images along with segmentation masks as inputs to be correctly trained. As a consequence, inspired by the work of [15], I generate synthetic RGB images coupled with the corresponding segmentation masks. For such purpose, I exploit Blender 2.8⁵, which is an open-source 3D computer graphics software compatible with Python language, and the Modular Tree⁶ addon to speed up tree generation. I design four main scenarios: two isolated trees, a group of heterogeneous trees, and a row-based scenario with various backgrounds and soils. Then, I write a script that can automatically capture RGB images and the corresponding segmentation masks of the scene from different positions and under varying illumination conditions, to obtain a randomized and complete synthetic dataset. An example of a synthetic RGB image, along with the corresponding segmentation mask, is shown in Fig. 10.8. Every rendering step takes approximately 30 seconds, resulting in roughly 23 hours of continuous work on an NVIDIA RTX 2080 GPU for all 2,776 samples. The total number of images, in conjunction with transfer learning, enables the training of a segmentation network with high generalization capabilities while minimizing data generation costs. The test set comprises 100 manually annotated images acquired in a real-world environment (Valle San Giorgio di Baone, Italy).

⁵blender.org

⁶github.com/MaximeHerpin/modular_tree

10.3.2 Training

To train both networks, I utilize TensorFlow⁷ on a PC equipped with 32 GB of RAM, an Intel i7-9700K CPU, and an NVIDIA GeForce 2080 Super GPU. DeepWay is trained following the methodology presented in [23], with an input dimension of $H = W = 800$ and $N = 2$ Residual Reduction modules, resulting in a subsampling factor of $k = 8$ and output dimensions of $U_H = U_W = 100$. I select a kernel size of 5 and 16 filters for all the convolutional layers, except the first and last ones, which have kernel sizes of 7 and 3, respectively. These hyperparameters were selected by performing a grid search over reasonable sets of values and adopting those that experimentally yielded the best convergence. As a loss function, a weighted mean squared function (L_2) is used to compensate for the higher number of negative cells (i.e., with no waypoint in the target image) compared to positive ones. I set these weights to 0.7 for the positive cells and 0.3 for the negative ones. The default distance threshold for the waypoint suppression algorithm is set to $d_{thr} = 8$ pixels, which is the minimum inter-row distance in my dataset, and the confidence threshold to $c_{thr} = 0.9$, to select only the most confident predictions. As in standard object detection algorithms, I adopt the Average Precision (AP) as a metric for the prediction quality. I compute the AP at different distance ranges d_r . A prediction is considered a True Positive (TP) only if it falls within a distance d_r from the target waypoint. On the 100 real-world images, I reach an AP of 0.9794 with $d_r = 8$ pixels, 0.9558 with $d_r = 4$ pixels, and 0.75 with $d_r = 2$ pixels.

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

Table 10.1 Comparison between different devices' energy consumption and inference performances with graph optimization (GO) and weight precision (WP).

⁷tensorflow.org

Regarding the segmentation network, I train my model by applying transfer learning to the selected backbone. Indeed, rather than using randomly initialized weights, I utilize MobileNetV3 variables derived from an initial training phase on the 1.3 million images of the ImageNet dataset [40]. Moreover, I pre-trained the overall segmentation network using Cityscapes [41], a publicly available dataset comprising 30 different classes and 5,000 annotated images. Finally, I employ a robust data augmentation approach that includes random cropping, brightness adjustment, saturation adjustment, contrast adjustment, rotation, and flipping. Together, these techniques significantly enhance the model's final robustness and overall generalization capability, using a reduced number of training samples. I train the network using stochastic gradient descent with a learning rate of 0.03 and Intersection over Union (IoU) as the loss function. 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 gap of 0.08. Moreover, my experimentation shows that larger input sizes improve segmentation over the synthetic dataset, but dramatically reduce accuracy over authentic images.

The trained network is optimized to reduce latency, inference cost, memory usage, and storage footprint. That is achieved through two distinct techniques: model pruning and quantization. The first simplifies the topological structure by removing unnecessary parts of the architecture, favoring a more sparse model that introduces zeros to the parameter tensors. Subsequently, with quantization, I 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 10.1, experimentation results with some reference architectures are summarized.



Fig. 10.9 A visual representation of the simulation environment.

