Politecnico di Torino

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

Doctoral Dissertation
Doctoral Program in Electrical, Electronics and Communications Engineering
(35.th cycle)

Machine Learning Algorithms for Robotic Navigation and Perception and Embedded Implementation Techniques

Francesco Salvetti

* * * * *

Supervisors
Prof. Marcello Chiaberge, Supervisor
Prof. Fabrizio Lamberti, Co-supervisor

Politecnico di Torino
2023

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.

. . . . . .

Francesco Salvetti
Turin, 2023

Summary

In the latest years, a lot of academic and industrial research interest has been focused on the world of Artificial Intelligence. In particular, the rise of Machine Learning and Deep Learning has revolutionized the AI world, setting a new standard for using data to solve a great number of tasks. Fields such as Computer Vision, Pattern Recognition, Speech Recognition, Natural Language Understanding, and many others have taken the greatest advantages from these new algorithmic paradigms, reaching new state-of-the-art results that outperformed classical methodologies. In this context, also the robotic field can benefit from the adoption of these modern AI approaches, in order to bring machine intelligence and autonomous agents to a higher level. Among the different robotic categories, service robots, defined as robots that perform useful tasks for humans or equipment, are rapidly evolving. Service robotic prototypes can be applied to a number of different contexts, ranging from smart agriculture to elderly assistance.

This thesis presents different methodologies that merge the AI world with service robotics. Deep learning solutions are proposed to tackle several tasks from the sensing to the application levels. An algorithmic pipeline for autonomous service robotic navigation in smart agriculture contexts is presented, adopting multiple deep learning methods in different levels of the stack. Moreover, learning-based methodologies are adopted to solve perception tasks targeting different applications, such as object detection for precision agriculture, indoor localization, human action recognition from visual data, and efficient video streaming for remote robotic teleoperation. All the proposed methods are explained from a theoretical point of view and thoroughly tested with precise experimentation setup in order to prove their effectiveness in solving the target tasks. State-of-the-art related works present in the literature are deeply analyzed and used to build comparisons with the proposed approaches. Moreover, great care is also given to the embedded implementation of the algorithms, especially when they are needed to run directly on the robots. Several approaches are used in order to increase the algorithmic inference efficiency and different hardware platforms are analyzed in terms of their ability to run machine learning methods in real-time. Collectively, this dissertation constitutes a step toward the full integration between robotics and AI and paves the way for future research in this direction.

Acknowledgements

I want to thank all the people with whom I collaborated during these years at Politecnico di Torino. In particular, a big thank goes to my PhD tutor Prof. Marcello Chiaberge, for giving me the opportunity to work with the PIC4SeR center and for his support throughout the entire development of this work. My gratitude goes also to all the people I met during these years, in particular the other PhD students that shared with me the time and efforts needed to achieve our goals and all the researcher that helped me work on the countless projects we developed. I must express all my gratitude to my family for being always present and to all my friends: you supported and encouraged me during these years. Thanks to Valeria for being always by my side and having shared with me most of the happy and difficult moments of these years. Finally, I want to dedicate this work to Rachele that came into life during the last period of writing this thesis.

Contents

Part I

Fundamentals

Chapter 1

Introduction

The International Organization for Standardization (ISO) defines, in the standard 8373 [107], a *robot* as a "*programmed actuated mechanism with a degree of autonomy to perform locomotion, manipulation or positioning*". Among the different categories, a *service robot* is defined as a "*robot in personal use or professional use that performs useful tasks for humans or equipment*". The tasks performed by a service robot include "*handling or serving of items, transportation, physical support, providing guidance or information, inspection, surveillance, cooking and food handling, grooming, and cleaning*". This class of robots is therefore characterized by specialized machines that provide several services to human beings, and it does not include applications such as industrial automation. Service robots can be employed in a number of different operation areas [20, 74] such as household management [54], smart cities [160], precision agriculture [186], defense [191], construction [212], healthcare [95], leisure and recreation [244], logistics [261]. That rich number of applications has led in recent years to a great interest in literature to service robotics, with several publications focusing on sensing, navigational, mechanical, control, and software methodologies.

One of the key aspects of service robots is the ability to carry over the assigned tasks autonomously. For the great majority of applications, where mobile robots are adopted, this autonomy requirement includes the ability to navigate in the environment, understand the surrounding conditions, recognize objects, obstacles and people, reach the area of action, and perform the required task. All these operations must be performed by the robotic platform with limited human intervention and complying with assigned time requirements. Therefore, research and development efforts must focus on making machines able to quickly and efficiently process sensing information, plan navigational trajectories and operations, act accordingly in the given environment and respond effectively to sudden changes in the working conditions. This collective software requirement can be summarized as giving machines a certain degree of intelligence. Artificial Intelligence (AI) is indeed an "*interdisciplinary field, usually regarded as a branch of computer science,*

Figure 1.1: The Clearpath Robotics Jackal UGV while autonomously navigating in an agricultural environment

dealing with models and systems for the performance of functions generally associated with human intelligence, such as reasoning and learning" [106]. Developments in AI, and in particular in Machine Learning (ML) methodologies, that give the ability to extract patterns and learn from data, have brought a high number of contributions in this field, proposing novel approaches to solve a high number of robotic-related tasks [170]. More recently, the Deep Learning (DL) [133] paradigm, which focuses on designing multi-layer neural networks, has been proven to be very effective in learning patterns from great collections of data, becoming the state-of-the-art (SOTA) standard method to solve computer vision, speech, sensing, and perception tasks.

In this dissertation, we leverage Machine Learning and Deep Learning methods to address service robotic tasks fundamental to build autonomous navigation and perception stacks. As an example, one of the key methods proposed in this dissertation focuses on the development of an algorithmic pipeline to tackle autonomous navigation in agricultural contexts, that adopts Deep Learning methodologies at different levels: first to refine satellite images to synthetically increase the spatial resolution, then to propose waypoint candidates to generate a global path, finally to locally control the robot movements by processing visual data onboard in the target environment. Fig. 1.1 shows an example of an unmanned ground vehicle (UGV) performing an autonomous navigation task, for service robotic agricultural applications. Together with the other proposed methodologies, this approach can represent

a step forward in leveraging cutting-edge advancements in the Deep Learning field in order to solve complex applied tasks in the service robotic context.

1.1 Contributions

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

- analyze Machine Learning and Deep Learning state-of-the-art methodologies for computer vision and pattern recognition applicable to solve robotic tasks
- propose a novel multi-step algorithmic pipeline for robotic path planning and navigation in an agricultural context that combines satellite images processing and a global waypoint-based path planner with different visual-based local navigation controls
- present different learning-based methods to solve robotic perception tasks such as object detection, super-resolution, human action recognition, and error mitigation
- apply different levels of optimization strategies to optimize the inference efficiency of neural networks with minimal loss of accuracy
- deploy Deep Learning-based models on several embedded platforms and benchmark their real-time performance

Most of the works presented in this thesis have been published on peer-reviewed journals [172, 214, 10, 173, 33, 11, 171] and conferences [32, 213, 12, 167], that gather state-of-the-art researches in the Machine Learning and Robotics fields. This dissertation can be seen as a summary of my work performed at PIC4SeR, the Politecnico di Torino Interdepartmental Center for Service Robotics, in a number of different projects, collaborating with other PhD students and researches. I contributed to all these works both in the methodological and in the experimental phases of the different projects. During these years as a PhD student, I also contributed to more general artificial intelligence and computer vision topics, not strictly related to the scope of the thesis. In particular, we proposed an efficient method for image classification with Capsule networks [174], we studied how domain generalization is influenced by backbone pre-training [9], we designed a DL-based robotic arm system playing Jenga [163], a reinforcement learning based point-to-point navigation system with ultra-wideband integration [235] and a person-following method based on computer vision [26]. All these works have collectively contributed to my experience in the AI and robotics fields and therefore represent important steps in my doctoral career.

1.2 How to read this work

This thesis is divided into three main parts. Part I presents this introduction, as well as the theoretical background on Machine Learning and Deep Learning (DL) for Computer Vision applications in Chapter 2, which gives a practical overview of clustering methods and neural networks. Moreover, Section 2.4 gives a brief presentation on methods useful for edge execution of neural networks, particularly relevant for onboard deployment of AI algorithms in robotic applications. All the theoretical concepts presented in Chapter 2 represent fundamental related works for the successive parts of the thesis, and can therefore be referenced for their full and complete understanding.

Part II is focused on robotic navigation and presents a Deep Learning based algorithmic pipeline for autonomous navigation in an agricultural context. Chapter 3 presents the main ideas behind the proposed pipeline, as well as its main algorithmic blocks, also referencing some early in-field experimental results. On the other hand, Chapters 4 and 5 enter more in detail on two specific modules of the pipeline based on DL, that focus on satellite remote sensing and path generation.

Finally, Part III deals with robotic perception, presenting DL-based methodologies related to data acquisition and sensor processing for mobile robotic platforms. In particular, Chapter 6 proposes an apple identification system based on object detection that can represent a valuable example of a perception task performed by a robot within the navigational context presented in Part II. Chapter 7 focuses on DL-based error compensation of range measurements with ultra-wideband sensors, that can be used in localization tasks. Chapter 8 presents a model for human action recognition suitable for real-time onboard execution, while Chapter 9 proposes a method based on efficient real-time Super-Resolution to tackle image transmission in low-bandwidth scenarios for robotic teleoperation.

The last Chapter 10 draws some conclusions, as well as indicates possible directions for future research in the field.

Chapter 2

Basics of Machine Learning

According to ISO 2382, Machine Learning (ML) can be defined as the "*process by which a functional unit improves its performance by acquiring new knowledge or skills [...]*" [106]. In particular, ML techniques are focused on recognizing patterns and extracting knowledge from big amounts of data. This chapter focuses on presenting basic concepts about ML algorithms related to this dissertation. The concepts defined in the following sections are the foundations on which the works presented in Parts II and III have been developed and can therefore be a useful reference for the reader.

Creating a taxonomy of ML techniques is not an easy task. Algorithms can be subdivided into different categories, depending on the characteristics under attention. A basic tentative taxonomy can depend on the learning strategy adopted. There are multiple ways to extract knowledge from data, but, in general, each ML algorithm can fall under one of the following groups:

- **supervised learning**: these techniques try to learn from the experience of several *input-output* pairs, trying to predict the expected outcome for each input sample and to minimize the error with respect to given *labels*;
- **unsupervised learning**: these techniques try to recognize patterns and groups from a big set of *unlabeled* data; an example of unsupervised methods are the clustering algorithms, that focus on separate data into coherent groups of samples;
- **semi-supervised learning**: the midpoint of the previous two groups; usually refer to methods that can rely only on a little fraction of labeled data, and a big quantity of unlabeled data;
- **self-supervised learning**: these techniques are usually adopted to learn vectorial representations of the input data, by trying to reconstruct parts of the input by extracting meaningful features from it; in this case, the labels are directly derived by the input data itself and do not need manual annotation;

- **reinforcement learning**: methods that rely on learning by *trial and error*, analyzing the feedback, or reward, of the environment received by an agent, following a specific action in a certain state; these algorithms do not required labeled data, but usually are based on the ability to simulate an environment and its evolution following a set of agent actions.

Another possible categorization can depend on the task to be solved given a certain set of data samples:

- **classification** aims at finding a discrete assignment, called *class*, for each sample in the dataset; usually, classification methods learn the probability distribution of belonging to each class; the classification is defined as *binary* if there is just a positive and negative class and the algorithm learns a one-dimensional distribution; in *multi-class* classification, there a number of possible classes and each sample has just a single target; in *multi-label* classification, each sample can be assigned to multiple categories;
- **regression** aims at predicting a continuous output for each input sample; regression algorithms should learn a generic non-linear mapping between the inputs and the outputs;
- **clustering** methods aim at separating samples into coherent groups, called *clusters*; differently from classification, in clustering there is not a unique target assignment, and there is no guarantee that the final groups have a strong semantic meaning;
- **representation learning** aims at learning a dense vectorial representation for the input data, by projecting it into a new n-dimensional *latent* space; this space usually must respect a set of constraints, for which similar representations must have a close semantic relation, while dissimilar representations should be semantically uncorrelated.

The following sections present some ML techniques relevant to this work. Sec 2.1 focuses on two widely used clustering methods, while Section 2.2 provides basic concepts on Artificial Neural Networks. Finally, Section 2.4 presents some techniques used for the Edge execution of ML algorithms on specific hardware accelerators.

2.1 Clustering

Clustering methods aim at separating samples into groups according to a certain spatial criterion. Unlike classification, this is entirely done in an unsupervised way, and the cluster assignment criterion is learned not from external given labels, but from internal data patterns. There are a really huge number of clustering approaches in literature [109, 218, 65]. In this chapter, we focus on the two methods used in the next chapters of this thesis: K-means and DBSCAN.

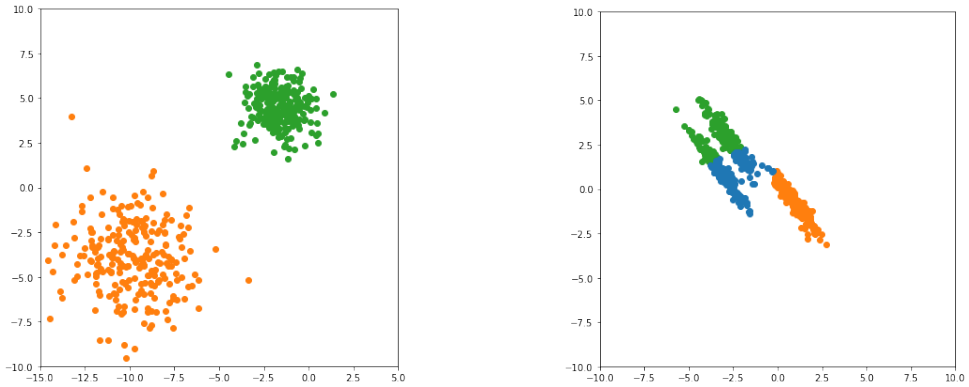


Figure 2.1: Example of K-means clustering: 2 centroids with Gaussian data samples (left) and 3 centroids with anisotropic samples (right). K-means, being based on Euclidean distance from a cluster center, does not perform well in the case of distributions with non-equal axis variances.

2.1.1 K-means

The K-means [149] method consists in finding k centroids that serve as mean for the data clusters to be found. For this method, the number of clusters must be decided in advance and thus represents a strong assumption on the nature of the data to be clustered. The algorithm iteratively tries to refine an initial estimation of the clusters until it converges to a stable solution. Given n data samples, denoted as x_j, we can define the i^{th} centroid at iteration t as $c_i^{(t)}$ and the set of points belonging to cluster i as $S_i^{(t)}$. The standard K-means algorithm is based on the following steps:

1. randomly initialize the k centroids, $c_1^{(0)}, c_2^{(0)}, \ldots, c_k^{(0)}$
2. assign each data sample to the cluster of the closest centroid, according to the Euclidean distance:
$$S_i^{(t)} = \left\{ x_j \mid i = \min_p \|x_j - c_p^{(t)}\|_2 \ , \ \forall j \in 1, ..., n \right\} \tag{2.1}$$
3. update the centroids with the mean of the cluster samples:
$$c_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \tag{2.2}$$
4. repeat steps 2 and 3 until convergence is reached

Steps 2 and 3 can be seen as a special case of the expectation-maximization (EM) algorithm [49], used to estimate parameters of statistical models by maximizing the log-likelihood. Therefore, K-means can be considered as a Gaussian

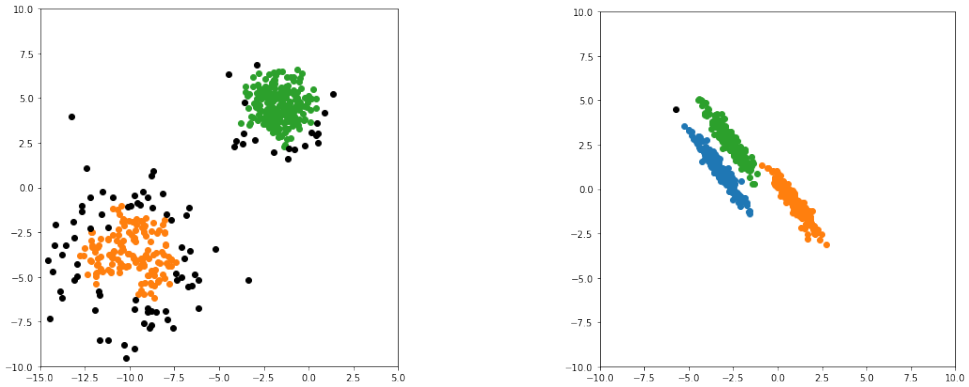


Figure 2.2: Example of DBSCAN clustering: 2 clusters with Gaussian data samples (left) and 3 clusters with anisotropic samples (right). Being density-based, DBSCAN is more suitable for clustering non-isotropic data, but suffers from a high number of unclustered points (outliers, in black), in the case of clusters with highly different densities.

Mixture Model [201], with the exception that it is only using sample means and not sample covariance matrices. For this reason, K-means equiprobability boundaries are always circular, and cannot fit well elliptical or anisotropic data, as shown in Fig. 2.1.

2.1.2 DBSCAN

DBSCAN [63] is a density-based clustering algorithm, that, instead of relying on the distance between a cluster center, computes all the relative distances between points and clusters those that are more closed together. Differently from K-means, the number of clusters k is not defined *a priori*, but depends on the nature of the data and on the two main hyper-parameters of the algorithm: ε, the maximum distance at which two points are considered as neighbors, and $n_{\min}$, the minimum number of points required to form a dense region.

The algorithm visits all the n data points x_j and classifies each in one of the following categories:

- *core point*: if $|\{x_p \mid \|x_j - x_p\|_2 \leq \varepsilon\ ,\ \forall j \in 1, ..., n\}| \geq n_{\min}$, i.e. there are at least $n_{\min}$ points within distance ε from x_j;
- *reachable point*: if $\exists\, x_p \mid \|x_j - x_p\|_2 \leq \varepsilon \wedge x_p$ is a *core point* , i.e. x_j is within distance ε from a core point;
- *outliers*: all the non-reachable points.

If x_j is a core point, it forms a cluster together with all the core and non-core points that are reachable from it. All the non-core reachable points form the edge of

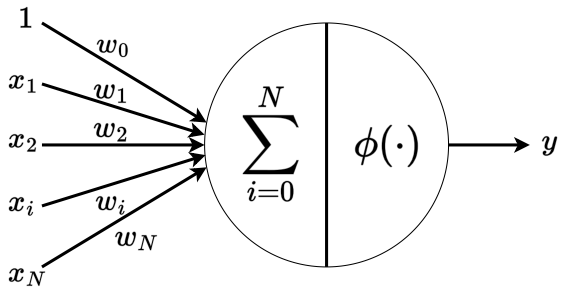


Figure 2.3: Artificial neuron model.

the cluster and cannot be used to reach further points. The output of DBSCAN is therefore a variable number of clusters, depending on the chosen hyper-parameters, and a set of unclustered outliers. The main advantages of this clustering method are the ability to form arbitrary-shaped clusters and its robustness to outliers, as opposed to K-means. However, it suffers when data present clusters with highly different densities, since it only admits one $(\varepsilon, n_{\text{min}})$ combination, as shown in Fig. 2.2.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) currently represent the state-of-the-art in a huge number of data processing applications, in particular for computer vision, speech recognition, natural language processing, pattern recognition, and many more [2, 253, 286, 188, 85, 37, 159]. Starting from some pioneering works published in the past century [175, 206, 215], ANN research gained a lot of momentum, especially after the backpropagation algorithm [209, 210] was proposed to efficiently train complex networks. With the advent of the so-called Deep Learning (DL) [133, 75], together with the great technological development and the availability of a big quantity of labeled data [50], ANN became the SOTA approach in a number of applications. In this section, the basics of ANN models are presented, followed by a brief introduction to current popular architectures, such as Convolutional Neural Networks [132] and Transformers [250].