10.3.3 Hardware

As a mobile platform, I have selected the Jackal UGV by Clearpath Robotics, which can be briefly described as a small, weatherproof rover with a 4×4 high-torque drivetrain. It is highly customizable and ROS-compatible, allowing fast deployment and algorithm testing. All the algorithms run on Jackal's onboard Mini-ITX PC, which features an Intel Core i3-4330TE processor at 2.4 GHz and 4 GB of DDR3 RAM. Regarding the localization sensors, the RTK-enabled GNSS receiver is the Piksi Multi by Swift Navigation⁸ mounted on an evaluation board, which provides easy input/output communication with the receiver at 10 Hz. In contrast, the inertial measurements are provided by the MPU-9250 IMU, with an acquisition rate of about 100 Hz. In addition, to get a front view of the environment, I select the Intel RealSense D455 RGB-D camera, which provides 30 FPS and is mounted on the front part of Jackal's top plate. Finally, the odometry is provided by the onboard quadrature encoders, which can operate at a rate of 78,000 pulses per meter. As mentioned in Section 10.1, the IMU and GNSS receiver data are fused using an EKF to obtain a global position estimate of the mobile platform at each time step, described in terms of (x, y, ψ) . However, GNSS positioning is highly inaccurate, with an error of 3 to 5 m without implementing any correction techniques. As a consequence, I provide RTK corrections to Piksi Multi receiver, coming from the SPIN3 GNSS⁹ of Piemonte, Lombardia, and Valle d'Aosta, through the Internet. Then, the GNSS receiver directly utilizes such corrections to obtain more reliable and accurate global position estimates, with an error range of 5 to 10 cm, provided the sky is clear and the antenna is in a good position. I utilize such a correction service because it is entirely free and can send out RTK corrections over the Internet.

10.3.4 Results

Simulated Environment

The presented pipeline is tested in a simulation environment before being applied in real-world tests to assess its basic performance and establish a preliminary

⁸swiftnav.com

⁹spingnss.it/spiderweb

	T1	T2	T3	T4	T5	T6
N_{rows}	4	4	4	4	4	4
Min. Error [m]	0.001	0.002	0.002	0.001	0.002	0.002
Max. Error [m]	0.726	0.678	0.600	0.633	1.21	1.21
MAE [m]	0.068	0.085	0.077	0.082	0.215	0.217
RMSE [m]	0.089	0.100	0.092	0.096	0.263	0.265
σ [m]	0.057	0.053	0.050	0.050	0.152	0.152

Table 10.2 Comparison of different error metrics in six different tests performed in two different simulation environments. The first row describes the number of visited rows in the corresponding test.

experimental setting for various hyperparameters. First, I built a custom simulation environment consisting of vine plants organized in rows and a bumpy, uneven terrain, as shown in Fig. 10.9, using the Gazebo¹⁰ simulator, which is ROS-compatible and open-source. Moreover, it provides advanced 3D graphics, dynamic simulation, and several plugins to simulate sensors, as GNSS, IMU, and cameras.

Then, I compare the UGV trajectory obtained using the proposed methodology with a ground truth line to evaluate different error metrics: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Standard Deviation (σ), as shown in Table 10.2. The ground truth is computed by manually annotating GNSS simulated positions that correspond to an ideal global path, centered in the inter-row space and with a safe distance from crops, while switching between two different rows. Then, such points are linearly interpolated.

I have conducted six different tests in two distinct simulation environments, varying the positions of the vine plants while maintaining a row-based organization. The obtained performances are promising in terms of MAE, RMSE, and σ , as shown in Table 10.2. The worst attained results are stated by an $RMSE = 0.265$ m and an $MAE = 0.217$ m in the sixth test, while the best one is achieved in the first test with an $RMSE = 0.089$ m and an $MAE = 0.068$ m. Finally, the mean time to complete a test is approximately 4 minutes, with a maximum speed of $0.5m/s$. All considered, the overall achieved performances are satisfactory, taking into account that the worst results are obtained in the last two tests, where the vineyard rows are organized with a slightly curved shape.

¹⁰gazebosim.org



(a) Vineyard Row

(b) Pear Orchard Row

Fig. 10.10 A visual representation of the real-world testing environments.

Real Environment

The overall system is extensively tested in two real environment scenarios with multiple experiments in different seasonal periods (Fig. 10.1): a vineyard and a pear orchard, shown in Fig. 10.10, for a total of more than 80 hours of experimentation from 9 a.m. to 6 p.m., but without particular adverse weather conditions (e.g., rain, snow, fog). Moreover, all tests have been performed using the same hardware and software setup to ensure consistent data. The vineyard is located in Grugliasco and managed by the Department of Agricultural, Forestry and Food Sciences of Università degli Studi di Torino. In contrast, the pear orchard is situated in Montegrosso d'Asti and managed by Mura Mura farm¹¹. The first scenario features an inter-row space of approximately 2.80 m and a height of approximately 2.0 m, while the second is organized in rows with an inter-row space of 4.50 m and a height of roughly 3.0 m.

Test	N_{rows}	$e_{min} [m]$	$e_{max} [m]$	MAE [m]	RMSE [m]	$\sigma [m]$
1	4	0.008	1.395	0.523	0.627	0.351
2	4	0.002	1.105	0.457	0.551	0.315
3	2	0.007	1.320	0.659	0.755	0.375

Table 10.3 Comparison of different error metrics in three different tests performed in the pear orchard. The second column describes the number of visited rows in the corresponding test.

The errors, described in Table 10.3 and Table 10.4, are computed by comparing the RTK-GNSS positions provided by the Piksi Multi receiver and the global path

¹¹muramura.it

Test	N_{rows}	e_{min} [m]	e_{max} [m]	MAE [m]	RMSE [m]	σ [m]
1	4	0.013	0.621	0.296	0.332	0.160
2	6	0.006	0.598	0.218	0.240	0.119
3	6	0.003	0.720	0.204	0.246	0.145

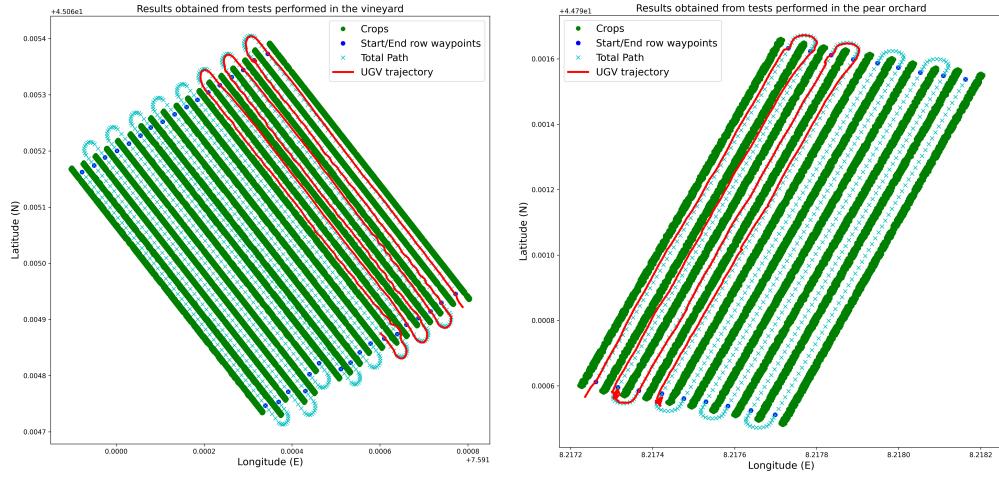
Table 10.4 Comparison of different error metrics in three different tests performed in the vineyard. The second column describes the number of visited rows in the corresponding test.

supplied 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, for this analysis, they have been referred to the known GNSS coordinate of the top-left corner pixel.

Table 10.3 and Table 10.4, together with the visual representation of Fig. 10.11, show some numerical results obtained by the proposed novel approach, and demonstrate that a methodology that exploits semantic segmentation along with a standard navigation approach based on the GNSS is a complete and reliable navigation throughout the whole row-based crop. The results obtained in the pear orchard (Table 10.3) are slightly worse than the vineyard ones; however, this effect may be due to the higher inter-row distance of the pear orchard than the vineyard. Eventually, the mean time of a single test is about 25 minutes, while the maximum velocity of the UGV is 0.5 m/s. All considered, the overall achieved performances, in terms of MAE and RMSE, are adequate for the used low-cost sensor setup and demonstrate the effectiveness of my approach.

10.4 Conclusion

I presented a novel, affordable algorithmic pipeline for autonomous navigation in row-based crops. My methodology provides a comprehensive solution that encompasses all navigation stages, from global to local path planning, utilizing only low-cost, low-range sensors. Moreover, the system efficiently addresses GNSS unreliability in the presence of dense vegetation and thick canopies, enabling the platform to navigate autonomously throughout all seasonal periods. Finally, the adopted domain generalization and optimization techniques significantly reduce



(a) Test n.3 in the vineyard scenario.