2.2.1 Artificial Neuron

An ANN is a mathematical structure that models a generic function $\boldsymbol{y} = f(\boldsymbol{x})$ between some input variable $\boldsymbol{x}$ and some output variable $\boldsymbol{y}$. The basic unit of an ANN is the artificial neuron, which approximates the basic behavior of biological neurons. A neuron is excited if the inputs it receives from other neurons are above a

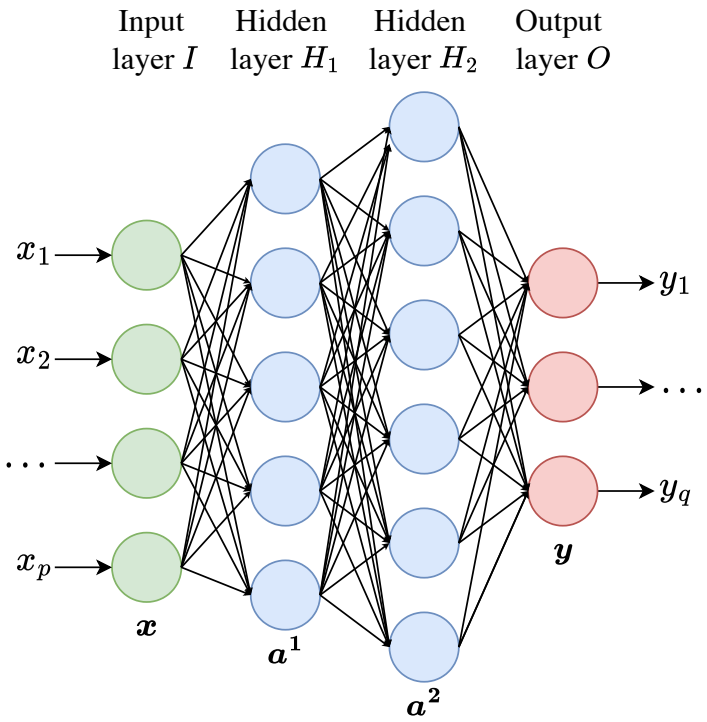


Figure 2.4: Multi-layer Perceptron with p inputs, two hidden layers and q outputs. The signal flows from left to right. Biases are omitted for simplicity.

certain level. Mathematically, this is implemented as a weighted sum of the inputs $\boldsymbol{x}$, followed by an activation function $\phi(\cdot)$:

$$y = \phi(\boldsymbol{w}\boldsymbol{x} + b) = \phi(\sum_{i=1}^{N} w_i x_i + b) = \phi(z) \tag{2.3}$$

where $\boldsymbol{w}$ is the vector of the weights and b is the neuron bias. In general, an artificial neuron has $N + 1$ parameters, where N is the number of inputs. Fig. 2.3 shows a representation of the artificial neuron model, where the bias term b is simply treated as an additional weight w_0 with unitary input.

2.2.2 Multi-layer Perceptron

Multiple neurons can be combined together to create a layer. Feed-forward Neural Networks (FNN), the most common type of ANN, are made of a succession of multiple layers of neurons with the signal flowing from the first layer toward the last one. A layer in which all the neurons take as input all the neurons of the preceding layer is defined as a Fully Connected (FC) or Dense layer. All the weights and biases in a FC layer with M neurons can be aggregated into a matrix

$\boldsymbol{W} = [\boldsymbol{w_1}, \boldsymbol{w_2}, \ldots, \boldsymbol{w_M}]^{\mathrm{T}}$ and a vector $\boldsymbol{b} = (b_1, b_2, \ldots, b_M)$, respectively. Thus the layer output vector $\boldsymbol{y}$ can be computed with matrix multiplication as follows:

$$\boldsymbol{y} = \phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \phi(\boldsymbol{z}) \tag{2.4}$$

where the activation function $\phi(\cdot)$ is applied pointwisely to all the layer neurons.

This idea of combining multiple Dense layers to create a single model is at the core of the Multi-layer Perceptron (MLP) [210], an evolution of the first proposed Perceptron [206]. This model is made of an input layer, that stores the input vector $\boldsymbol{x}$, followed by a number of hidden layers and a final output layer, that computes the model output vector $\boldsymbol{y}$. The intermediate layers are defined as *hidden* since they are not visible from outside the model. Figure 2.4 shows an example of an MLP with two hidden layers.

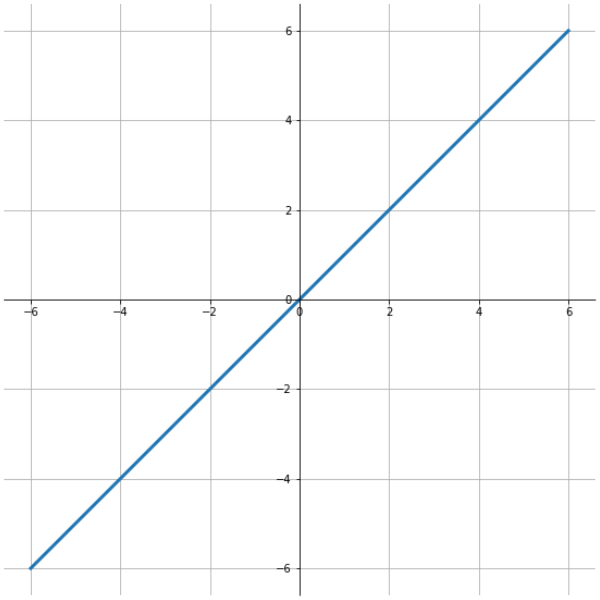
It has been proved that this kind of architecture is theoretically able to model any non-linear function, regardless of the number of inputs and outputs [96], given a proper number of hidden units. However, this does not mean that MLPs are suitable to solve any ML task since properly optimizing them, given a generic noisy dataset, is a complex task, and will probably result in an under-performing solution. For this reason, other architectures have been investigated in the literature.

2.2.3 Activation functions

Given a generic layer of neurons, a wide number of different functions can be used as activation functions $\phi(\cdot)$, depending on a number of factors, such as the linearity, the output domain, the computation speed, the level of sparsity and others. A list of relevant activations related to the present dissertation is the following:

Linear

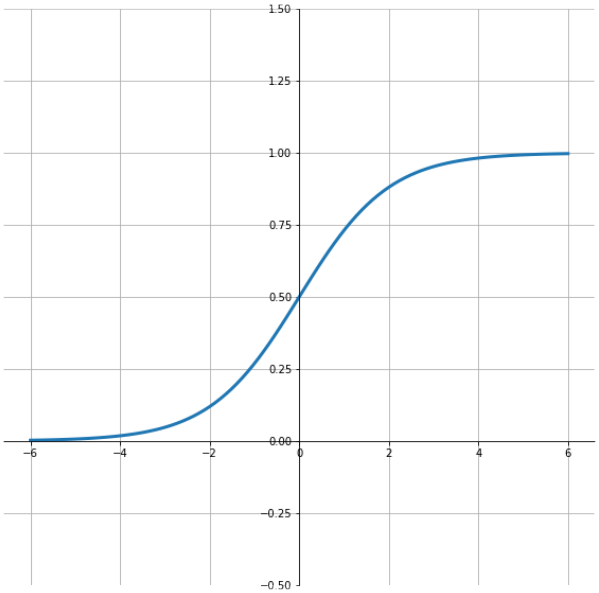
$$\phi(z) = z \tag{2.5}$$



It does not add any non-linearity; mainly used for regression tasks in which the output domain is unbounded.

Sigmoid

$$\phi(z) = \frac{1}{1 + e^{-x}} \tag{2.6}$$



Derived by logistic regression; bounded between 0 and 1; mainly used to model the probability of belonging to a class in binary or multi-label classification.

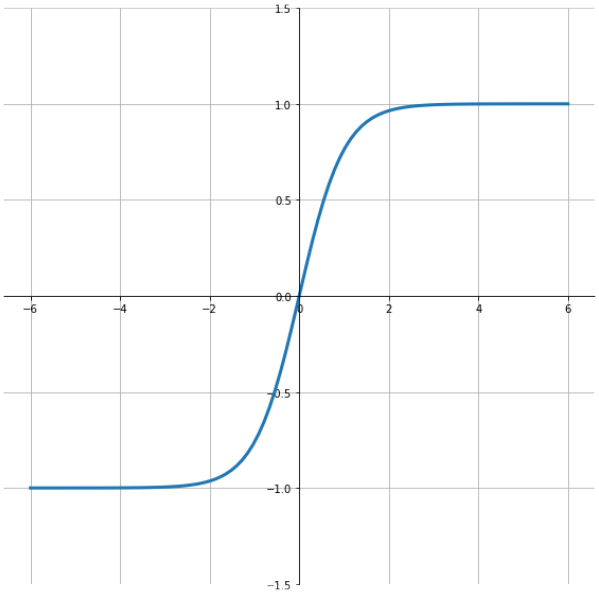
Softmax

$$\phi(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \quad , \quad \sum_{j=1}^{M} \phi(z_j) = \sum_{j=1}^{M} \frac{e^{z_j}}{\sum_k e^{z_k}} = \frac{\sum_j e^{z_j}}{\sum_k e^{z_k}} = 1 \tag{2.7}$$

A generalization of the sigmoid activation over a layer of multiple neurons; bounded between 0 and 1; used to model probability distributions for multi-class classification. Given a layer with M neurons, Eq. 2.7 is applied to all the neurons z_j. This activation normalizes the output space of the layer, ensuring that the sum of the outputs is always 1.

Tanh

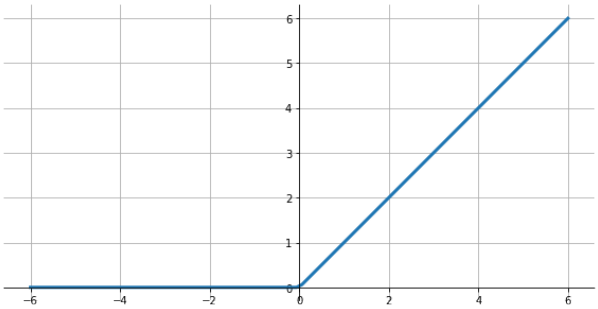
$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.8}$$



Bounded between -1 and 1; mainly used to model relative continuous values or as activation in hidden layers.

Rectified Linear Unit (ReLU) [180, 72]

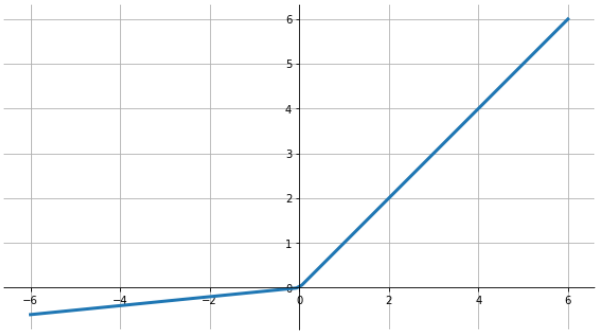
$$\phi(z) = z^{+} = \max(0, z) \tag{2.9}$$



The positive part of its arguments; when the $z \leq 0$ the unit is said to be inactive and the output is 0, for $z > 0$, the unit is linear; very common as hidden layers activation; very fast to compute; it can lead to the *dead neuron* effect when a unit always outputs zero for all the possible inputs.

Leaky ReLU [155, 269] and Parametric ReLU (PRelu) [88, 269]

$$\phi(z) = \max(\alpha z, z) \tag{2.10}$$

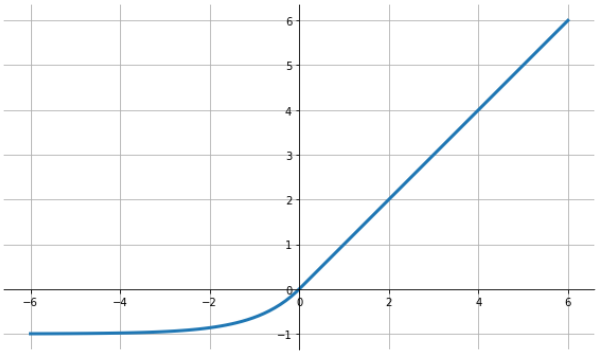


Variants of ReLU to avoid the *dead neuron* effect and ensure gradient flow for negative values. For Leaky ReLU, α is predetermined; for PReLU, α is considered a parameter of the network and learned together with the weights and the biases.

Continuous variants of ReLU Several activations have been proposed in the literature in order to ensure the continuity of the function for $z = 0$ and have a smooth transition to the inactive range:

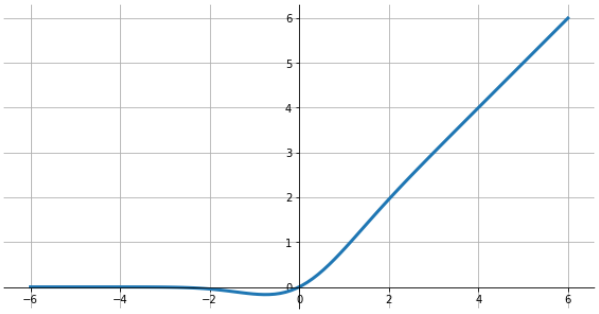
- Exponential Linear Unit (ELU) [43]

$$\phi(z) = \begin{cases} \alpha(e^z - 1) & \text{if } z \leq 0 \\ z & \text{otherwise} \end{cases} \tag{2.11}$$



- Gaussian Error Linear Unit (GELU) [90]

$$\begin{aligned}\phi(z) &= z\Phi(z) = zP(Z \leq z) \\ &= \frac{1}{2}\Big[1 + \text{erf}(z/\sqrt{2})\Big]\end{aligned}$$

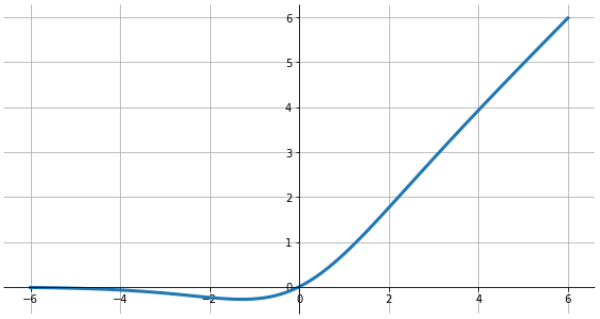


(2.12)

where $Z \sim \mathcal{N}(0, 1)$, $\Phi(z)$ is the standard Gaussian cumulative distribution function, $\text{erf}(\cdot)$ is the Gaussian error function.

- Swish [196]

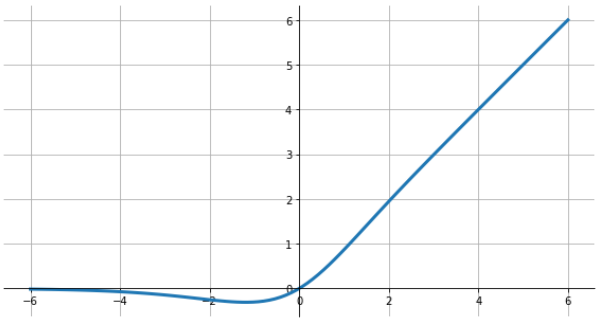
$$\phi(z) = z \cdot \text{sigmoid}(z)$$



(2.13)

- Mish [176]

$$\phi(z) = z \cdot \tanh(\text{softplus}(z))$$



(2.14)

2.2.4 Training an Artificial Neural Network

An Artificial Neural Network learns by finding its optimal parameters to solve the target task. For example, in a generic MLP model, the set of parameters $\boldsymbol{p}$ is the collection of all the weights and all the biases of the model neurons. To tune these parameters, supervised or self-supervised approaches are usually adopted. Independently by the type of learning process, how well the task to be performed is achieved is expressed by a certain loss function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{p})$. This function maps the input $\boldsymbol{x}$ to a negative estimation of the model performance, given a certain set of network parameters $\boldsymbol{p}$. The learning process is therefore represented as the minimization of the loss for all the inputs. The main difference between supervised and self-supervised methods lies in how this loss is computed, i.e. computing an error with respect to given labels (supervised) or using the input data itself to derive the target output (self-supervised). For all the following sections, we will refer to a supervised condition, but the same approach holds for a self-supervised problem since the only difference lies in the loss computation. Moreover, we will refer to a generic loss function $\mathcal{L}$, that represents the cumulative loss for all the available inputs $\boldsymbol{x}$ in the considered dataset, given the parameters $\boldsymbol{p}$).

For binary classification tasks, binary cross-entropy is usually adopted as loss, in combination with sigmoid activation (Eq. 2.6). Given a label y and the model prediction $\hat{y}$, the loss is computed as:

$$\mathcal{L} = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \tag{2.15}$$

Instead, in case of K classes, categorical cross-entropy, together with softmax (Eq. 2.7), is mainly adopted:

$$\mathcal{L} = -\sum_{k=1}^{K} y_k \log(\hat{y}_k) \tag{2.16}$$

On the other hand, for regression tasks, the most adopted loss is the ℓ-norm of the error. Given a d-dimensional label $\boldsymbol{y}$ and its network estimation $\hat{\boldsymbol{y}}$, the loss is computed as:

$$\mathcal{L} = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_\ell = \sum_{j=1}^{d} \left(|y_j - \hat{y}_j|^\ell \right)^{1/\ell} \tag{2.17}$$

with the special cases of Mean squared error (MSE) with $\ell = 2$ and Mean absolute error (MAE) with $\ell = 1$.

Gradient Descent

Given a generic ANN with parameters $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$, where n is the total number of parameters, we want to optimize the corresponding loss function $\mathcal{L}$, in

order to find its global minimum, that represents the best model to solve the target task. To do this, we want to iteratively update the parameters in order to slowly converge toward the optimum. Gradient descent (GD) is an algorithm that allows converging towards the minimum of a generic high-dimensional function, following the negative direction of the gradients. If we consider a generic small variation of the parameters $\boldsymbol{p}$, we can approximate the corresponding change in the loss function as follows:

$$\Delta\mathcal{L} \approx \frac{\partial\mathcal{L}}{\partial p_1}\Delta p_1 + \frac{\partial\mathcal{L}}{\partial p_2}\Delta p_2 + ... + \frac{\partial\mathcal{L}}{\partial p_n}\Delta p_n = \nabla\mathcal{L}\cdot\Delta\boldsymbol{p} \tag{2.18}$$

where $\nabla\mathcal{L} = \left(\frac{\partial\mathcal{L}}{\partial p_1}, \frac{\partial\mathcal{L}}{\partial p_2}, ..., \frac{\partial\mathcal{L}}{\partial p_n}\right)$ is the vector of the gradients of the loss function with respect to the parameters at the current point. In order to decrease the loss function, we should look for an update $\Delta\boldsymbol{p}$ such that

$$\nabla\mathcal{L}\cdot\Delta\boldsymbol{p} < 0 \tag{2.19}$$

Thus, a possible good choice is:

$$\Delta\boldsymbol{p} = -\eta\,\nabla\mathcal{L} \qquad \Rightarrow \quad \nabla\mathcal{L}\cdot\Delta\boldsymbol{p} = -\eta\,\|\nabla\mathcal{L}\|^2 < 0 \quad \text{if} \;\; \eta > 0 \tag{2.20}$$

This means that if move in the direction of the negative gradient, we reduce the loss. Equation 2.20 is called gradient descent and, when used to compute the parameters update, is referred to as an *optimizer*. The quantity η is called *learning rate* and should always be positive. The choice of a good learning rate is vital, since a too small value causes a really slow convergence or confinement in a poor local minimum; while a too large value can cause unstable behavior and divergence. Usually, the learning rate is decreased during the training process when the loss tends to stagnate, in order to allow a more fine search for the global minimum. To take into consideration these learning rate variations, specific algorithms are adopted, called learning rate *schedulers*, in order to tune the η value according to predetermined rules or functions.

Due to practical limitations, the computation is usually not performed for all the input samples at the same time, but we iteratively update the parameters using only a subset of the input data, called a *batch*. When we only use one sample at a time, GD gets the name of stochastic gradient descent (SGD). On the other hand, when we use a generic batch size higher than 1, we refer to it as mini-batch gradient descent. In almost all applications, optimization is currently performed by computing mini-batch gradients. A complete pass over the full dataset over subsequent optimization steps is called training *epoch*. This process is usually repeated for multiple epochs until convergence is reached at a minimum loss point.