(b) Test n.2 in the orchard scenario.

Fig. 10.11 A visual representation of the obtained results. Both images feature the path followed by the UGV (red line), the provided global path (cyan x), the row waypoints (blue dots), and the crop (green dots).

the time and computational cost of training and inference of the deep neural network. Further work will aim to assess my proposed algorithmic pipeline on a more complex vehicle and introduce a collision avoidance algorithm in the segmentation-based control.

Bibliography

- [1] S. Cerrato *et al.*, “A deep learning driven algorithmic pipeline for autonomous navigation in row-based crops,” *IEEE Access*, 2024.
- [2] A. Navone, F. Romanelli, M. Ambrosio, M. Martini, S. Angarano, and M. Chiaberge, “Lavender autonomous navigation with semantic segmentation at the edge,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2023, pp. 280–291.
- [3] A. Navone, M. Martini, A. Ostuni, S. Angarano, and M. Chiaberge, “Autonomous navigation in rows of trees and high crops with deep semantic segmentation,” in *2023 European Conference on Mobile Robots (ECMR)*, IEEE, 2023, pp. 1–6.
- [4] M. S. N. Kabir *et al.*, “Performance comparison of single and multi-gnss receivers under agricultural fields in korea,” *Engineering in agriculture, environment and food*, vol. 9, no. 1, pp. 27–35, 2016.
- [5] S. Marden and M. Whitty, “Gps-free localisation and navigation of an unmanned ground vehicle for yield forecasting in a vineyard,” in *Recent Advances in Agricultural Robotics, International workshop collocated with the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, 2014.
- [6] A. Navone, M. Martini, M. Ambrosio, A. Ostuni, S. Angarano, and M. Chiaberge, “Gps-free autonomous navigation in cluttered tree rows with deep semantic segmentation,” *Robotics and Autonomous Systems*, vol. 183, p. 104 854, 2025.
- [7] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, “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.
- [8] O. Ly, H. Gimbert, G. Passault, and G. Baron, “A fully autonomous robot for putting posts for trellising vineyard with centimetric accuracy,” in *2015 IEEE International Conference on Autonomous Robot Systems and Competitions*, IEEE, 2015, pp. 44–49.
- [9] S. J. Moorehead, C. K. Wellington, B. J. Gilmore, and C. Vallespi, “Automating orchards: A system of autonomous tractors for orchard maintenance,” in *Proceedings of the IEEE international conference of intelligent robots and systems, workshop on agricultural robotics*, 2012.

- [10] S. Bonadies and S. A. Gadsden, “An overview of autonomous crop row navigation strategies for unmanned ground vehicles,” *Engineering in Agriculture, Environment and Food*, vol. 12, no. 1, pp. 24–31, 2019, ISSN: 1881-8366.
- [11] W. Winterhalter, F. Fleckenstein, C. Dornhege, and W. Burgard, “Localization for precision navigation in agricultural fields—beyond crop row following,” *Journal of Field Robotics*, vol. 38, no. 3, pp. 429–451, 2021.
- [12] S. Zaman, L. Comba, A. Biglia, D. R. Aimino, P. Barge, and P. Gay, “Cost-effective visual odometry system for vehicle motion control in agricultural environments,” *Computers and Electronics in Agriculture*, vol. 162, pp. 82–94, 2019.
- [13] I. Nevljudov, S. Novoselov, O. Sychova, and S. Tesliuk, “Development of the architecture of the base platform agricultural robot for determining the trajectory using the method of visual odometry,” in *2021 IEEE XVIIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, IEEE, 2021, pp. 64–68.
- [14] Y. Ma, W. Zhang, W. S. Qureshi, C. Gao, C. Zhang, and W. Li, “Autonomous navigation for a wolfberry picking robot using visual cues and fuzzy control,” *Information Processing in Agriculture*, vol. 8, no. 1, pp. 15–26, 2021, ISSN: 2214-3173.
- [15] 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.
- [16] P. Kim, B. Coltin, and H. J. Kim, “Low-drift visual odometry in structured environments by decoupling rotational and translational motion,” in *2018 IEEE international conference on Robotics and automation (ICRA)*, IEEE, 2018, pp. 7247–7253.
- [17] D. Anthony and C. Detweiler, “Uav localization in row crops,” *Journal of Field Robotics*, vol. 34, no. 7, pp. 1275–1296, 2017.
- [18] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [19] A. Kamilaris and F. X. Prenafeta-Boldú, “Deep learning in agriculture: A survey,” *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018.
- [20] D. Aggi, V. Mazzia, and M. 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.
- [21] D. Aggi, S. Cerrato, V. Mazzia, and M. Chiaberge, “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.
- [22] J. Zoto, M. A. Musci, A. Khalil, M. Chiaberge, and I. Aicardi, “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.

- [23] V. Mazzia, F. Salvetti, D. Aghi, and M. Chiaberge, “Deepway: A deep learning waypoint estimator for global path generation,” *Computers and Electronics in Agriculture*, vol. 184, p. 106 091, 2021.
- [24] S. Cerrato, D. Aghi, V. Mazzia, F. Salvetti, and M. Chiaberge, “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.
- [25] R. C. Coulter, “Implementation of the pure pursuit path tracking algorithm,” Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.
- [26] D. Misra, “Mish: A self regularized non-monotonic neural activation function,” *arxiv: 1908.08681*, vol. 4, p. 2, 2019.
- [27] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, “Cbam: Convolutional block attention module,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.
- [28] J. Matas, C. Galambos, and J. Kittler, “Robust detection of lines using the progressive probabilistic hough transform,” *Computer vision and image understanding*, vol. 78, no. 1, pp. 119–137, 2000.
- [29] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., “A density-based algorithm for discovering clusters in large spatial databases with noise.,” in *Kdd*, vol. 96, 1996, pp. 226–231.
- [30] P. Hu et al., “Real-time semantic segmentation with fast attention,” *IEEE Robotics and Automation Letters*, vol. 6, no. 1, pp. 263–270, 2020.
- [31] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, “Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation,” *arxiv: 2004.02147*, 2020.
- [32] A. Howard et al., “Searching for mobilenetv3,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.
- [33] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, “Rethinking atrous convolution for semantic image segmentation,” *arxiv: 1706.05587*, 2017.
- [34] D. Aghi, V. Mazzia, and M. Chiaberge, “Local motion planner for autonomous navigation in vineyards with a rgb-d camera-based algorithm and deep learning synergy,” *Machines*, vol. 8, no. 2, p. 27, 2020.
- [35] D. Fox, W. Burgard, and S. Thrun, “The dynamic window approach to collision avoidance,” *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.
- [36] M. Martini, S. Cerrato, F. Salvetti, S. Angarano, and M. Chiaberge, “Position-agnostic autonomous navigation in vineyards with deep reinforcement learning,” in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2022, pp. 477–484.
- [37] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*, PMLR, 2018, pp. 1861–1870.

- [39] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, “Improving sample efficiency in model-free reinforcement learning from images,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 10 674–10 681.
- [40] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.
- [41] M. 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.

Conclusion

And so, our journey into the world of efficient and robust AI for autonomous systems has come to an end. This thesis was not meant to be a comprehensive overview of such a broad research field. Instead, it aimed to provide the reader with a clear understanding of the primary motivations behind the research I conducted during my Ph.D., the fundamental background I drew upon, and the advancements I made to the state of the art. Before heading out, I will briefly reiterate my major contributions and outline a roadmap for future research.

Main Contributions

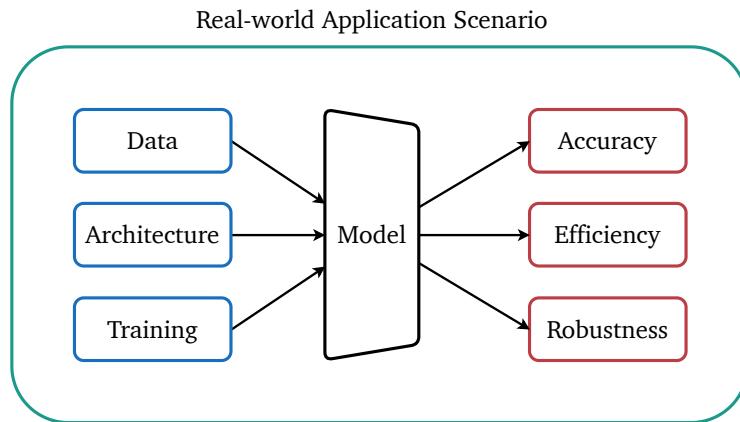


Fig. 10.12 The elements constituting a machine learning model (left) and its properties (right) finally find their purpose in solving complex tasks in unstructured application contexts.