Backpropagation

To apply GD we have to compute the partial derivatives of $\mathcal{L}$ with respect to each of the n parameters. This operation would require a big computational effort if we have many layers with a high number of neurons, but the backpropagation algorithm allows to speed up the process. This procedure was applied to train an ANN for the first time in the 1980s [210] and is based on the chain rule of derivatives to efficiently reuse computations performed in a certain layer to obtain the gradients of previous layers.

If we consider a generic neuron layer of the last layer of the network, with m inputs, the output is obtained with Eq. 2.3:

$$y = \phi(\boldsymbol{w}\boldsymbol{x} + b) = \phi(\sum_{i=1}^{m} w_i x_i + b) \tag{2.21}$$

To apply GD, we should first compute the derivative δ of the loss with respect to the activation function defined as $\delta_1 = \frac{\partial \mathcal{L}}{\partial \phi}$. Then, applying the chain rule of derivatives, we can obtain the derivatives with respect to the parameters:

$$\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial \phi}\frac{\partial \phi}{\partial w_i} = \delta_1 \frac{\partial \phi}{\partial w_i} \qquad \frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial \phi}\frac{\partial \phi}{\partial b} = \delta_1 \frac{\partial \phi}{\partial b} \tag{2.22}$$

If we have another layer before with activation $\psi(\cdot)$, we can get the derivative of the loss with respect to ψ as :

$$\delta_2 = \frac{\partial \mathcal{L}}{\partial \psi} = \frac{\partial \mathcal{L}}{\partial \phi}\frac{\partial \phi}{\partial \psi} = \delta_1 \frac{\partial \phi}{\partial \psi} \tag{2.23}$$

This δ_2 can then be used to get derivatives with respect to this layer's parameters. Thus, we can use the computation of δ_1 performed for the last layer also for the derivatives of the previous layer. Iterating this approach for all the parameters in all the layers, also for more complex networks, the complete backpropagation algorithm can be derived.

Other optimizers

Together with the plain GD optimizer (Eq. 2.20), several other algorithms have been proposed in the literature for computing the parameters update, given the gradients of the loss [208].

Momentum [193] This method helps accelerate GD in the relevant direction, damping oscillations in the others. This method slightly changes the update equation, keeping a memory of the direction of the previous updates and giving inertia to the current update. The update vector of the current step $\boldsymbol{m_t}$ is built with the current gradient and with a fraction γ of the previous step update vector $\boldsymbol{m_{t-1}}$.

$$\begin{aligned} \boldsymbol{m_t} &= \gamma \boldsymbol{m_{t-1}} + \eta \nabla \mathcal{L} \\ \Delta \boldsymbol{p} &= -\boldsymbol{m_t} \end{aligned} \tag{2.24}$$

The parameter γ, called momentum term, is always positive and lower than 1.

Nesterov accelerated gradient (NAG) [181] Instead of computing the loss in the current point $\mathcal{L}(\boldsymbol{p})$, NAG anticipates the application of momentum and computes the gradient in the point $\mathcal{L}(\boldsymbol{p} - \gamma \boldsymbol{m_{t-1}})$, then it applies Eq. 2.24:

$$\begin{aligned} \boldsymbol{m_t} &= \gamma \boldsymbol{m_{t-1}} + \eta \nabla \mathcal{L}(\boldsymbol{p} - \gamma \boldsymbol{m_{t-1}}) \\ \Delta \boldsymbol{p} &= -\boldsymbol{m_t} \end{aligned} \tag{2.25}$$

Root Mean Squared Propagation (RMSprop) [91] This algorithm, together with others like AdaGrad [61] or AdaDelta [278], adapts the learning rate to the different parameters, performing updates proportional to the frequency of the associated features. The principle of the algorithm is to divide the learning rate by a quantity that is different for each parameter and is related to the previous updates. A running average E_t at time step t is defined recursively as:

$$\boldsymbol{E_t} = \gamma \boldsymbol{E_{t-1}} + (1 - \gamma)(\nabla \mathcal{L})^2 \tag{2.26}$$

The term γ is similar to the momentum term and is always positive and lower than 1. The running average accumulates a fraction of the previous squared value of the gradients. If a particular parameter p_i has been updated a lot in the past steps (i.e had a high gradient), the corresponding running average value $E_{t,i}$ will be high. After some steps with small updates, $E_{t,i}$ will then start to decrease.

The update vector is computed by dividing the learning rate by the square root of $\boldsymbol{E_t}$ plus a little value ϵ, to avoid dividing by 0. In this way, a frequently updated parameter will have a lower learning rate, while a rarer one will have a higher learning rate.

$$\Delta \boldsymbol{p} = -\frac{\eta}{\sqrt{\boldsymbol{E_t} + \epsilon}} \nabla \mathcal{L} \tag{2.27}$$

Adaptive moment estimation (Adam) [122] This algorithm computes adaptive learning rates like RMSprop, also adding momentum into account. In addition to a running average of the squared gradients $\boldsymbol{E_t}$, Adam also keeps a running average of the gradients $\boldsymbol{m_t}$, that can be seen as a sort of momentum with friction. The two averages at each time step t are defined as:

$$\begin{aligned} \boldsymbol{m_t} &= \beta_1 \boldsymbol{m_{t-1}} + (1-\beta_1)(\nabla \mathcal{L}) \\ \boldsymbol{E_t} &= \beta_2 \boldsymbol{E_{t-1}} + (1-\beta_2)(\nabla \mathcal{L})^2 \end{aligned} \tag{2.28}$$

Since the two vectors are initialized with zeros, they are biased towards 0, especially during the first steps, and if the two parameters β_1 and β_2 are close to 1. To decrease the effects of these biases, the two values are corrected as:

$$\begin{aligned} \hat{\boldsymbol{m}}_{\boldsymbol{t}} &= \frac{\boldsymbol{m_t}}{1-(\beta_1)^t} \\ \hat{\boldsymbol{E}}_{\boldsymbol{t}} &= \frac{\boldsymbol{E_t}}{1-(\beta_2)^t} \end{aligned} \tag{2.29}$$

In this way, when t is small, i.e. in the first steps, if the values start from 0, almost the entire value of the gradient is used to compute the two averages. As t increases, the denominators tend to 0, making the bias correction irrelevant. The parameter update is then computed as:

$$\Delta \boldsymbol{p} = -\frac{\eta}{\sqrt{\hat{\boldsymbol{E}}_{\boldsymbol{t}} + \epsilon}} \hat{\boldsymbol{m}}_{\boldsymbol{t}} \tag{2.30}$$

Alternatives to Adam, such as AdaMax [122], NAdam [60], AdamW [153] and RAdam [145], has been proposed, with slight variations to the original formulation. However, it has been found that the actual best optimizer to adopt is highly dependent on the architecture and the dataset under consideration and must be verified experimentally.

2.2.5 Regularization techniques

One key problem when dealing with ANN is their ability to generalize to unseen data and avoid *overfitting* the training set. A common interpretation of this generalization error is the so-called bias-variance decomposition [124], which states that it is the sum of three terms:

- the *bias* error, that comes from wrong assumptions in the model, i.e. the model is too simple and does not capture the correct input-output relation (underfitting);
- the *variance* error, that comes from a too high sensitivity for small fluctuations in the training set, i.e. the model is too complex and is learning noise in addition to actual knowledge from data (overfitting);
- the *irreducible* error, that is intrinsic in the problem itself.

An effective training procedure should look into reducing both bias, by increasing the network complexity, and variance by avoiding overfitting. Among the different strategies to reduce overfitting, such as increasing the training set or adopting an effective data augmentation strategy, one of the most important is network regularization. Regularizing a model means constraining its parameter space, thus reducing its ability to capture high-frequency fluctuations such as noise. The final objective of regularization is reducing the generalization error, without affecting the training error [75].

$L_{\mathbb{p}}$ regularization This method adds a term to the loss that penalizes the $\mathbb{p}$-norm of the parameters:

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \frac{\alpha}{\mathbb{p}} \|\boldsymbol{p}\|_{\mathbb{p}}^{\mathbb{p}} = \mathcal{L} + \frac{\alpha}{\mathbb{p}} \sum_{i}^{n} |p_i|^{\mathbb{p}} \tag{2.31}$$

In this way, the learning process tends to select solutions with parameters around zero, reducing the model ability to generate high-degree non-linear boundaries. The constant α, called regularization parameter, controls how much regularization is added to the loss. Usually, L_1, also called Lasso regularization, and L_2, also called Ridge regularization, are used, that penalize the sum of absolute values and the sum of squared values of parameters, respectively.

A particular type of regularization, called Elastic Net, adds both L_1 and L_2 regularizations to the loss term as follows:

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + r\alpha \|\boldsymbol{p}\|_1 + (1-r)\frac{\alpha}{2} \|\boldsymbol{p}\|_2^2 = \mathcal{L} + r\alpha \sum_{i}^{n} |p_i| + (1-r)\frac{\alpha}{2} \sum_{i}^{n} p_i^2 \tag{2.32}$$

where the parameter r controls the ratio between the two regularization terms: for $r = 1$, it is equivalent to L_1, while for $r = 0$, it is equivalent to L_2.

Dropout When Dropout [93, 232] is applied to a layer of the network, at each training step every neuron of that layer has an independent probability d, called *dropout rate*, to be inactivated, i.e. to give a zero output. That prevents co-adaptation, which is a phenomenon in which a certain neuron is only helpful in a context with other active neurons, and pushes feature redundancy. At inference time, all the neurons are kept active, enabling the full potential of the model. This approach can be considered as a sort of ensemble training of multiple simpler dependent models, in which certain connections are randomly dropped. For this reason, Dropout can be considered a regularization technique and is very effective in reducing model variance and overfitting.

More recently, another approach, called Monte Carlo (MC) Dropout [70], has been proposed. In this case, several predictions are subsequently performed keeping Dropout active. In this case, a proper ensemble is adopted, and the average

prediction is used. This can further push test performance, by reducing overfitting, but at the cost of multiplying inference cost. This method also allows to compute an estimation of the mean and the variance of the output distribution, thanks to the stochastic nature of the predictions.

Activations normalization This class of methods regularizes the model by normalizing the layer activations. This ensures that data flowing inside the computational graph is re-centered and re-scaled and, in general, that leads to more stable training and less prone to overfitting. The most popular normalization method is the Batch Normalization (BN, BatchNorm) [104]. This method normalizes activations channel-wise on the entire batch, and then applies a linear transformation to re-center and re-scale them. Taking as example a batched 2D tensor $\boldsymbol{x}$ with dimensions $N \times T \times C$, first the channel-wise mean and the standard deviation are computed as follows:

$$\mu^{(c)} = \sum_{i=1}^{NT} x_i^{(c)} \quad , \quad \sigma^{(c)} = \sum_{i=1}^{NT} \left(x_i^{(c)} - \mu^{(c)} \right) \qquad \forall c \in 1, \dots, C \tag{2.33}$$

where x_i is iterated on both the batch dimension N and on the first data dimension T, and $^{(c)}$ denotes the c-th channel on the last dimension C. Then, the normalization is applied with the computed channel-wise statistics:

$$\hat{\boldsymbol{x}}^{(\boldsymbol{c})} = \frac{\boldsymbol{x}^{(\boldsymbol{c})} - \mu^{(c)}}{\sqrt{\left(\sigma^{(c)}\right)^2 + \varepsilon}} \qquad \forall c \in 1, \dots, C \tag{2.34}$$

where $\hat{\boldsymbol{x}}$ is the normalized tensor and ε is an arbitrary small constant added for numerical stability. Eq. 2.34 performs a zero-centering and unitary-variance scaling of the channel-wise sub-tensors. Finally, the tensor is linearly re-centered and re-scaled with channel-wise parameters γ and β:

$$\boldsymbol{y}^{(\boldsymbol{c})} = \gamma^{(c)} \hat{\boldsymbol{x}}^{(\boldsymbol{c})} + \beta^{(c)} \qquad \forall c \in 1, \dots, C \tag{2.35}$$

where γ and β parameters are learnt with back-propagation.

Since $\mu^{(c)}$ and $\sigma^{(c)}$ represent the mean and standard deviation computed on the training batch dimension, they cannot be properly computed at inference time, where usually the batch is unitary. For this reason, a running average of these two quantities is aggregated with all the training steps in order to have an estimate for the population mean $E[\boldsymbol{x}^{(\boldsymbol{c})}]$ and the variance $\mathrm{Var}[\boldsymbol{x}^{(\boldsymbol{c})}]$, as follows:

$$\begin{aligned} E[\boldsymbol{x}^{(\boldsymbol{c})}] = E[\mu^{(c)}] \quad , \quad \mathrm{Var}[\boldsymbol{x}^{(\boldsymbol{c})}] = \frac{N}{N-1} E[\left(\sigma^{(c)}\right)^2] \\ \boldsymbol{y}_{\mathrm{inf}}^{(\boldsymbol{c})} = \gamma^{(c)} \frac{\boldsymbol{x}_{\mathrm{inf}}^{(\boldsymbol{c})} - E[\boldsymbol{x}^{(\boldsymbol{c})}]}{\sqrt{\mathrm{Var}[\boldsymbol{x}^{(\boldsymbol{c})}] + \varepsilon}} + \beta^{(c)} \qquad \forall c \in 1, \dots, C \end{aligned} \tag{2.36}$$

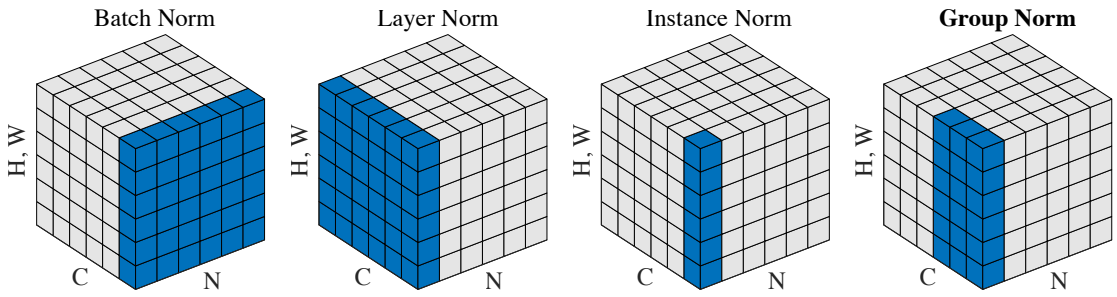


Figure 2.5: Activations normalization methods on batched 3D image tensors. C is the number of channels, H, W the spatial dimensions, N the number of samples in the batch. Data is re-centered and re-scaled independently on the highlighted sub-tensors. Batch Norm is the only method to normalize on the batch, therefore it requires the population statics accumulation for inference. Image is from [263].

This approach also ensures that deterministic output is always guaranteed at inference time. Overall, each BN layer adds a total of $2C$ learnable parameters (γ and β) and $2C$ non-learnable parameters (estimate of the population mean and variance).

Other types of normalization have been proposed in the literature, that re-center and re-scale activations with respect to different sub-tensors. The more relevant normalization methods are the following:

- **Layer Norm** [17]: it normalizes each sample individually on all the channels, i.e. the entire layer; mean and standard deviation computed sample-wise;
- **Instance Norm** [245]: it normalizes each sample on each channel individually; mean and standard deviation computed sample-and-channel-wise, i.e. just on the first data dimension T;
- **Group Norm** [263]: it normalizes each sample on subsets of g channels together, called groups; mean and standard deviation computed sample-and-group-wise.

Unlike Batch Norm, all these methods normalize activations on each sample independently, therefore they do not require any population mean and variance accumulation and do not present any execution difference during training and inference. A graphical representation of the different normalization methods is shown in Fig. 2.5. In this case, the data type is a batched 3D image tensor with dimensions $N \times H \times W \times C$.

Early stopping This regularization technique simply consists in stopping the training algorithm as soon as the validation loss begins to increase. Any further

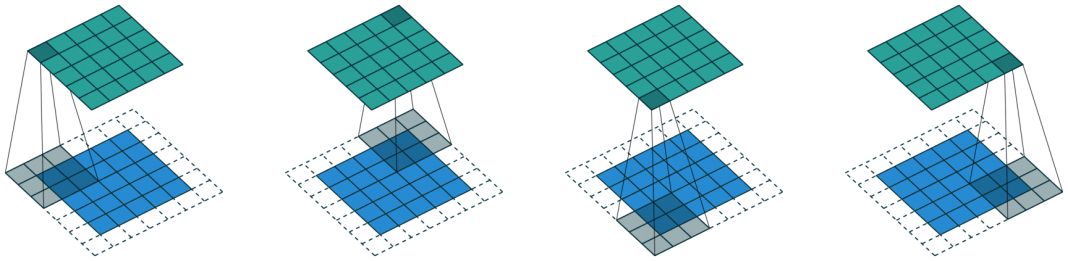


Figure 2.6: Graphical representation of a 2D convolution, with input shape $5 \times 5 \times 1$, kernel size $K_h = K_w = 3$, padding $p_h = p_w = 1$ and strides $s_h = s_w = 1$. Input is in blu, output in green. Image is from [62].

learning is considered to be just overfitting of the training set, and therefore only contributes to an increase in the model variance. For this reason, it is always important to monitor both training and validation losses during the learning process.

2.3 Other ANN architectures

2.3.1 Convolutional Neural Networks

A particular type of feed-forward ANN are the Convolutional Neural Networks (CNNs). This type of model has emerged since the 1980s for image recognition applications [69, 134, 135] and has been developed starting from the study of particular neuron cells present in the visual cortex of animals. A CNN is based on convolutional neurons, that react to information coming only from specific regions of the input image and to simple geometric patterns. Following neurons then, have a bigger receptive field and can recognize more complex patterns, building a hierarchical feature extraction model. In this way, a locality prior is added to the network, stating that data that are close to each other in the input 2D reference system have a high probability to be correlated. This approach ensures locality and weight sharing in the convolutional layers, which also has the advantage of shrinking the network parameters and increasing efficiency. CNNs were first developed to manage 2D data, such as images, but later they were generalized to 1D and 3D data. In this section, 2D convolutions are presented, together with common architectures to build SOTA feature extractors.

2D convolutions

The basic element of a CNN is the *convolutional layer*, that takes as input a 3D tensor $\boldsymbol{X}$, with shape $H \times W \times C$, where H and W are the height and width (in pixels) of the image, and C the number of channels. The first big difference of CNNs with respect to fully-connected NNs is that neurons have a *receptive field*,

which means that they are connected to only a part of the input matrix. This receptive field is a rectangular area of $K_h \times K_w$ pixels on the spatial dimensions and all the channels C on the last dimension. A neuron in position i, j has receptive field in positions $[i, i+1, \ldots, i+K_h-1], [j, j+1, \ldots, j+K_w-1]$ of the previous layer output. The operation implemented by a neuron is the same as Eq. 2.3, i.e. a weighted sum of the inputs, followed by an activation function $\phi(\cdot)$. The weight matrix is called convolutional *kernel* and is shared between all the neurons of the layer, such that their receptive fields are applied with a single-pixel shift. The kernel is also called $filter$, since the operation is similar to the application of filters as in classical image processing. It is possible to change how the kernel is applied, by sliding the window every $[s_h, s_w]$ pixels, called strides. In this way, fewer neurons are needed to cover the whole image, performing a spatial subsampling. Together, all the neurons sharing the same kernel, output again a 3D tensor, called *feature map*. Usually, each convolutional layer learns a certain number F of feature maps, each with a different convolutional kernel. Overall, a convolutional layer performs the following operation:

$$z_{ijk} = \sum_{t=0}^{K_h-1} \sum_{u=0}^{K_w-1} \sum_{v=0}^{C-1} w_{tuvk} x_{(i \cdot S_h + t)(j \cdot S_w + u)(v)} + b_k \quad \begin{cases} \forall i \in 0, \ldots, \lfloor (H - K_h)/S_h \rfloor \\ \forall j \in 0, \ldots, \lfloor (W - K_w)/S_w \rfloor \\ \forall k \in 1, \ldots, F \end{cases} \tag{2.37}$$

where $\boldsymbol{w}$ is the weight tensor that aggregates the kernel for all the feature maps, with shape $K_H \times K_W \times C \times F$. The total number of parameters for a convolutional layer is, therefore, $K_H \cdot K_W \cdot C \cdot F + F$ biases. If the kernel sizes are not unitary, the output feature maps have smaller spatial dimensions with respect to the input tensor. To avoid this, inputs are often padded of $[P_h, P_w]$ values before and after each spatial axis. If $P_h = \lfloor K_h/2 \rfloor$ and $P_w = \lfloor K_w/2 \rfloor$, the padding perfectly compensate the kernel effect. The padding is usually performed by simply adding constant zeros values, but more sophisticated padding strategies are possible. On the other hand, the strides $[s_h, s_w]$ perform a spatial subsampling by dividing the output dimensionality with a factor proportional to the stride values. The final output spatial dimensions H' and W' can be computed as follows:

$$H' = \left\lfloor \frac{H - K_h + 2P_h}{S_h} \right\rfloor + 1 \quad , \quad W' = \left\lfloor \frac{W - K_w + 2P_w}{S_w} \right\rfloor + 1 \tag{2.38}$$

Figure 2.6 shows the corner operations of a convolutional filter with input shape $5 \times 5 \times 1$, kernel size $K_h = K_w = 3$, padding $P_h = P_w = 1$ and strides $S_h = S_w = 1$. The output matrix has the same dimension as the input.

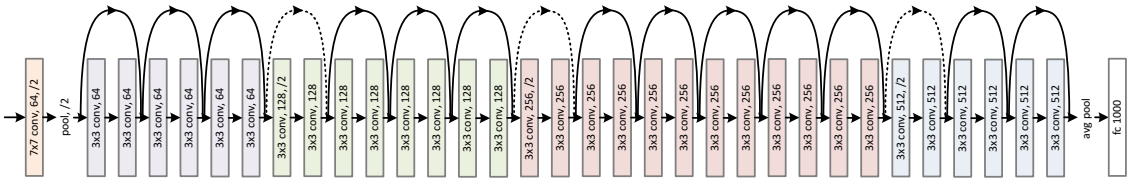


Figure 2.7: Architecture of a ResNet34 CNN. The model is based on residual connections and progressively halves the spatial dimension with stride 2 convolutions and pooling. After the backbone, an FC layer projects the extracted features in the output space (1000 classes). Image is from [87].

Pooling layers

Other frequently used layers in CNN are pooling layers. These operations spatially downsample the feature maps by aggregating values over a rectangular spatial window whose dimensions are called pool size. The spatial window is applied to the whole image in the same way as a convolutional kernel but with default strides equal to the pool size. Therefore, following Eq. 2.38, pooling layers divide the spatial dimensions of a factor equal to the pool sizes. Max Pooling takes the maximum value over the spatial window, while Average Pooling takes the mean value, therefore being equivalent to a convolution with a kernel made of all 1 and dimensions equal to the strides. In case the pool window has dimensions equal to the input spatial dimensions H and W, the pooling layers take the name of Global Max Pooling (GMP) and Global Average Pooling (GAP), respectively. These poolings reduce the tensor to unitary spatial dimensions and are therefore frequently used before a flattening operation to vectorize the extracted features.

CNN architectures

Typical CNN architectures have a lot of convolutional layers stacked one after the other. In classification problems, the input space is usually quickly reduced by avoiding padding, adopting non-unitary strides, and using pooling layers. After a feature extraction backbone characterized by convolutions and pooling, usually, a classification head is used, made of FC layers with a final softmax layer that models the probability distribution among the classes. This kind of architecture has been followed by a lot of SOTA methods, such as LeNet5 [134], AlexNet [125], GoogleLeNet [236], VGG [229] and ResNet [87]. Several methodologies have been proposed in the literature to help CNN feature extraction and generalization. Among the most relevant, it is worth mentioning inception blocks [236], residual connections [87], dense connections [101], depthwise separable convolutions [42], squeeze-and-excitation [99], channel and spatial attention [262]. Figure 2.7 shows a graphical representation of a ResNet34, as an example of a typical CNN architecture.

2.3.2 Transformer-based networks

Another popular ANN architecture is the Transformer [250]. It has been originally proposed for sequence-to-sequence problems for Natural Language Processing (NLP) embedding translation. Traditionally, sequential data has been processed with 1D CNNs or recurrent neural networks (RNNs), such as long short-term memory (LSTM) networks [94]. The Transformer is instead based on a structure inspired by autoencoders [3], made of an Encoder block, that extracts a vectorial representation of the input in a latent space, and a Decoder, that translates this representation into the output sequence. This kind of architecture has quickly reached the state-of-the-art for natural language processing [52], automatic speech recognition [79], but also computer vision with the Vision Transformer model [59].

Multi-head self-attention

The fundamental operation at the basis of the Transformer is the multi-head self-attention (MSA). This operation allows to escape the locality of convolutions and relate different positions in the sequence, even at a high temporal distance. On the contrary, CNNs are based on a sliding kernel and are therefore able to extract local patterns only.

Given three input sequence $\boldsymbol{X}$ with shape $T \times D_{\text{model}}$, self-attention is computed with the so-called scaled dot-product attention, in H independent heads with dimensionality $D_h = {}^{D_{\text{model}}}/_{H}$. As first step, three representations of the input sequence are obtained, namely query ($\boldsymbol{Q}$), key ($\boldsymbol{K}$) and value ($\boldsymbol{V}$):

$$\boldsymbol{Q}_i = \boldsymbol{X}\boldsymbol{W}_i^{(q)} \quad , \quad \boldsymbol{K}_i = \boldsymbol{X}\boldsymbol{W}_i^{(k)} \quad , \quad \boldsymbol{V}_i = \boldsymbol{X}\boldsymbol{W}_i^{(v)} \qquad \forall i \in 1, \dots, H \tag{2.39}$$

where $\boldsymbol{W}_i^{(q)}$,$\boldsymbol{W}_i^{(h)}$,$\boldsymbol{W}_i^{(v)}$ are weight vectors of FC layers with shape $D_{\text{model}} \times D_h$. Then, the self-attention can be computed as:

$$\boldsymbol{A}_i = \text{SA}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{D_h}}\right)\boldsymbol{V} \tag{2.40}$$

Finally, the output is obtained by concatenation of the different heads and a linear projection:

$$\boldsymbol{Y} = \text{MSA}(\boldsymbol{X}) = [\boldsymbol{A}_1, \boldsymbol{A}_2, \dots, \boldsymbol{A}_H]\boldsymbol{W}_y \tag{2.41}$$

where $\boldsymbol{W}_y$ has shape $D_{\text{model}} \times D_{\text{model}}$, since $H \cdot D_h = D_{\text{model}}$.

Transformer model

As already said, a Transformer is made of two main blocks: an Encoder and a Decoder. Both these blocks are based on N subsequent self-attention modules

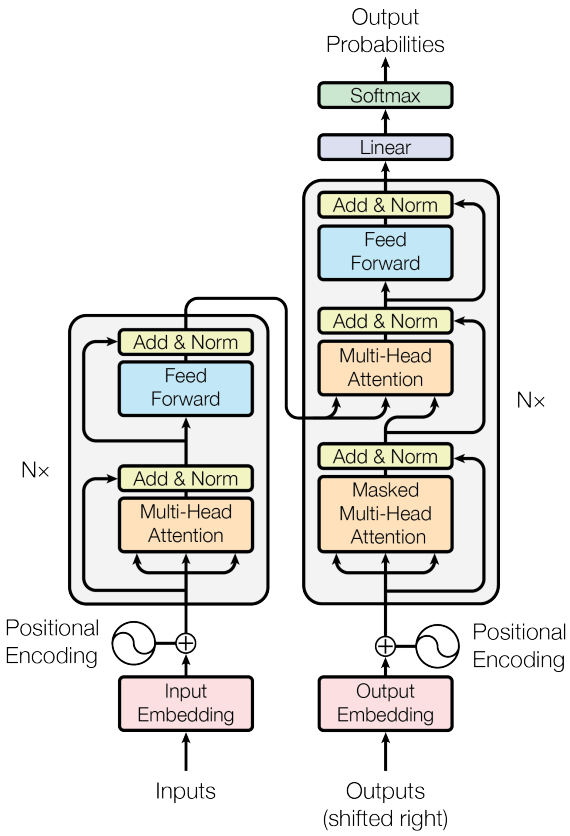


Figure 2.8: Architecture of the Transformer model for sequence-to-sequence tasks. Image is from [250].

and MLP modules that re-project the sequence vectors after attention. All these modules are arranged in a residual fashion and normalized with a Layer Norm [17]. Moreover, both the Encoder and the Decoder present an initial positional embedding layer, that mathematically encodes relative positions in the sequence vector by adding a vector to the input embedding. These positional embeddings can be learnable or made of constant geometric progressions. The main architectural difference between the Encoder and the Decoder is that the latter presents two attention modules:

- the *masked self-attention* performs an MSA on the decoded sequence, ensuring that each generic position i in the sequence can only attend past positions $j \leq i$;
- the *encoder-decoder attention* performs an MSA between queries from the Decoder sequence and keys and values from the Encoder output.

Moreover, at the end of the N Decoder blocks, an FC classification head with

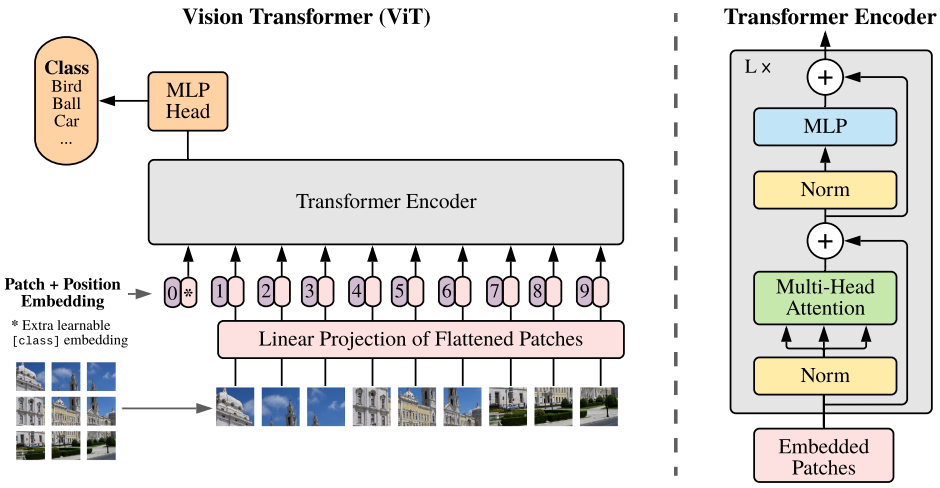


Figure 2.9: Architecture of the Vision Transformer (ViT). Image is from [59]

softmax activation predicts the next embedding in the output sequence. At inference time, the decoding process is iteratively performed until a pre-determined stopping embedding is predicted. The decoded sequence is initialized with a pre-determined starting embedding and, at each iteration, the embedding predicted in the last available sequence position is added to the Decoder input. Figure 2.8 shows the Transformer architecture, as presented in the original paper [250].

Transformers for computer vision

The Vision Transformer (ViT) [59] is an adaptation of the Transformer model for computer vision tasks. Due to the sequential nature of this kind of architecture, the input image with dimensions $H \times W$, is subdivided into N patches of a certain dimension $P \times P$, such that $N = {}^{HW}/_{P^2}$. The patches are then flattened and linearly projected by an FC layer to reach the target dimensionality D_{model}. An additional embedding, called class token, is prepended to sequence to encode the class representation. The obtained sequence is then processed with a standard Transformer Encoder architecture, with pre-normalization, i.e. Layer Norm si applied at the beginning of the residual unit, before the main module (MSA or MLP). A final classification head is used to predict the output distribution using the class token state, only. Figure 2.9 shows the ViT model architecture.

This kind of architecture has been proven very effective for computer vision tasks, reaching state-of-the-art performances. Several variants has been proposed such as the DeiT [242], the ConViT [47], the LeViT [78] and the Swin Transformer [147].

2.4 Edge execution of ANN

ANN are usually executed on general purpose graphic processing units (GP-GPU), thanks to specialized libraries, such as Nvidia CUDA[1], that allow for very efficient parallel matrix computations. However, for robotic applications, an important factor to be considered is the possibility of running ML and ANN algorithms directly on-board, using the robot's main computing platform with limited power, memory, and computational ability. All the techniques conceived to deploy AI algorithms on low-power embedded platforms come under the name of *Edge-AI*. Following such an approach can bring enormous advantages in terms of latency, privacy, and connectivity requirements since the AI computation is performed directly where the data is collected and the result is needed. To this aim, several graph optimization and quantization methods have been investigated, to both decrease model size and computational cost. In the literature, different techniques to increase ANN efficiency can be found [73, 80, 83, 84, 108]. In particular, the relevant approaches for this dissertation are the following:

- network pruning and layer fusing that consists in optimizing the graph by removing low-weight nodes that give almost no contribution to the outcome and fuse different operations to increase efficiency;
- weights quantization that consists of reducing the number of bits required to represent each network parameter;
- activations quantization, that 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 it;
- knowledge distillation, which consists in transferring knowledge from a large model to a smaller one.

An implementation of different optimization and quantization methods is available with the TFLite library[2]. For what concerns the different quantization schemes, TFLite allows for several approaches. The first is the float16 quantization of weights, in which the model is still executed with float operations, but each weight will require half the memory. Weights can also be quantized to 8-bit precision, a method referred to as dynamic range quantization, to allow for almost 4 times reduction of model size, with minimal loss in precision. Another approach is the full

[1] https://developer.nvidia.com/cuda-zone

[2] https://tensorflow.org/lite

integer quantization, which converts both weights and activations to 8-bit fixed-point integers. This allows for a great reduction in model size, as well as a good boost in inference speed. However, this approach can cause a significant drop in accuracy, depending on the specific application. This scheme can also be followed using quantization-aware training, which 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, reducing the accuracy drop, at the cost of a more complex training setup. All these methodologies can be used to target specific mobile GPUs or for simple CPU execution. Another library, TensorRT[3], developed by Nvidia, allows instead to optimize models to target GP-GPUs.

Full-integer quantization

This strategy is the most radical to increase network efficiency by changing the representation of both weights and activations to 8-bit integers, greatly reducing memory and computational demands due to the high efficiency of integer computations. The only exception are the biases parameters that are quantized as 32-bit integers. This reason behind this choice is to preserve good end-to-end neural network accuracy: since biases are added to many output activations, having a big quantization error in the bias vectors would mean introduce an overall bias error to network output [108]. Moreover, this quantization scheme is acceptable since biases account for only a tiny fraction of the parameters in a neural network, so the their memory footprint is limited. According to the methodology presented by Jacob et al. [108], each floating point value is quantized with the following scheme:

$$r = S(q - Z) \tag{2.42}$$

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. The zero point has the same type of q and represents the quantized value corresponding to the real value $r = 0$.

A fixed-point multiplication approach is adopted to cope with 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. Given a multiplication between two real values r_1 and r_2, using Eq. 2.42, we can write:

$$\begin{aligned} r_3 &= r_1 r_2 \\ S_3(q_3 - Z_3) &= S_1(q_1 - Z_1)S_2(q_2 - Z_2) \\ q_3 &= Z_3 + M(q_1 - Z_1)(q_2 - Z_2) \end{aligned} \tag{2.43}$$

[3] https://developer.nvidia.com/tensorrt

where the multiplier $M = \frac{S_1 S_2}{S_3}$ is the only non-integer value involved. This approach can be extended for generic matrix multiplications, having a single scale for each tensor. M can be computed offline since it only depends on the quantization parameters, not on the actual activations, and it is experimentally in the (0,1) range. Therefore it can be normalized as:

$$M = 2^{-n} M_0 \tag{2.44}$$

with $M_0 \in [0.5,1)$ and n integer. M_0 can be represented as a 32-bit fixed point with an int32 value and multiplied with a fixed-point multiplication [108]. On the other hand, the 2^{-n} multiplication can be simply implemented as a shift right operation. With this approach, all the operations involve integers-only operations and can be executed on specific hardware that does not support floating points.

Knowledge distillation

Knowledge distillation (KD) [92] is a methodology to transfer knowledge from one model, the teacher, to another one, the student. If used between a large model and a smaller one, it can be considered a way to decrease the model computational complexity and therefore speed up inference and reduce memory footprint. The classical distillation approach adds a contribution to the student loss that takes into account the teacher probability distribution. Denoting the logits distribution of the student and the teacher as $\boldsymbol{z_s}$ and $\boldsymbol{z_t}$ and the softmax activation as $\phi(\cdot)$, the loss is computed as:

$$\mathcal{L}_\text{s} = (1-\alpha)\,\text{CE}(\phi(\boldsymbol{z_s}), \boldsymbol{y}) + \alpha\tau^2\,\text{KL}(\phi(\boldsymbol{z_s}/\tau), \phi(\boldsymbol{z_t}/\tau)) \tag{2.45}$$

where the first part is the standard cross-entropy loss with respect to the labels $\boldsymbol{y}$, while the second is the Kullback-Leibler divergence [126] loss between the student and teacher logits 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. λ is the parameter balancing the two loss contribution.

Touvron et al. [242] introduced a hard label KD for training a data-efficient ViT, denoted as DeiT. In this case, a simple cross-entropy contribution is added to the loss, without temperature, to match the student output distribution to the teacher hard prediction $\boldsymbol{y_t}$, using an additional distillation token:

$$\mathcal{L}_\text{s} = \frac{1}{2}\,\text{CE}(\phi(\boldsymbol{z_s}), \boldsymbol{y}) + \frac{1}{2}\,\text{CE}(\phi(\boldsymbol{z_s}), \boldsymbol{y_t}) \tag{2.46}$$

Several other strategies for distillation have been proposed in the literature such as distilling intermediate representations [204] or attention-based distillation [277].

Hardware devices for Edge-AI

ANN models can be optimized for inference on a generic CPU allowing their edge execution on almost every possible robotic platform. However, especially for very deep image processing neural networks, it may not be enough, causing inference with very low frames per second (fps). Certain applications, such as visual-based robotic control, require very low-latency predictions to ensure real-time execution and prompt response to sudden changes in environmental conditions. For this reason, specific computing platforms can be selected, in order to accelerate the execution of AI models. Among the different commercially available solutions, those particularly relevant for the present dissertation are:

- Intel Movidius VPU (Vision Processing Unit): processors specifically designed for accelerating computer vision and AI models;
- Nvidia Jetson platforms: single-board computers with dedicated embedded GPUs to accelerate matricial operations with the CUDA library software;
- Coral Edge TPU (Tensor Processing Unit): integrated circuit designed for neural network fast inference; since it is limited to integer-only operations, it requires full-integer model quantization.

These devices present very different features in terms of inference performance, supported data types, power consumption, physical characteristics, and price. Tab. 2.1 presents an overview of different AI-oriented platforms. Considering the requirements of the specific application, the most suitable device can be selected for the onboard ANN deployment.