At the beginning of this journey, we discovered that differentiable models are essentially constituted of data, architecture, and training. Then, we explored the

subfield of efficient and robust AI, encompassing model compression, generalization, and hardware devices, while also outlining some relevant application fields of this research. Finally, we reviewed some relevant research works I published during my Ph.D., demonstrating the benefits of studying efficient and robust methodologies for numerous real-world applications.

In particular, we considered the scheme in Fig. 10.12 and throughout the chapters, we delved deep into what makes a model accurate, efficient, and robust at the same time.

We studied **data**, starting from the observation that, while vast datasets are often available for general-enough tasks, the same cannot be said for unstructured, hard-to-model target problems. To overcome this obstacle, one could manually label thousands of samples, trying to cover most of the variance of the real distribution (including corner cases). However, this approach is often too expensive. Synthetic data is currently the most cost-effective alternative, although it inevitably creates a sim-to-real gap that may drastically compromise performance. In this scenario, I demonstrated how a thorough examination of the generalization problem (Chapter 7) can facilitate the development of innovative methods to bridge the accuracy gap of models trained on simulated data or in the presence of other distribution shifts (Chapter 8).

We also analyzed the **architecture**, recognizing that the number of parameters in state-of-the-art models has been heavily increasing over the years. This trend, enabled by the growing availability of computational power and data, has often overlooked the fact that many real-world applications still need low-latency responses and low power consumption. For this reason, I tried to bridge the gap between overparametrized state-of-the-art models and the constraints of real-time applications. I introduced several techniques that can unlock better performance at smaller scales, spanning architectural design choices (Chapter 4) and compression techniques (Chapter 6).

Furthermore, we investigated **training**, as the framing of the optimization problem heavily impacts the resulting models. Recent years have seen the rise of self-supervised learning as a way to leverage gigantic unlabeled datasets, spanning all possible domains, and condense them into large transformer models. Other training paradigms, such as knowledge distillation, can be used to transfer certain desired features between models. I applied self-supervised learning

to build a multi-task model for remote sensing (Chapter 9), proving that this training approach improves robustness against radiation-induced faults. I also proposed a novel knowledge distillation method to effectively compress super-resolution models, making them suitable for real-time tasks such as teleoperation (Chapter 5).

As a final, comprehensive demonstration, I reported an applied example that integrates all three elements (data, architecture, and training) to solve autonomous navigation in crops Section 10.2. By solving this unstructured task, I demonstrated the contribution that research in efficient and robust AI can have on real-world strategic problems.

Future Works

The insights and contributions presented in this thesis open up several promising directions for future research, combining the latest advancements in the field of AI to develop efficient and robust autonomous systems.

Data: A key open challenge remains the systematic integration of synthetic and real-world data. While this work has demonstrated that generalization-aware design can mitigate the sim-to-real gap, future research could focus on adaptive data generation, which involves procedurally generating synthetic samples based on real-world performance feedback. Moreover, integrating active learning or continual data refinement could allow models to selectively request or generate new samples that address their specific weaknesses, especially in dynamic or evolving environments. In general, I believe that true generalization does not come solely from increasing the dataset and model size, but is also feasible at minor scales through the correct induction biases.

Architecture: While I explored several efficient architectural designs and compression techniques, future work could further investigate neural architecture search (NAS) under resource constraints, particularly in multi-task or continual learning contexts. Moreover, vanilla transformers are not suitable for low-power, real-time computation due to their inefficient attention mechanism. There is significant potential in exploring hybrid architectures that combine the inductive biases of convolutional networks with the representational power of transformers,

explicitly optimized for deployment on edge hardware or radiation-hardened processors.

Training: Self-supervised learning and knowledge distillation proved effective in several applications, but many questions remain. One promising direction is the integration of robust self-supervised objectives that are aware of domain shifts or adversarial conditions. In the context of knowledge distillation, exploring teacher-student co-evolution or domain-aware distillation strategies could further improve generalization across unseen environments. Moreover, distillation still performs poorly when the difference in size between teacher and student is too large. Further study on the hidden mechanism of knowledge transfer may unlock new, effective strategies.

System-Level Integration. Finally, a broader engineering challenge lies in co-ordinating these advances at the system level. Future work should explore how data, architecture, and training can be jointly optimized in real-time systems that operate under conditions of uncertainty, limited supervision, and fault-proneness. That includes designing lifelong learning pipelines, robust deployment strategies, and full-stack optimizations (from model design to hardware scheduling) for integrated robotic systems.