	Movidius NCS†	**Movidius NCS 2**†	**Jetson Nano**	**Jetson AGX Xavier**	**Coral USB Accelerator**†	**Coral Dev Board**
HW Accelerator	Myriad 2 VPU	Myriad X VPU	128-core Nvidia Maxwell GPU	512-core Nvidia Volta GPU	Edge TPU	Edge TPU
Computational performance	100 GFLOPS	150 GFLOPS	472 GFLOPS	16 TFLOPS	4 TOPS	4 TOPS
Inference data type	FP16	FP16	FP32	FP32	INT8	INT8
CPU	N.A.	N.A.	Quad-core Arm Cortex-A57	Octa-core NVIDIA Carmel Arm	N.A.	Quad-core Arm Cortex-A53
Memory	4GB LPDDR3	4GB LPDDR3	4GB LPDDR4	16GB LPDDR4	8 MB SRAM	1 / 4 GB LPDDR4 + 8 MB SRAM (ETPU)
Storage	N.A.	N.A.	16 GB eMMC	32 GB eMMC	N.A.	8 GB eMMC
Power consumption	1 W	1.5 W	5 / 10 W	10 / 15 / 30 W	2 W	8.5 W
Size	73 x 26 mm	73 x 27 mm	70 x 45 mm	100 x 87 mm	65 x 30 mm	88 x 60 mm
Price	$ 70	$ 74	$ 99	$ 699	$ 60	$ 130 / 170

Table 2.1: Main specifications of some AI-oriented embedded platforms. Each device is reported with its related commercial price at the time of writing.
†: this device requires an additional system with a CPU (e.g. Raspberry Pi single-board computer).

Part II

Navigation

Chapter 3

A Deep Learning Driven Pipeline for Autonomous Navigation in Row-based Crops

Agriculture 4.0 is introducing digital tools and technologies in precision farming. According to this innovative paradigm, Big Data, Artificial Intelligence and robotics play a key role in increasing the economic, environmental, and social sustainability of agricultural processes, thanks to the efficient and automatic data collection and the processing tools they provide. In the last years, DL research has been substantially contributing to the development of new technologies for precision agriculture applications. Self-driving machines represent a crucial component in reducing the costs of agricultural processes by providing autonomous, full-time and weather-independent operators. In this context, designing a reliable autonomous navigation system in constrained row-based crops such as vineyards and orchards is fundamental, since the ability to autonomously navigate in the fields with full coverage represents the very first step to perform any agricultural task.

Building on the latest DL research for computer vision and signal processing, in this part of the thesis, we present a complete algorithmic pipeline for autonomous navigation in row-based crops. The proposed robust solution is explicitly designed to ensure complete coverage of the field in different situations and make use of a combination of out-field steps to generate a suitable path to be followed and in-field robotic navigation control. Firstly, we adopt a robust data-driven methodology to generate a viable path for the autonomous machine, covering the full extension of the crop, starting from a bird-eye view of the region-of-interest obtained either by satellite data or drone footage, from which the occupancy grid map information is

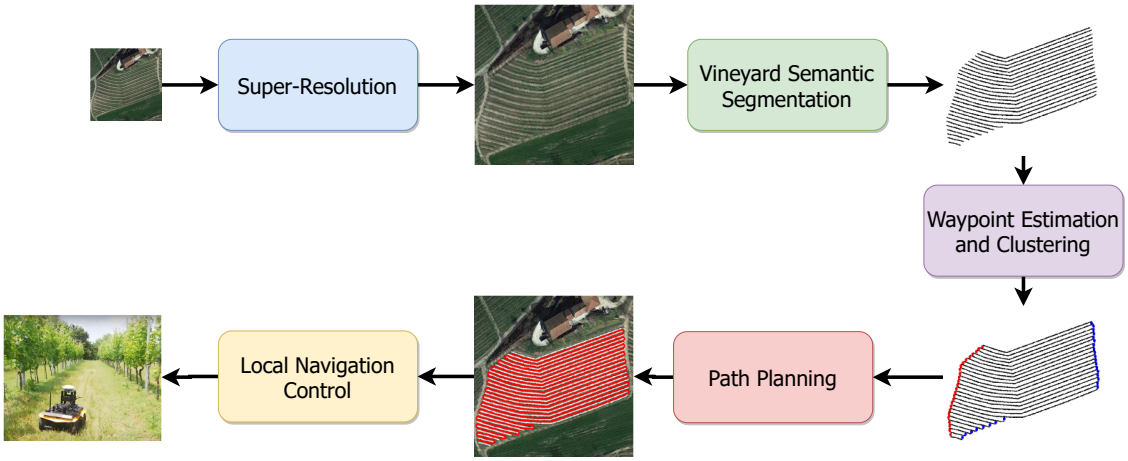


Figure 3.1: Scheme of the proposed pipeline for robotic autonomous navigation in row-based crop fields.

extracted. Successively, using the previously planned trajectories, the navigation control can be achieved using different strategies, such as a purely Global Navigation Satellite System (GNSS) path following or vision-based navigation algorithms that make use of a low-cost RGB-D camera to navigate. It must be underlined how the proposed approach allows to autonomously navigate through row-based crops, adopting a low cost GPS sensor for the intra-row navigation, since it uses vision as main source of information; however a more expensive Real-Time Kinematic (RTK) sensor is still needed to take care of the inter-row segments that connect one row to the other. Future work may investigate alternative approaches to handle end-of-row turns, in order to completely avoid the need for this expensive hardware.

3.1 The pipeline

Fig. 3.1 shows a graphical representation of the proposed pipeline. The autonomous system is organized into several modules that should collaborate with each other in order to obtain effective and reliable driverless navigation throughout the whole field. The input of the system is a generic image $\boldsymbol{X}_{\text{in}} \in \mathbb{R}^{H \times W \times C}$, with H and W are the height and width in pixels and C the number of channels (3 for RGB data). This image should contain a georeferenced bird-eye view of the target row-based field and can be obtained with remote sensing methodologies such as satellite monitoring or drone imagery. Georeferentiation allows to translate each pixel coordinate in the GNSS reference frame, allowing for successive in-field matching between the robot position and the path to be followed.

Super-Resolution

This optional module can increase the input resolution in case it is needed. Depending on the specific input sensor adopted, the image resolution can be non-optimal for the successive path planning operation. For example, the European Space Agency (ESA) Sentinel-2 satellite[1] provides resolutions up to 10 m per pixel, depending on the bandwidth. To solve this problem, several methodologies, referred to as Super-Resolution (SR), can be adopted in order to reconstruct an upsampled version $\boldsymbol{X}_{\text{HR}}$ of the input. Due to the quite stable conditions of the agricultural environments, several views of the same region-of-interest can be taken with limited variations in space and time, in order to adopt the so-called Multi-Image SR (MISR) strategy, which exploits these multiple sources of non-redundant information, in order to better estimate the higher resolution image. A state-of-the-art methodology for MISR applied to remote sensing data is presented in Chapter 4.

Vineyard Semantic Segmentation

This module has the objective of obtaining an occupancy grid of the target field, that is a binary map $\boldsymbol{X}_{\text{occ}}$, that marks with 1 the plant rows and with 0 the free terrain. Semantic segmentation is a very well established task in DL, in which the objective is to predict masks to classify pixels on a semantic basis. Differently from standard image classification tasks, which simply assign a label to the whole image, in this kind of problems, labels are assigned on pixel scale. Classical approaches to this task are based on fully-convolutional networks organized in an encoder-decoder fashion [151, 205]. A methodology for vineyard segmentation, together with a specific dataset based on row-based field remote-sensed images, is presented in [225], where a U-net [205] with a ResNet50 [87] backbone and HRNet+OCR [274] networks are compared on the specific task. This work has been developed as a master thesis in our research center and has been focused in order to be easily integrated into the proposed pipeline.

Waypoint Estimation and Clustering

This module has the objective to estimate N navigational waypoints $\boldsymbol{P} = \{\boldsymbol{p}_i \in \mathbb{R}^2 \ \forall i \in 1, \dots, N\}$ that should be touched by the path in order to perform full-coverage navigation. These waypoints can be clustered into the row starting/ending groups, and the final order is simply obtained following an A-B-B-A scheme between the two. To achieve this, a DNN is used to perform both waypoint estimation and clustering, adopting a custom synthetic occupancy grid dataset for training, and real-world satellite images for testing. This module is presented and detailed in Chapter 5.

[1] https://sentinel.esa.int/web/sentinel/missions/sentinel-2/

Path Planning

With this block, a full path is obtained, which is the actual succession of points that the robot should follow. This trajectory should connect the waypoints in the order given by the previous module, avoiding any collision with the crops, and ensuring, if possible, centrality with respect to the field rows. Any algorithm that supports an occupancy grid and point-to-point planning can be used in this context, such as the popular A* [86], RRT* [131] and D* [234, 123]. A custom global planner named Adaptive Row Crops Path Generator (ARC-PG) [32] allows for a reduced number of iterations and a higher path centrality, in the specific case of row-crop planning. Examples of full-coverage paths obtained with ARC-PG, together with an explanation of its basic working principles, are presented at the end of Chapter 5.

Local Navigation Control

This module should ensure path following and obstacle avoidance by in-field control of the robot. The most basic approach is simply to adopt an arbitrary controller in the GNSS reference frame, such as, for example, the Dynamic Window Approach (DWA) [67] or the Pure Pursuit [46], using Real-Time Kinematic (RTK), GNSS signals and inertial data provided by an IMU for localization. Since the original path is obtained in this reference system, thanks to the georeferenced input image $\boldsymbol{X}$, this is the most straightforward approach to tackle navigation. Any suitable additional hardware and software modules can be integrated, to adopt common navigational features such as obstacle avoidance, with the usage, for example, of ranging sensors, or path smoothness. However, meteorological conditions and especially lush vegetation and thick canopies, can significantly affect GNSS reliability, degrading its precision and consequently the overall navigation performance [114, 164]. For this reason, vision-based strategies can be exploited, in order to rely on semantic information of the environment to navigate between rows, and depth information to refine the underline control smoothness, disentangling from the necessity of a precise localization. This alternative approach, makes use of RTK corrections, GNSS signals, and inertial data to take care of inter-row segments that connect one row to the other, since outside the row space a good sky view is available. On the other hand, the intra-row segments can be navigated by exploiting visual information, adopting a camera and DL-based algorithms to overcome GNSS signals unreliability. Future work may investigate alternative methodologies for inter-row segments, as well, in order to completely avoid the need for an expensive RTK sensor. Two different DL approaches have been proposed to use visual data for intra-row navigation.

Segmentation-based control This approach [4, 33] takes an RGB frame $\boldsymbol{X}_{\text{rgb}} \in \mathbb{R}^{H_{\text{rgb}} \times W_{\text{rgb}} \times C_{\text{rgb}}}$ from the robot front RGB-D camera and produces a binary map,

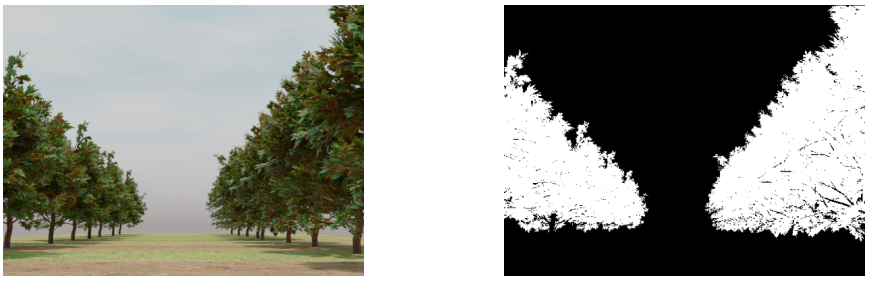


Figure 3.2: An example of synthetic RGB image obtained in a simulated row-based field and the corresponding segmentation mask.

$\boldsymbol{X}_{\text{seg}} \in \mathbb{R}^{H_{\text{rgb}} \times W_{\text{rgb}}}$ with H_{rgb}, W_{rgb} and C_{rgb} as height, width and channels, respectively. The output positive class segments the crops and the foliage in the camera view. Ideally, it should be equally split into the sides of the frame for a perfectly centered path due to the environment symmetry. Among all recent real-time semantic segmentation models available in the literature, we carefully select an architecture that guarantees high accuracy levels by also containing hardware costs, optimization simplicity, and computational load. Indeed, the segmentation-based control does not considerably benefit from fine-grained predictions and elaborated encoder-decoder networks [100] or two-pathway backbones [272]. Therefore, a very lightweight backbone, MobileNetV3 [97], is adopted followed by a reduced version of the Atrous Spatial Pyramid Pooling module [38] to capture richer contextual information with minimal computational impact. The training is performed with synthetic images obtained in a simulated row-based field and tested on real-world samples. An example of a training image, together with the corresponding segmentation mask, is shown in Fig. 3.2. To ease the dataset construction process, segmentation is performed on RGB data only.

The extracted semantic information $\boldsymbol{X}_{\text{seg}}$ is used in conjunction with the camera depth map (D channel) to control the movements of the platform, with the objective of equalizing the segmented pixels on the sides of the frame. The depth information is adopted to reduce the line of sight of the actual scene and remove some background noise. A sum of S segmentation maps, obtained in subsequent temporal instants, is used in order to obtain a more stable control. The control is based on the identification of the largest cluster of zeros in the segmentation map, which represents free space. This cluster contains the obstacle-free space information that can be exploited to safely drive the mobile platform; as a consequence the linear (v) and angular (ω) velocities are computed using the center of the selected cluster, which ideally corresponds to the center line of the row. The desired velocities are obtained by means of the following equations:

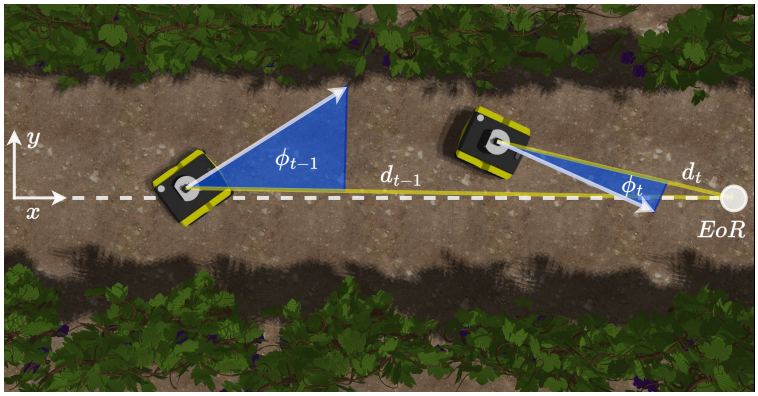


Figure 3.3: In the DRL navigation control algorithm [167], the reward at each time step t is computed as a function of the distances from the end of the row (EoR) d_t and d_{t-1}, and the angle ϕ_t between the robot orientation and the shortest path to EoR. This information is available while training the agent although it does not constitute its input. All the training process is performed in a simulated vineyard environment.

$$v = v_{\max}\left[1 - \left(\frac{2d}{W_{\text{rgb}}}\right)^2\right] \tag{3.1}$$

$$\omega = \omega_{\text{gain}} d \tag{3.2}$$

where $v_{\max} = 1.0$ and $\omega_{\text{gain}} = 0.01$ are two constants that depend on the specific robotic platform, and d is defined as:

$$d = x_c - \frac{W_{\text{rgb}}}{2} \tag{3.3}$$

with x_c center coordinate of the selected cluster. Eventually, the control velocity commands sent to the actuators are smoothed using an exponential moving average (EMA), in order to prevent the mobile platform from sharp motion.

DRL control An alternative DL-based approach for navigation control relies on training an end-to-end sensorimotor agent [167], which directly maps a noisy depth image $\boldsymbol{X}_{\text{depth}} \in \mathbb{R}^{H_{\text{depth}} \times W_{\text{depth}} \times 1}$ and position-agnostic robot state information to velocity commands (linear velocity v and angular velocity ω), with the objective of guiding the robot to the end of a row. Instead of having complex processing operations on the input depth image, this approach directly concatenates visual latent representations, extracted from $\boldsymbol{X}_{\text{depth}}$ with a simple 4-layer convolutional backbone, to the state of the robot at time t (yaw ψ_t and previous commands v_{t-1}

and ω_{t-1}). This feature tensor is then processed with a 3-layer MLP that predicts the new commands v_t and ω_t. The input features are selected considering the odometric and perception data available during the real vineyard navigation task, only. Therefore, this approach enables localization-independent navigation, with an affordable perception system such as a simple depth camera and an IMU, avoiding any possibly unreliable GNNS signal. The resulting simple and efficient model is trained with the Deep Reinforcement Learning (DRL) paradigm, following the Soft actor-critic (SAC) algorithm [82]. This learning methodology relies on a trial-and-error approach through simulation, giving the agent feedback with the aim of maximizing a function called the *reward*. The training process is therefore performed with a model of the robotic platform in a simulated vineyard environment.

For the specific application, the reward should take into consideration two main contributions: robot orientation and distance from the goal. In particular, we first define a reward contribution r_h to keep the robot oriented towards the end of the row:

$$r_h = \left(1 - 2\sqrt{\left|\frac{\phi}{\pi}\right|}\right) \tag{3.4}$$

where ϕ is the heading angle of the robot, namely the angle between its linear velocity and the end of the row. Then we adopt a distance-based reward to strongly encourage the agent to reach the end of the row:

$$r_d = d_{t-1} - d_t \tag{3.5}$$

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. 3.3. Robot pose information is uniquely used for reward computing while training, and is not included as agent input, as already stated before. We finally include a sparse reward contribution for end-of-episode states, assigning $r_s = 1000$ for the successful completion of the task, $r_c = -500$ if a collision occurs, and $r_\psi = -500$ if the robot overcomes a $\pm 85°$ yaw limit and starts to go in the opposite direction. Stopping the episode when the robot exits the vineyard row or reverses its motion direction is fundamental to keep collecting meaningful sample transitions for the task. The final reward is computed as follows:

$$r = \alpha \cdot r_h + \beta \cdot r_d + \begin{cases} r_s & \textit{if success} \\ r_c & \textit{if collision} \\ r_\psi & \textit{if reverse} \end{cases} \tag{3.6}$$

where $\alpha = 0.6$ and $\beta = 35$ are numerical coefficients to combine the different contributions.

(a) Test vineyard row (b) Test pear orchard row

Figure 3.4: Real-world testing environments used in the experimentation.

3.2 Real-world experimentation

The full pipeline is tested in two real environment scenarios with multiple experiments in different seasonal periods: a vineyard and a pear orchard, depicted in Fig. 3.4. We apply the full pipeline to plan the path and then measure the navigation performance in-field, using the segmentation-based control. All the tests are performed with the same hardware and software setup to obtain 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 (UNITO). Instead, the pear orchard is located in Montegrosso d'Asti and managed by Mura Mura farm. The first scenario has an inter-row space of about $2.80m$ and a height of about $2.0m$, while the second is organized in rows with an inter-row space of $4.50m$ and a height of about $3.0m$.

Robotic hardware and sensors setup

As mobile platform, we select the Jackal Unmanned Ground Vehicle (UGV) by Clearpath Robotics[2], which can be briefly described as a small and weatherproof rover (IP62 code) with a 4x4 high-torque drivetrain. It is highly customizable and ROS-compatible allowing fast deployment and algorithm testing. All the algorithms run on the onboard Mini-ITX PC with a CPU Intel Core i3-4330TE @2.4GHz and 4GB DDR3 RAM. For what concern the localization sensors, the RTK-enabled GNSS receiver is the Piksi Multi by Swift Navigation mounted on an evaluation board, which provides easy input/output communication with the

[2] https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

Test	N. rows	MAE [m]	RMSE [m]
Test n. 1	4	0.296	0.332
Test n. 2	6	0.218	0.240
Test n. 3	6	0.204	0.246

Table 3.1: 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.