In summary, the tools and ideas developed in this thesis lay the groundwork for more adaptive, resilient, and deployable AI. The following steps will involve closing the loop between learning and deployment, moving from robust models to robust systems that can continually adapt, learn, and perform in the complexities of the real world.

Foreword

Just a few more words before the curtains fall (I promise it will be brief). There are many topics I discussed only briefly, although I believe they may be paramount for the future advancement of AI research. I will list few of them here because I did not find a better spot, or did not have the time: parameter-efficient fine-tuning [1], the blessing of dimensionality [2], learning representations by compressing¹², autoregressive approaches [3], joint embedding predictive architectures

¹²Ted Chiang, “*ChatGPT is a Blurry JPEG of the Web*”, The New Yorker (2023)

[4], DINO [5], and SAM [6]. The same applies to other minor papers I worked on and the unpublished work I conducted over the last few months on compressing large foundation models (e.g., [7]) and applying them to robotic social navigation tasks (e.g., [8]). I also skimmed on many practical engineering aspects: training and serving large models is a considerable engineering effort that requires, among other things, distributed strategies, compilers, and advanced deployment techniques and frameworks. But this is a research work after all, and I did not want to bother my reviewers with more chapters about libraries, code, and hardware stuff.

Additionally, I must acknowledge that this thesis already feels somewhat dated. When I started my Ph.D., transformers were on the verge of taking over, but the pace at which new architectures and methods have been proposed has been astonishing over the last few years. The emergence of large language (and later multimodal) models has opened up new avenues for what a large number of simple computational elements can do. The knowledge of their inner workings is no longer a prerequisite, from prompt engineering to model chaining and agentic behaviours. It is remarkable and unexpected, and also provides (perhaps) “*the best impetus we’ve had in two thousand years to understand better just what the fundamental character and principles might be of that central feature of the human condition that is human language and the processes of thinking behind it*”¹³.

Nevertheless, I believe the research I conducted is more relevant than ever, even in the context of large AI agents. The growing need for AI has quickly raised concerns about the cost, power consumption, and environmental impact of training large models. Efficient AI models lower operational costs by reducing computational demands and infrastructure overhead. That democratizes access to advanced AI technologies, making them viable for startups and smaller organizations, while enabling scalability for larger enterprises without excessive resource expenditure. Moreover, regulations like the EU’s AI Act¹⁴ demand robust models that align with ethical standards and environmental considerations. Efficient and reliable AI systems are better equipped to meet these requirements while maintaining performance. Investing in research on efficiency and robustness enables AI to scale sustainably, adapt to evolving challenges, and deliver

¹³Stephen Wolfram, “*What Is ChatGPT Doing... and Why Does It Work?*” (2023)

¹⁴“*The EU Artificial Intelligence Act*”, Official Journal of the European Union (2024)

transformative value across industries. I hope to be part of this transformation and that I have transmitted its importance to the readers of this work.

This thesis has a companion webpage¹⁵, where I plan to publish additional findings, considerations, and updates that relate to the topics addressed here. I hope you appreciated the journey! Please do not hesitate to contact me with comments, suggestions, or feedback on the thesis.

¹⁵simeoneangarano.github.io

Bibliography

- [1] E. J. Hu *et al.*, “Lora: Low-rank adaptation of large language models.” *ICLR*, vol. 1, no. 2, p. 3, 2022.
- [2] A. N. Gorban and I. Y. Tyukin, “Blessing of dimensionality: Mathematical foundations of the statistical physics of data,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 376, no. 2118, p. 20 170 237, 2018.
- [3] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [4] A. Bardes *et al.*, “Revisiting feature prediction for learning visual representations from video,” *arxiv: 2404.08471*, 2024.
- [5] M. Oquab *et al.*, *Dinov2: Learning robust visual features without supervision*, 2023.
- [6] N. Ravi *et al.*, “Sam 2: Segment anything in images and videos,” *arxiv: 2408.00714*, 2024.
- [7] J. Shang *et al.*, “Theia: Distilling diverse vision foundation models for robot learning,” in *8th Annual Conference on Robot Learning*, 2024.
- [8] A.-C. Cheng *et al.*, “Navila: Legged robot vision-language-action model for navigation,” *arxiv: 2412.04453*, 2024.