Test	N. rows	MAE [m]	RMSE [m]
Test n. 1	4	0.523	0.627
Test n. 2	4	0.457	0.551
Test n. 3	2	0.659	0.755

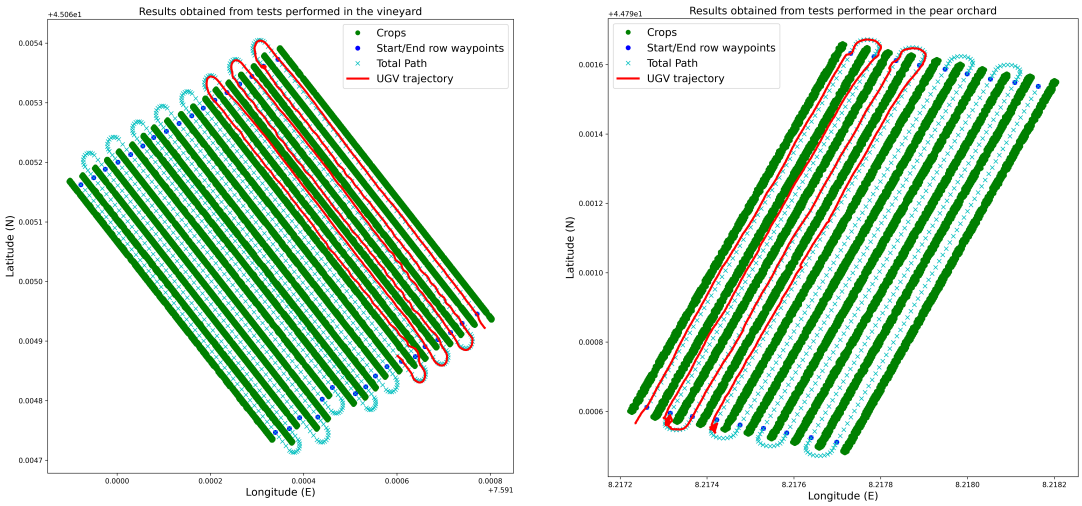
Table 3.2: Comparison of different error metrics in three different tests performed in the pear orchard. The second column describes the number of visited rows in the corresponding test.

receiver (acquisition rate of 10 Hz), while the inertial measurements are provided by the MPU-9250 IMU (acquisition rate of 100 Hz). In addition, to get a front view of the environment, we select the Intel Realsense D455 RGBD camera, that provides frames at 30 fps, mounted on the front part of Jackal top plate. Finally, the odometry is provided by the onboard quadrature encoders that are able to run at 78000 pulses/m. As mentioned before, the Local Navigation Control module fuses the IMU and GNSS receiver data in order to obtain a global position estimate of the mobile platform time by time. However, GNSS positioning is highly inaccurate without implementing any corrections technique. As a consequence, we provide RTK corrections to Piksi Multi receiver, coming from the SPIN3 GNSS[3] of Piemonte, Lombardia, and Valle d'Aosta, through an internet connection. Then, the GNSS receiver directly uses such corrections to obtain more reliable and accurate global position estimates, with an error range of $[0.05,0.10]m$, in a clear view of the sky and a good antenna position.

In-field results

Tab. 3.1 and 3.2 show navigational errors, in terms of mean absolute error (MAE) and root mean squared error (RMSE), computed comparing the RTK-GNSS positions provided by the Piksi Multi receiver and the target path provided by the

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



(a) Test n. 3 in the vineyard scenario

(b) Test n. 2 in the orchard scenario

Figure 3.5: A visual representation of the obtained results. Both images contain the path followed by the UGV (red line), the target global path provided by the Path Planning module (cyan x), the start/end row waypoints (blue dots) and the crop (green dots).

Path Planning module 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 GNSS coordinates. However, for analysis purposes, they have been transformed in meters with respect to the top left corner pixel of the georeferenced occupancy grid map. Results demonstrate that a methodology that exploits visual data with semantic segmentation, along with a standard navigation approach based on GNSS signals, is able to provide complete and reliable navigation throughout the whole row-based crop. Results obtained in the pear orchard are slightly worse than the vineyard ones; however, this effect may be due to the greater inter-row space of the pear orchard. 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 to the used low-cost sensors setup and demonstrate the effectiveness of the proposed approach. Fig. 3.5 shows the trajectory followed by the UGV in two real-world tests.

Chapter 4

RAMS: Multi-Image Super-Resolution of Remotely Sensed Images using Residual Attention Deep Neural Networks

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

SR is very significant to Remote Sensing because it provides the opportunity to enhance LR images despite the inherent problems often faced in remote-sensing scenarios. The hardware and material costs for smaller missions around data accumulation are very high. Additionally, onboard instruments on satellites continue to generate ever-increasing data as spatial and spectral resolutions also increase, and this has progressively become challenging for compression algorithms [247], as they try to meet the bandwidth restrictions [23, 246]. Remote sensing is fundamental in obtaining images covering most of the globe, permitting many vital projects such as disaster monitoring, military surveillance, urban maps, and vegetation growth monitoring. It is thus imperative that enhancements and progress be made in

post-processing techniques to overcome obstacles of increasing spatial resolution.

There are two main methods used in Super-Resolution: Single-Image SR (SISR) and Multi-Image SR (MISR). SISR employs a single image to reconstruct an HR version of it. Contrastingly, MISR involves multiple LR images of the same scene acquired from the same or different sensors to construct an HR image. The significant advantage MISR holds over SISR is in how it can draw out otherwise unavailable information from the different image observations of the same scene.

Learning-based SISR methods build upon the relation between LR-HR images, and there have been many recent advancements in this approach, mostly due to deep convolutional neural networks (CNNs) [57, 120, 55]. The leading force for this was Dong et al. [56], who achieved superior results by proposing a Super-Resolution CNN (SRCNN), and Tai et al. that pioneered deep recursive residual network (DRRN) [237]. Shi et al. [224] introduced an efficient sub-pixel convolution layer that learns an array of upscaling filters to efficiently extract the main network features from the LR image, and only project them to the HR space in the last block of the model. This methodology became the *de facto* standard for SR, supplanting other feature upscaling methods such as bilinear or bicubic interpolation or transpose convolutions. Later, Ledig et al. [136] proposed SRGAN, a Generative Adversarial Network (GAN) for photo-realistic SR with perceptual losses [113]. In recent past years, enhancements in deep networks have been proposed and showed promising results for SISR, for example, in [139], an Enhanced Deep Super-Resolution (EDSR) network was developed to improve the performance by removing unnecessary modules and expanding the model size with the stable training process in conventional residual networks. Yu et. al [273] demonstrated better results in terms of accurate SR by generating models with a wide range of features before ReLU activation and training with normalized weights. Zhang et. al [283] proposed residual channel attention networks (RCAN) that exploit very deep network structure based on residual in residual (RIR) which bypasses excessive low-frequency information through multiple skip connections.

On the other hand, Multi-Image SR (MISR) involves the extraction of information from many LR observations of the same scene to reconstruct HR images [275]. Iterative Back Projection (IBP), introduced by Irani and Peleg [105], used a back-projection of the difference between the actual LR images obtained and the simulated LR images to the SR image. The forward imaging process is inverted and iteratively attempted in updates. An enhanced Fast and Robust SR (FRSR) [66] employed estimation of maximum likelihood analysis and simplified regulation. More recently, many DL-based approaches have been exploited to address the MISR problems in the context of enhancing video sequences [116, 28, 112]. Kawulok et al [119] demonstrated the potential benefits of information fusion offered by multiple satellite images reconstruction and learning-based SISR approaches. In their work, EvoNet framework [118] based on several deep CNNs was adopted to exploit SISR in the preprocessing phase of the input data for MISR. Recently, a challenge

was set by the European Space Agency (ESA) to super-resolve multi-temporal PROBA-V satellite imagery. In this context, a new CNN-based architecture DeepSUM was proposed by Molini et. al [177], in which an end-to-end learning approach was established by exploiting both spatial and temporal correlations. Most recently, Deudon et. al presented HighRes-net [51], proposing an end-to-end mechanism that learns the sub-tasks involved in MISR, which are co-registration, fusion, upsampling, and registration-at-the-loss.

In this contribution, we propose a DL MISR solution for remote-sensing applications that exploits both spatial and temporal correlations to combine multiple low-resolution acquisitions smartly. Indeed, our model provides a real end-to-end efficient solution to recover high-resolution images, overcoming limitations of previous similar methodologies, and providing enhanced reconstruction results. In particular, the main contributions of this work lie in:

1. the use of 3D convolutions to efficiently extract, directly from the stack of multiple low-resolution images, high-level representations, simultaneously exploiting spatial and temporal correlations;

2. the introduction of a novel feature attention mechanism for 3D convolutions that lets the network focus on the most promising high-frequency information largely overcoming main locality limitations of convolutional operations. Moreover, the concurrent use of multiple nested residuals, inside the network, lets low-frequency components flow directly to the output of the model;

3. the conceptualization and development of an efficient, highly replicable, deep learning neural network for MISR that makes use of 2D and 3D convolutions exclusively in the low-resolution space. It has been extensively evaluated on a major multi-frame open-source remote-sensing dataset proving state-of-the-art results by a considerable margin.

The complete code with a pre-trained version of our model is available online [1].

4.1 Methodology

MISR aims at recovering an HR image $\boldsymbol{I}^{\mathrm{HR}}$ from a set of T LR images $\boldsymbol{I}^{\mathrm{LR}}_{[1,T]}$ of the same scene acquired in a certain temporal window. In contrast to SISR, MISR can simultaneously benefit from spatial and temporal correlations, being able to achieve far better reconstruction results theoretically. Either way, SR is an inherently ill-posed problem since a multiplicity of solutions exist for any given set of low-resolution images. So, it is an underdetermined inverse problem, of which

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

the solution is not unique. Our proposed methodology, based on a deep learning model, aims at generating a super-resolved image $\boldsymbol{I}^{\mathrm{SR}}$ applying a function H_{RAMS} to the set of $\boldsymbol{I}^{\mathrm{LR}}_{[1,T]}$ images:

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

In other words, we propose a fully convolutional Residual Attention Multi-Image Super-Resolution (RAMS) network that can efficiently extract high-level features concurrently from T LR images and fuse them exploiting a built-in visual attention mechanism. Attention directs the focus of the model only on the most promising extracted features, reducing the importance of less relevant ones and mostly transcending limitations of the local region of convolutional operations. Moreover, extensive use of nested residual connections lets all the redundant low-frequency information, present in the set $\boldsymbol{I}^{\mathrm{LR}}_{[1,T]}$ of LR images, flow through the network, leaving the model focusing its computation only on high-frequency components. Indeed, high-frequency features are more informative for HR reconstruction, while LR images contain abundant low-frequency information that can directly be forwarded to the final output [283]. Finally, as the majority of the model for SISR [139, 273, 57, 48], all computations in our network are efficiently performed in the LR feature space requiring only an upsample operation at the final stage of the model.

4.1.1 Network architecture

An overview of the RAMS network, with its main three blocks and two branches, is shown in Fig. 4.1. The set of low-resolution images $\boldsymbol{I}^{\mathrm{LR}}_{[1,T]}$ that can be represented as a 4D tensor $\boldsymbol{x}$ with shape $H \times W \times T \times C$ where H, W and C are the height, width, and channels of the single LR images, respectively. The upper global residual path proposes a simple SR solution, making an aware fusion of the T input images. On the other hand, the central branch exploits 3D convolutions residual-based blocks in order to extract spatial and temporal correlations from the same set of T LR images and provides a refinement to the residual simple SR image.

More in detail, in the first part of the main path of the model, we use a simple 3D convolutional layer, with each filter of size $f_h \times f_w \times f_t$, to extract F shallow features from the input. Then, we apply a cascade of N residual feature attention blocks that increasingly extract higher-level features, exploiting local and non-local, temporal, and spatial correlations. Moreover, we make use of a long skip connection for the shallow features and several short skip connections inside each feature attention block to let flow all redundant low-frequency signals and let the network focus on more valuable high-frequency components. The three dimensions H, W and T are always preserved through reflecting padding. The first part of the main branch can be modeled as a single function φ_1 that maps each tensor $\boldsymbol{x}$ to a new higher dimensional one $\boldsymbol{x}_1$ with shape $H \times W \times T \times F$:

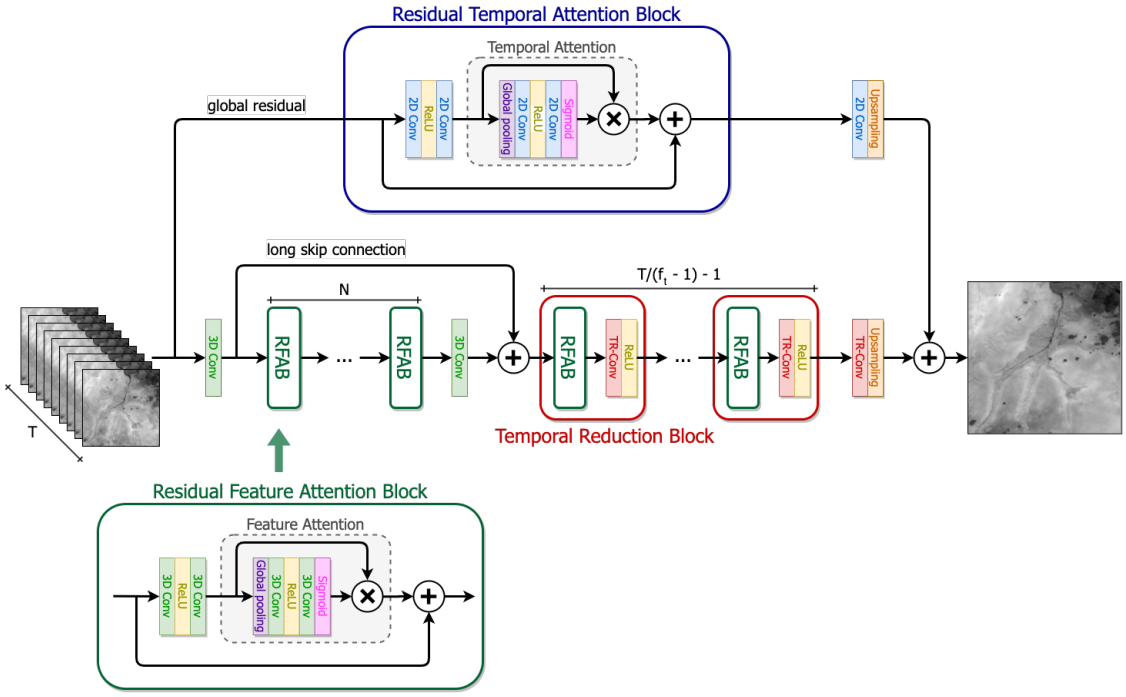


Figure 4.1: Overview of the Residual Attention Multi-Image Super-Resolution Network (RAMS), assuming to work with single-channel LR images ($C = 1$) to simplify the discussion. A tensor of T single-channel LR images constitutes the input of the proposed model. The main branch extracts features, with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

$$\boldsymbol{x_1} = \varphi_1(\boldsymbol{x}) \tag{4.2}$$

In the second part of the main branch, we further process the output tensor with $\lfloor T/(f_t - 1) \rfloor - 1$ temporal reduction blocks. In each block, we intersperse a residual feature attention block with a 3D convolutional layer without padding on the temporal T dimension (TR-Conv). So, H, W and F remain invariant and only the temporal dimension is reduced. The output of this second block is a new tensor $\boldsymbol{x_2}$ with shape $H \times W \times f_t \times F$, where the temporal dimension T is reduced to f_t:

$$\boldsymbol{x_2} = \varphi_2(\boldsymbol{x_1}) \tag{4.3}$$

Finally, the output tensor is processed by a further TR-Conv layer that reduces

the temporal dimension to 1 and an upscale function $H_{\text{UP}|_{3D}}$ that generates a tensor $\boldsymbol{x}_{\text{UP}|_{3D}}$ of shape $sH \times sW \times C$ where s is the scaling factor.

The overall output of the main branch is summed to the trivial solution provided by the global residual. This global path simply weights the T LR images of the input tensor with a residual temporal attention block with filters of size $f_h \times f_w$. Then it produces an output tensor $\boldsymbol{x}_{\text{UP}|_{2D}}$ of shape $sH \times sW \times C$ that is added to the output of the main branch. So, the final SR prediction of the network $\hat{\boldsymbol{y}} = \boldsymbol{I}^{\text{SR}}$ is the sum of the two contribution:

$$\hat{\boldsymbol{y}} = \varphi_{\text{RAMS}}(\boldsymbol{x}) = (\boldsymbol{x}_{\text{UP}|_{3D}} + \boldsymbol{x}_{\text{UP}|_{2D}}) \tag{4.4}$$

The upscaling procedure is identical for both branches; after several trials with different methodologies, such as transposed convolutions [51], bi-linear resizing and nearest-neighbor upsampling [184], we adopted a sub-pixel convolution layer as explained in detail in [224]. So, for either branch, the last 2D or 3D convolution generates $s^2 \cdot C$ features in order to produce the final tensors of shape $sH \times sW \times C$ for the residual sum.

4.1.2 Residual attention blocks

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

Residual feature attention

In this block, used in the main branch, the features are weighted up in order to trace the most promising high-frequency components, and a residual connection lets low-frequency information flow through the network.

Given a generic feature tensor $\boldsymbol{x}_*$, the output of a residual feature attention block is equal to:

$$\varphi_{\text{RFA}}(\boldsymbol{x}_*) = \boldsymbol{x}_* + \varphi_{\text{FA}}(\boldsymbol{x}_\dagger) \cdot \boldsymbol{x}_\dagger \tag{4.5}$$

where φ_{FA} is the feature attention block and $\boldsymbol{x}_\dagger$ is the output of two stacked 3D convolutional layers, with $f_h \times f_w \times f_t$ as kernel dimension and F filters. The first convolutional layer has ReLU activation, while the second is linear.

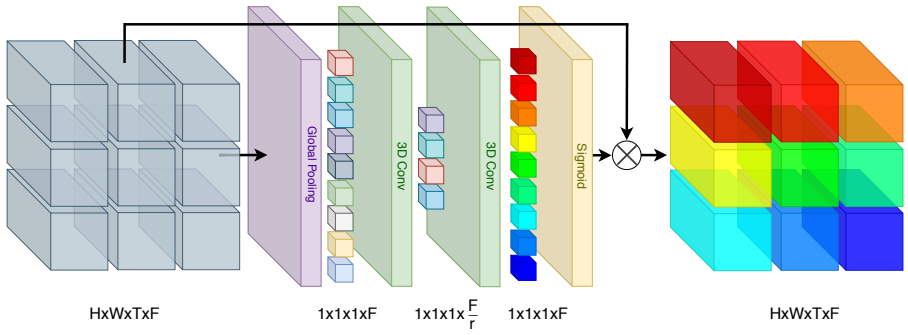


Figure 4.2: Reference architecture of a feature attention block. A series of convolutional operations and non-linear activations are applied to the input tensor with shape $H \times W \times T \times F$ in order to generate different attention statistics for each feature F that concurrently take advantage of local and non-local correlations. Consequently, each feature is properly re-scaled, enabling the network to focus on the most promising components and letting residual connections heed all redundant low-frequency signals.

The feature attention block φ_{FA} extracts a feature descriptor $\boldsymbol{z}_F \in \mathbb{R}^F$ by using a Global Average Pooling (GAP) layer, averaging the spatial and temporal axes (H, F and T). The output tensor $\boldsymbol{z}_F$ is further processed by a stack of two 3D convolutional layers with a ReLU and sigmoid activation function, respectively. The stack of two convolutional layers with the filter of size $1 \times 1 \times 1$ allows to create a non-linear mapping function that is able to deeply capture feature-wise dependencies from the aggregated information extracted by the global pooling operation. The first 3D convolutional layer reduces the feature size by a factor of r, and then the second layer restores the original dimension and constrains its values from zero to one with a sigmoid function in a non-mutually exclusive relationship. These convolutional layers are equivalent to FC layers on the one-dimensional vector $\boldsymbol{z}_F$. Finally, the original tensor $\boldsymbol{x}_\dagger$ is weighted up by the processed attention statistics as shown in Eq. 4.5. Overall, the feature attention block implements a 3D generalization of the Squeeze-and-Excitation (SE) module [99]. Fig. 4.2 shows a graphical representation of the residual feature attention block.

Residual temporal attention

The primary purpose of the global residual path is to generate a starting trivial solution for the upsampling problem. More accurate is this starting prediction, and more simplified is the role of the main branch of the network, leading to a lower reconstruction error. The input of the model $\boldsymbol{x}$ has T different LR images that have to be merged. Intuitively, for each input sample $\boldsymbol{I}^{\text{LR}}_{[1,T]}$, there are some LR images more similar to each other. So, giving them more relevance when merging the T

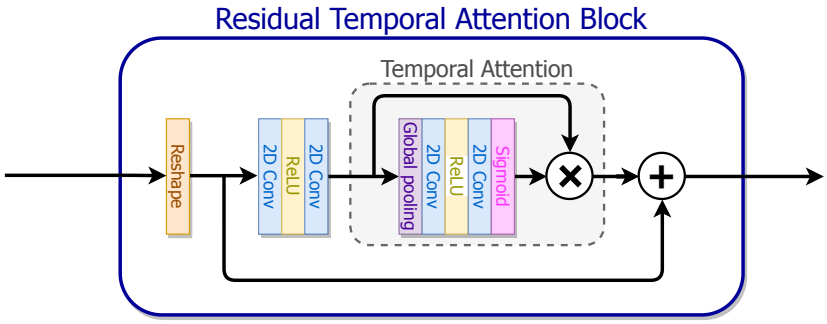


Figure 4.3: Reference architecture of a residual temporal attention block. If the number of channels $C \neq 1$ the input tensor $\boldsymbol{x}$ is reshaped in $H \times W \times (T \cdot C)$. Consequently, all temporal channels are weighted with some statistics computed by the layers of the temporal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

LR images would most probably lead to higher quality predictions. In this context, the aim of the residual temporal attention block is to make an aware weighting of the different input temporal images, letting the network make an upsample solution with primarily the most correlated temporal steps.

That is accomplished with a similar mechanism to the one employed in the residual feature attention blocks and can be summarized as follows:

$$\varphi_{\text{RTA}}(\boldsymbol{x}) = \boldsymbol{x} + \varphi_{\text{TA}}(\boldsymbol{x}_{\ddagger}) \cdot \boldsymbol{x}_{\ddagger} \tag{4.6}$$

where H_{TA} is the temporal attention function and $\boldsymbol{x}_{\ddagger}$ is the product of a stack of two 2D convolutional operations with $f_h \times f_w$ as kernel dimensions and $T \cdot C$ features. Then, in the same way as the feature attention blocks, the temporal block takes the temporal-wise global spatial information into a feature descriptor by using a GAP layer. Finally, those statistical descriptors are processed by a stack of 2D convolutional layers with ReLU and sigmoid as activation functions, respectively, scaling the $T \cdot C$ channels of the input tensor, as shown in Eq. 4.6. As for feature attention blocks, the first convolutional layer reduces the number of the last dimension by a factor of r, giving the network the possibility to fully capture temporal-wise dependencies. This block can be therefore seen as a plain SE module [99] applied to the input tensor image $\boldsymbol{x}$.

4.1.3 Temporal reduction blocks

The aim of the last block of the main branch is to slowly reduce the temporal dimension to one. Indeed, the tensor $\boldsymbol{x_1}$ (see Eq. 4.2) has T temporal dimensions that need to be merged. To this end, we further process the incoming tensors with $\lfloor T/(f_t - 1) \rfloor - 1$ temporal reduction blocks. Each one is composed of a residual feature attention block and a 3D convolutional layer without any reflecting padding

in the temporal dimension, denoted as TR-Conv. So, each TR-Conv layer reduces the temporal dimension of $f_t - 1$. The attention blocks allow the network to learn the best space to decouple image features, selecting more promising features to maintain when reducing the temporal dimension. The output of the last temporal reduction block is the $\boldsymbol{x_2}$ tensor with shape $H \times W \times f_t \times F$ (see Eq. 4.3). The last TR-Conv, before the upsampling function $\varphi_{\text{UP}|_{3D}}$, reduces to one the number of temporal steps and generates $s^2 \cdot C$ features for the sub-pixel convolutional layer [224].

4.1.4 Training process

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

Several loss functions have been proposed and investigated for the SISR problem, such as L_1 [128, 139, 129, 273], L_2 [177, 55, 238, 121] and perceptual and adversarial losses [136, 113]. However, in typical MISR remote-sensing problems, LR images are taken within a certain time window and they could have an undefined spatial misalignment from one to another. So, we must take into account that the super-resolved output of the model $\boldsymbol{I}^{\text{SR}}$ will be inherently not registered with the target image $\boldsymbol{I}^{\text{HR}}$. Moreover, since we can have very different conditions among the images part of the same scene, it is important to make the loss function independent from possible intensity biases between the super-resolved $\boldsymbol{I}^{\text{SR}}$ and the target $\boldsymbol{I}^{\text{HR}}$. Indeed, if we get a super-resolved image $\boldsymbol{I}^{\text{SR}} = \boldsymbol{I}^{\text{HR}} + \epsilon$, with ϵ constant and low enough to avoid numerical saturation, we can consider its reconstruction perfect since it represents the scene with the same level of detail of the ground truth.

With these premises, inspired by the metric proposed in [165], we defined $\boldsymbol{I}^{\text{SR}}_{\text{crop}}$ as the super-resolved output cropped of d pixels on each border and we consider each possible patch $\boldsymbol{I}^{\text{HR}}_{u,v}$, $u, v \in [0{,}2d]$ of size $(sH - 2d) \times (sW - 2d)$ extracted from the ground truth $\boldsymbol{I}^{\text{HR}}$. We compute the mean biases between the cropped $\boldsymbol{I}^{\text{SR}}_{\text{crop}}$ and the patches $\boldsymbol{I}^{\text{HR}}_{u,v}$ as follows:

$$b_{u,v} = \frac{\sum_{i=1}^{sH-2d} \sum_{j=1}^{sW-2d} \left[\boldsymbol{I}^{\text{HR}}_{u,v} - \boldsymbol{I}^{\text{SR}}_{u,v}\right](i,j)}{(sH - 2d)(sW - 2d)} \tag{4.7}$$

The loss is then defined as the minimum mean absolute error (L_1 loss) between $\boldsymbol{I}^{\text{SR}}_{\text{crop}}$ and each possible alignment patch $\boldsymbol{I}^{\text{HR}}_{u,v}$. We use the MAE instead of the most used MSE since we experimentally find that provides better results for image restoration problems, as proved by previous works [285, 139, 283].

$$\mathcal{L} = \min_{u,v \in [0,2d]} \frac{\|\boldsymbol{I}^{\text{HR}}_{u,v} - (\boldsymbol{I}^{\text{SR}}_{u,v} + b_{u,v})\|_1}{(sH - 2d)(sW - 2d)} \tag{4.8}$$

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

4.2 Experiments

In this section, we test the proposed methodology in an experimental context, training it on a dataset of real-world satellite images and evaluating its performance in comparison with other approaches, including a state-of-the-art SISR algorithm, to demonstrate the superiority of Multi-Image models. We first present the dataset and the preprocessing stages, we define all the parameters used during the experimentation, and then we propose quantitative and qualitative results. We also perform an ablation study to demonstrate the contribution of the global residual branch that implements a temporal attention mechanism.

4.2.1 The Proba-V Dataset

To train our model, we exploit the dataset released by the Advanced Concept Team of the European Space Agency (ESA) [165]. This dataset has been specifically conceived for MISR problems, and it is composed of several images taken by the Proba-V satellite in the two different spectral bands RED and NIR (near-infrared). Proba-V satellite was launched by ESA in 2013 and is specifically designed for land covering and vegetation growth monitoring across almost the entire globe. The satellite provides images in two resolutions with different revisit frequencies. HR images have a 100m per pixel spatial resolution and are released roughly every five days, while LR images have a 300m per pixel resolution and are available almost daily. The characteristics of the Proba-V imagery make it particularly suitable for MISR algorithms since it provides both resolutions natively, allowing for the application of the SR process without the need for artificially degrading and downsampling the HR images.

The dataset has been released for the Proba-V Super Resolution challenge [2] and is composed of two main parts: the train part provides both LR and HR images, while the test part LR images, only. In order to verify the effectiveness of our approach, we consider the train part and not the test part, since it has been conceived for the challenge evaluation only and does not include the ground truths. Thus, we subdivide the train part into training and validation sets. To ease the comparison with previous methods, we use the same validation images used in [177]. In total, we have 415 scenes for training and 176 for validation for the RED band and 396 for training and 170 for validation for NIR.

Each scene is composed of several LR images (from 9 to 35, depending on the scene) with a dimension of 128x128 pixels and a single HR ground truth with a

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

dimension of 384x384 pixels. The images are encoded as 16-bit png files, even if the actual signal bit-depth is 14 bits. Additionally, each image features a binary mask that distinguishes reliable pixels from unreliable ones (e.g., due to cloud coverage). This information is vital since the images are not taken in the same weather and temporal conditions, but a maximum period of 30 days can be covered in a single scene. For this reason, non-trivial changes in the landscape can occur between different LR images and their HR counterpart and are essential to understand which pixels carry meaningful information and which do not. Trying to infer the value of pixels that are concealed by clouds would mean being able to predict the weather in an unknown time by merely looking at the condition in other unspecified moments. For this reason, it is essential to train the network so that unreliable pixels do not influence the SR process. To assess the quality of, each image, we define c as the clearance of the image, i.e. the fraction of reliable pixels in the corresponding binary mask.

4.2.2 Data pre-processing

Before training the model, we pre-process the dataset with the following steps, which are common to other reference works present in the literature [177, 51, 58]:

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

Since each LR image is taken at a different time and with some intrinsic spatial misalignment with respect to the others, it is important to spatially resample each pixel value in order to have a coherent reference frame. For each scene of the dataset, we consider a reference image the one with the maximum clearance c. During the registration process, we consider translation as a transformation model, which computes the necessary shifts to register each image for both axes. Masks are taken into consideration during this process in order to avoid bad registration caused by unreliable pixels. The registration is performed in the Fourier domain using normalized cross-correlation as in [189]. After computing the shifts, both LR images and the corresponding masks are shifted accordingly. We use a reflect padding to add pixels to LR images and a constant zero padding for masks. In this way, these extra pixels will be considered unreliable.

For each scene, we must select some LR images in order to match the temporal dimension T of the network. We set a threshold $c_{\text{min}} = 0.85$ on the clearance

for an image to be accepted to avoid using awful images that can worsen the SR performance. The acceptable images are then sorted in order of clearance, and the best T are selected. In the case of a scene with less than T images, we sample randomly from the set of acceptable images until T are reached. If a scene is only composed of clearances under $c_{\min}$, it is entirely removed from the dataset. This selection process is performed after the registration so that heavily bad registered images are also removed, even if they had an initial clearance above the threshold. Since each scene of the dataset contains at least 9 LR images, we set $T = 9$ to fully exploit all the available information for most of the scenes. The selection process results in the deletion of 3 NIR scenes, thus ending up with 415/393 training scenes for RED/NIR and 176/170 validation scenes for RED/NIR.

One of the characteristics of the Proba-V dataset is that the LR images of a particular scene have no clear temporal order. Therefore, there is no reason to prefer a specific order in the T input images to another. The training dataset is, therefore, pre-augmented by performing n_p random temporal permutations of the selected T input images to help generalization. In this way, we can train the algorithm to identify the best temporal image independently of the position inside the input tensor. We set this permutation parameter to $n_p = 7$, reaching a total of 2905 training scenes for RED and 2751 for NIR. The validation split is not pre-augmented in this way, and therefore we keep the original temporal order, ending up with 176 and 170 scenes for RED and NIR, respectively.

Finally, each image is normalized by subtracting the mean pixel intensity value computed on the entire dataset and dividing by the standard deviation. After the forward pass in the network, all the tensors are then denormalized, and the subsequent evaluations are performed on the 16 bits unsigned integer arrays.

4.2.3 Experimental settings

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

In most of the SR applications present in the literature, LR images are obtained

from the artificial degradation of the target HR images. In contrast, the real-world nature of the dataset, in which LR images are obtained independently from HR images, causes an unavoidable misalignment between the super-resolved output and the ground truth. To take into account this problem, the authors of the dataset consider a maximum shift of ±3 pixels on each axis between $\boldsymbol{I}^{\mathrm{SR}}$ and target $\boldsymbol{I}^{\mathrm{HR}}$, computed on the basis of the geolocation accuracy of the Proba-V satellite [165]. When computing the loss function presented in Section 4.1.4, we can therefore set $d = 3$ as the cropping parameter. Besides, since the Proba-V dataset also provides binary masks that mark with one reliable pixel and with 0 unreliable (e.g., concealed by clouds) ones, we adapt the loss function to use this information to refine the training process. During the loss computation, we want pixels marked as unreliable in the target binary mask $\boldsymbol{M}^{\mathrm{HR}}$ not to influence the loss computation. Practically, we can simply mask the cropped super-resolved image $\boldsymbol{I}^{\mathrm{SR}}_{\mathrm{crop}}$ and the HR patch $\boldsymbol{I}^{\mathrm{HR}}_{u,v}$ with the corresponding cropped mask $\boldsymbol{M}^{\mathrm{HR}}_{u,v}$ and average all the quantities over the number of clear pixels. The bias computation is therefore adapted from Eq. 4.7 as:

$$b_{u,v} = \frac{\sum_{i,j}\left[\boldsymbol{I}^{\mathrm{HR}}_{u,v} \cdot \boldsymbol{M}^{\mathrm{HR}}_{u,v} - \boldsymbol{I}^{\mathrm{SR}}_{u,v} \cdot \boldsymbol{M}^{\mathrm{HR}}_{u,v}\right](i,j)}{\|\boldsymbol{M}^{\mathrm{HR}}_{u,v}\|_1} \tag{4.9}$$

In the same way, the loss is adapted from Eq. 4.8 as:

$$\mathcal{L} = \min_{u,v\in[0,6]} \frac{\|\boldsymbol{I}^{\mathrm{HR}}_{u,v} \cdot \boldsymbol{M}^{\mathrm{HR}}_{u,v} - (\boldsymbol{I}^{\mathrm{SR}}_{u,v} \cdot \boldsymbol{M}^{\mathrm{HR}}_{u,v} + b_{u,v})\|_1}{\|\boldsymbol{M}^{\mathrm{HR}}_{u,v}\|_1} \tag{4.10}$$

To train the model, we extract from each LR training and validation scene 16 patches with a size of 32 × 32 pixels and the corresponding HR and mask patches with a size of 96 × 96. We further check every single patch and remove those that have a target mask M^{HR} with less than 0.85 clearance c. The total number of training samples obtained is 41678 for RED and 40173 for NIR. The total number of validation samples obtained is 2621 for RED and 2478 for NIR. During the training process, we further perform online data augmentation with random rotations of 90°, 180° and 270° and random horizontal flips.

We set the batch size to 32. We optimize the loss function with Adam algorithm [122] with default parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-7}$. We set an initial learning rate $\eta_{\mathrm{i}} = 5 \times 10^{-4}$ and we reduce it with a linear decay down to $\eta_{\mathrm{f}} = 5 \times 10^{-7}$. We train two different networks for RED and NIR spectral bands on a workstation with an Nvidia RTX 2080Ti GPU with 11GB of memory and 64GB of DDR4 SDRAM. We use the TensorFlow 2 [1] framework with CUDA 10. In total, we train the models for 100 training epochs for about 16 hours.

4.2.4 Quantitative results

To evaluate the obtained results, we need to use a slightly modified version of Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) [259] criteria to take into consideration all the aspects we considered in the previous section to obtain a proper loss function. Martens et al. [165] propose a corrected version of the PSNR, called cPSNS, that is obtained from a corrected mean squared error (cMSE). The computation of the cMSE is performed in the same way as we did for the loss in Eq. 4.10: it is the minimum MSE between $\boldsymbol{I}^{\text{SR}}_{\text{crop}} + b_{u,v}$ and the HR patches $\boldsymbol{I}^{\text{HR}}_{u,v}$:

$$\text{cMSE} = \min_{u,v \in [0,6]} \underset{\text{clear}}{\text{MSE}}_{u,v} \tag{4.11}$$

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

$$\underset{\text{clear}}{\text{MSE}}_{u,v} = \frac{\|\boldsymbol{I}^{\text{HR}}_{u,v} \cdot \boldsymbol{M}^{\text{HR}}_{u,v} - (\boldsymbol{I}^{\text{SR}}_{u,v} \cdot \boldsymbol{M}^{\text{HR}}_{u,v} + b_{u,v})\|_2^2}{\|\boldsymbol{M}^{\text{HR}}_{u,v}\|_1} \tag{4.12}$$

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

$$\text{cPSNR} = 10 \log_{10} \frac{(2^{16}-1)^2}{\text{cMSE}} = \max_{u,v \in [0,6]} 10 \log_{10} \frac{(2^{16}-1)^2}{\underset{\text{clear}}{\text{MSE}}_{u,v}} \tag{4.13}$$

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

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

$$\text{cSSIM} = \max_{u,v \in [0,6]} \text{SSIM}\Big(\boldsymbol{I}^{\text{HR}}_{u,v} \cdot \boldsymbol{M}^{\text{HR}}_{u,v}, \boldsymbol{I}^{\text{SR}}_{\text{crop}} \cdot \boldsymbol{M}^{\text{HR}}_{u,v} + b_{u,v}\Big) \tag{4.14}$$

Temporal self-ensemble (RAMS+)

As in Section 4.2.2, during the training process images are augmented with random permutation in the temporal axis. For this reason, it is possible to maximize the performance of the model, by adopting a self-ensemble mechanism during inference, similarly to what was done in previous SR works [139, 240, 283]. For each input scene, we consider a certain number P of random permutations on the temporal axis and we denote as $\left\{\boldsymbol{I}^{\text{LR}}_{[1,T],0}, \cdots, \boldsymbol{I}^{\text{LR}}_{[1,T],P}\right\}$ the resulting set of inputs. The output of the inference process is therefore the average of the predictions on the

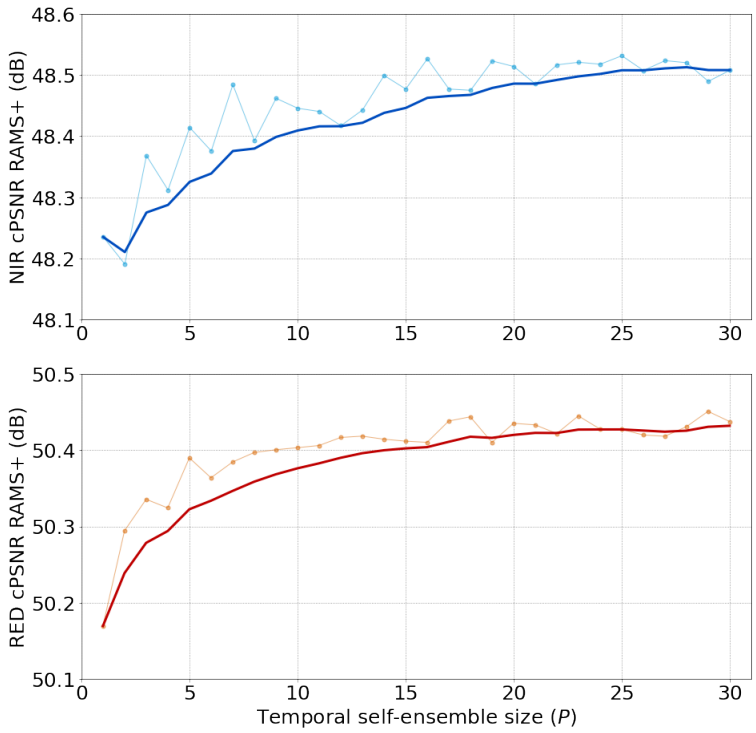


Figure 4.4: Results with a temporal self-ensemble of size P. The highlighted curves represent an exponential moving average of the results to clearly show the trend. The values for $P = 1$ are equivalent to RAMS.

whole set. We call this methodology RAMS+$_P$, where P is the number of random permutations performed:

$$\boldsymbol{I}^{\mathrm{SR}} = \frac{1}{P} \sum_{i=1}^{P} \varphi_{\mathrm{RAMS}} \left(\boldsymbol{I}^{\mathrm{LR}}_{[1,T],\,i} \right) \tag{4.15}$$

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

Band	NIR		RED	
Metric	cPSNR	cSSIM	cPSNR	cSSIM
Bicubic	45.12	0.9767	47.63	0.9846
IBP[105]	45.96	0.9796	48.21	0.9865
BTV[66]	45.93	0.9794	48.12	0.9861
RCAN[283]	45.66	0.9798	48.22	0.9870
VSR-DUF[112]	47.20	0.9850	49.59	0.9902
HighRes-net[51]	47.55	0.9855	49.75	0.9904
3DWDSR [58]	47.71	0.9864	49.83	0.9909
DeepSUM[177]	47.84	0.9858	50.00	0.9908
DeepSUM++[178]	47.93	0.9862	50.08	0.9912
RAMS (ours)	**48.23**	**0.9875**	**50.17**	**0.9913**
RAMS+$_{20}$ (ours)	**48.51**	**0.9880**	**50.44**	**0.9917**

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

Comparison with state-of-the-art methods

Tab. 4.1 shows the comparison of cPSNR and cSSIM metrics with several methods on the validation set. We consider as the baseline the bicubic interpolation of the best image of the scene selected considering the clearance, i.e., the number of clear pixels as marked by the binary masks.
IBP [105] and BTV [66] methods are tested with the same methodology presented in Molini et al. [177]. They achieve slightly better results than the baseline with both metrics.
RCAN [283] is one of the SISR state-of-the-art networks. We trained from scratch two models, one for each spectral band, setting $G = 5$ and $B = 5$, as the number of residual groups and residual channel attention blocks respectively, for a total of about 2M parameters. We train the two models from scratch on the Proba-V dataset, selecting the best image per scene as input. RCAN shows better performance with respect to classical methods but is far beyond the other MISR networks, showing how the additional information coming from both spatial and temporal correlations is vital to boost the SR process.

VSR-DUF [112] has been developed to upsample video signals using a temporal window of several frames. We train two models from scratch, one for each spectral band, using 9 LR images as input as in our methodology. The authors consider three different architectures depending on the number of convolutional layers and find better results, increasing the depth of the model. We select the baseline 16 layers deep architecture, which already has more than double parameters with respect to RAMS, with the same number of input images.

HighRes-net [51] algorithm got second place in the Proba-V challenge and featured a single network for both spectral bands that recursively reduce the temporal dimension to fuse the input LR images. We train the model on our training dataset with default architecture. For a fair comparison, since the authors constrained the model to have an input temporal dimension multiple of 2, we set it to 16, as it is closest to our temporal size of 9.

3DWDSR [58] is an adaptation of the SISR WDSR network [273] to MISR. It implements 3D convolutional layers maintaining the block structure of the original network and a global residual branch with bicubic up-sampling. We train two separate models for NIR and RED bands for 200 epochs using the default parameters and our training dataset.

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

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

It is worth mentioning that our methodology reaches a result of 0.93368 on the test set of the Proba-V challenge as provided by the official site and places at the top of the leaderboard available after the end of the official challenge[3] as of January 2021. This score is computed as the mean ratio between the cPSNR values of the challenge baseline on each testing scene, and the corresponding submitted cPSNR for both spectral bands. This result has been obtained by retraining the two networks with both training and validation datasets together.

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

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

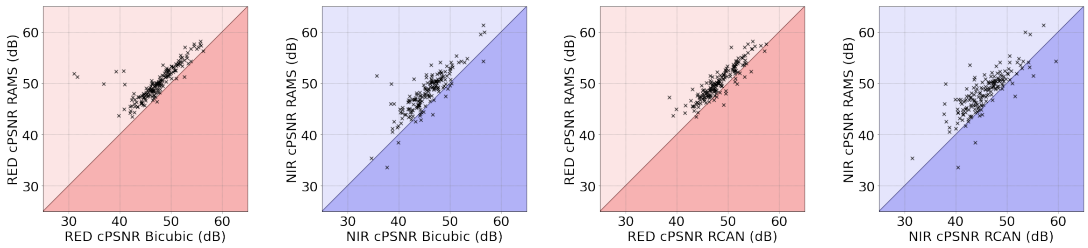


Figure 4.5: cPSNR comparison between RAMS and bicubic interpolation and RAMS and RCAN(SISR) on the validation set. Each data point represents a scene of the dataset: when a cross is above the line, the corresponding scene is reconstructed better by RAMS.

	without RTA		**with RTA**	
	cPSNR	cSSIM	cPSNR	cSSIM
NIR	47.96	0.9869	48.23	0.9875
RED	47.98*	0.9863*	50.17	0.9913

Table 4.2: RAMS results with and without RTA (residual temporal attention) branch. Values for RED without RTA are computed with the last valuable parameters before training diverges.

92% for RED, and 91% for NIR. That is reasonable since RCAN results are in some way in the middle between bicubic and RAMS.

Importance of the residual temporal attention branch

As a final analysis, we perform an ablation study to demonstrate the importance of the global residual branch that implements a temporal attention mechanism. We train two alternative networks, one for each spectral band, that have the same architecture of RAMS, except that we delete the residual temporal attention branch. These reduced networks are trained from scratch independently from the complete ones.

Tab. 4.2 shows a significant drop in the results obtained without the global residual branch and demonstrates the importance of selecting the best temporal views to ease the SR process of the main branch. We find this difference particularly relevant for the RED band, since the training repeatedly failed without the RTA branch, with a diverging behavior after some epochs. The result reported in the table is computed with the last valuable parameters before the divergence starts.

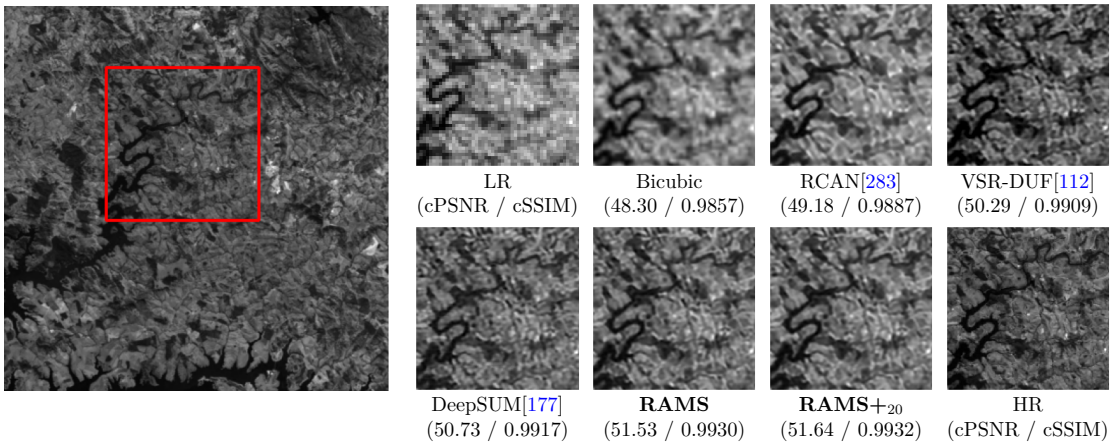


Figure 4.6: Qualitative comparison between different methods on RED imgset0302.

4.2.5 Qualitative results

A visual comparison between some of the methods taken in the exam is shown in Fig. 4.6 and 4.7 for a RED and NIR image respectively. We provide a zoomed patch of the best LR input image of the scene, its bicubic interpolation, and the inference output of RCAN, VSR-DUF, DeepSUM, RAMS and RAMS+$_{20}$, together with the target HR ground truth. cPSNR and cSSIM scores for the image under analysis are also provided. From this comparison, MISR methods clearly show a better performance with respect to bicubic and SISR (RCAN). However, it is not trivial to understand which method is better among MISR algorithms with a visual inspection of the results, only. As found by Ledig et al. [136], the task of achieving pleasant-looking results is a different optimization problem from maximizing the fidelity of the reconstructed information. Therefore, results with high content-related metrics such as PSNR and SSIM frequently appear less photo-realistic to the human eye. However, in the context of remote sensing, the fidelity of the content of the pixels is vital to ensure that the super-resolved images are meaningful, thus the quality of results should be inferred by using content-related metrics, rather than by visual inspection.

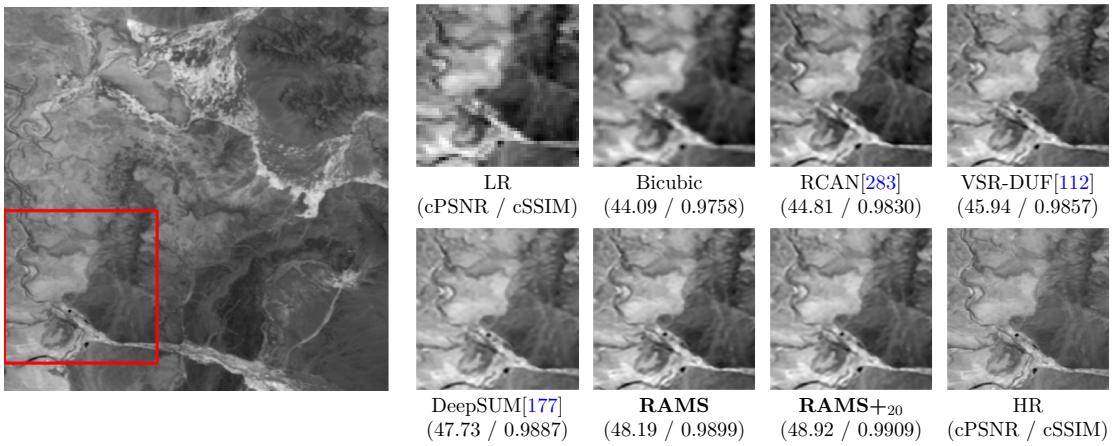


Figure 4.7: Qualitative comparison between different methods on NIR imgset0596.

Chapter 5

Waypoint Generation in Row-based Crops with Deep Learning and Contrastive Clustering

The autonomous navigation problem for robotics agents is a crucial task to be solved to enable the full potential of robotic systems for precision agriculture. In recent years, some approaches to solve this problem have been proposed in the literature. In particular, for the specific case of row-based crops, many works proposed local planners combining deep learning with computer vision [5, 6] or other sensor processing methods [202, 16, 18]. However, local planners provide a solution for intra-row navigation only, and therefore a global path generator is always needed. In a complex scenario such as a row-based environment, where traversing each row is the practical navigation goal, the problem of developing an efficient global path planner has been quite neglected by the research community. Existing solutions usually tackle the problem by clustering visual data obtained from satellites or UAVs. For example, in [288] authors use a classical clustering method to identify vineyard rows from a 3D model of the terrain reconstructed from UAV data and then compute the path accordingly. However, as pointed out in [251], the extraction of relevant information about row geometry from images can be a complex task, in addition to being extremely computationally expensive. This limitation also holds considering other approaches besides clustering. For instance, in [45] authors adopted 3D point cloud aerial photogrammetry to detect the structure of vineyards.

To this end, we propose the DeepWay method to efficiently combine DL and clustering for the generation of start and end row waypoints given an occupancy

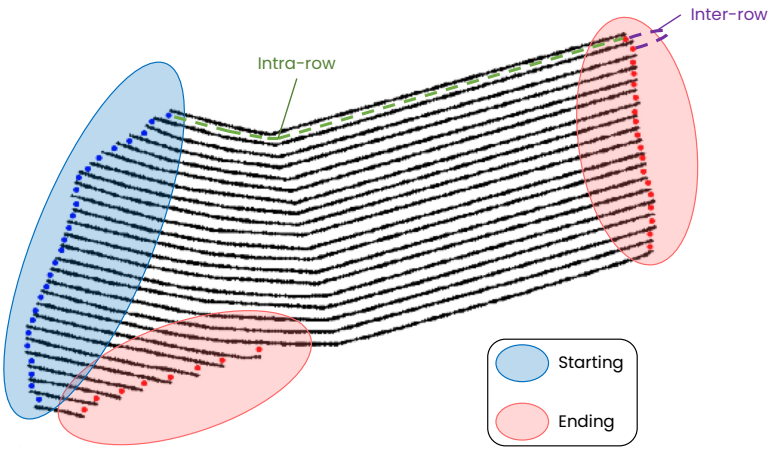


Figure 5.1: Every row-based crop can be seen as a set of lines or curves that identify two regions comprising the starting and ending points of each row. It is possible to plan a full-coverage path in the row-based environment simply by alternating intra-row segments, that connect the starting region to the ending region, and inter-row segments, that connect two starting or two ending points of consecutive rows.

grid of the vineyard. As input, it requires just an occupancy grid of the considered parcel and provides, as output, the waypoints to be used to plan a global path to cover each row of the considered crop. Furthermore, we enhance the original DeepWay method in order to simultaneously predict the position of the navigation waypoints for each row and cluster them in a single forward step. Hence, we train our model with an additional contrastive loss. The deep neural network is trained on a carefully devised synthetic dataset of both straight and curved occupancy grids and tested on manually-labeled real satellite images. Our extensive experimentation demonstrates that the proposed solution successfully predicts precise waypoints also in real-world crop maps. We also consider complex conditions such as both curved and straight rows. All of our training and testing code and data are open source and publicly available [1].

5.1 Methodology

Due to its intrinsic nature, every row-based crop is characterized by a set of lines or curves that identify two regions comprising the starting and ending points

[1]There are two repositories related to this work:

- https://github.com/fsalv/DeepWay for the original DeepWay implementation
- https://github.com/fsalv/ClusterWay for the enhanced version

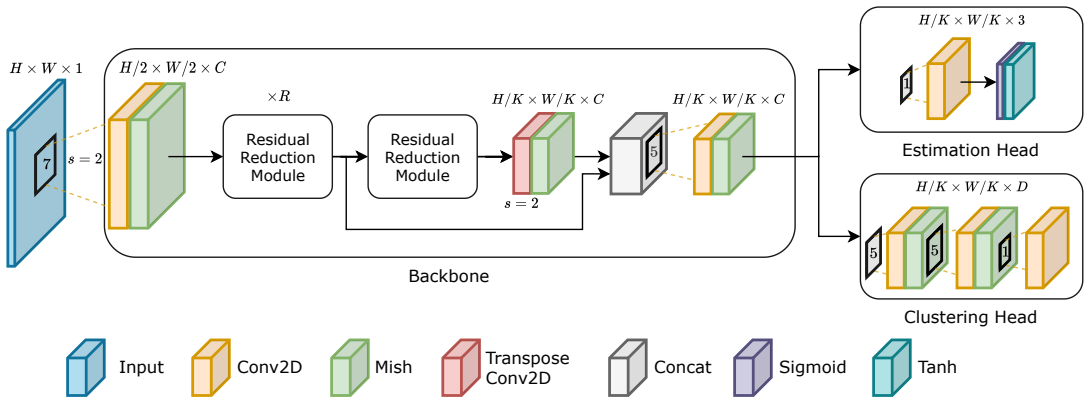


Figure 5.2: Architecture of the backbone and the two regression heads. The number of residual reduction modules in the main block R determines the backbone compression factor $K = 2^{R+1}$.

of each row, respectively. In this scenario, a robotic path should cover the whole field, and it can be divided into intra-row segments, that connect the starting region to the ending region, and inter-row segments, that connect two starting or two ending points. As shown in Fig. 5.1, given an optimal estimation of these starting and ending waypoints, it is possible to plan a full-coverage path in the row-based environment simply by alternating intra-row and inter-row segments. Therefore, the planning process heavily relies on two main steps: waypoint estimation, which identifies candidates for the points of interest, and waypoint clustering, which assigns each estimated point to one of the two regions.

We frame the waypoint generation process as a regression problem, in which we estimate the coordinates of the points with a deep neural network, starting from a top-view map of the environment. The map consists of a 1-bit single-channel occupancy grid that identifies with 1 the plant rows and with 0 the free terrain. Therefore, this kind of estimation process can be easily applied to geo-referenced segmented masks of the target fields obtained from satellites or UAV imagery. The waypoints and the planned path can then be converted from the image reference system to a Global Navigation Satellite System (GNSS) reference frame to be used in real-world navigation. To tackle waypoint clustering, we first propose a method based on DBSCAN [63] with the successive merging of subclusters in order to reach the final starting and ending groups. In addition, we propose a supervised approach based on a contrastive loss to perform point assignment. Therefore, the proposed model simultaneously performs both estimation and clustering with a single forward pass, without the need for complex post-processing operations based on heuristic geometrical-based rules.

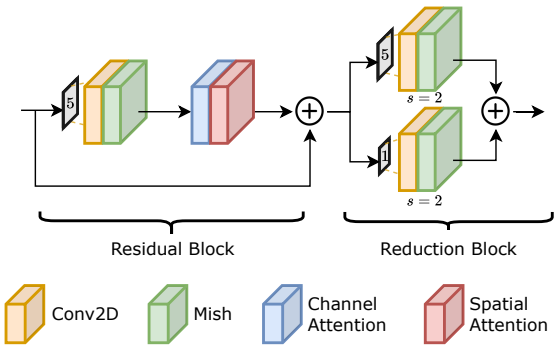


Figure 5.3: Residual reduction module architecture. The channel and spatial attentions are implemented as in [262].

5.1.1 Backbone design

We implement the model as a convolutional neural network characterized by a feature extraction backbone, followed by two specialized heads. A head is responsible for the estimation task, while the other deals with clustering.

The basic block of the network is the residual module, characterized by a stack of a 2D convolution and spatial and channel attention [262]. Each residual block is followed by a reduction module characterized by convolutions with stride 2 that progressively halve the spatial dimensions. The backbone is a stack of R residual reduction modules, made by combining a residual module and a reduction module. The final part of the network is made by an additional downsampling block, followed by a transposed convolution upsampling stage, all arranged in a residual fashion. This combination of compression and expansion has been proven very effective for different computer vision tasks such as segmentation [205] and representation learning [243]. Overall, the model performs a dimensionality compression of a factor of 2^{R+1}, where R is the number of residual reduction modules in the main block. As main activation function, we use Mish [176] in all the network blocks. We select this activation after experimentally observed a better convergence. The complete backbone structure is detailed in Fig. 5.2 and Fig. 5.3.

5.1.2 Waypoint estimation

The waypoint estimation is framed as a regression problem, similarly to object detection approaches in computer vision [197]. In particular, given an input occupancy grid map $\boldsymbol{X}$ with dimensions $H \times W$, we subdivide it into a grid of $K \times K$ cells. Each cell is responsible for predicting the confidence c that a waypoint falls in that region, as well as its relative horizontal and vertical displacements with respect to the cell center $\boldsymbol{\Delta} = (\Delta x, \Delta y)$. The displacements are defined in the range $[-1, +1]$ and represent a shift relative to half of the cell dimension, with

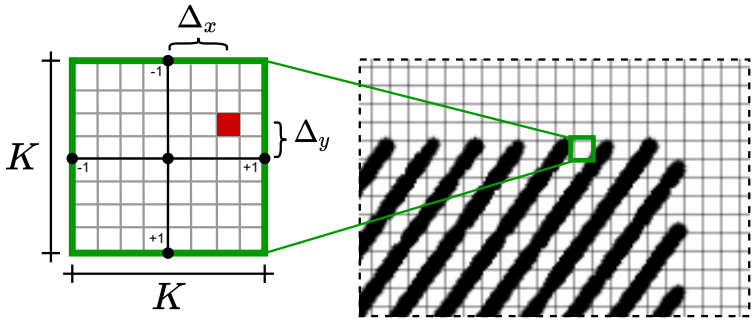


Figure 5.4: The input occupancy grid is subdivided into a grid of $K \times K$ cells. For each cell, the waypoint estimation head outputs the confidence c of a waypoint presence, as well as the relative horizontal and vertical displacements with respect to the cell center $\boldsymbol{\Delta} = (\Delta x, \Delta y)$.

-1 identifying the left/top borders and $+1$ the right/bottom ones. An example of prediction with its corresponding displacements is shown in Fig. 5.4. Given a prediction $\hat{\boldsymbol{p}}_{\mathbf{out}} = (\hat{x}_{\text{out}}, \hat{y}_{\text{out}})$ in the output reference frame, the waypoint coordinates in the input reference frame $\hat{\boldsymbol{p}}_{\mathbf{in}}$ can be reconstructed with the following equation:

$$\hat{\boldsymbol{p}}_{\mathbf{in}} = \hat{\boldsymbol{p}}_{\mathbf{out}}\ K + \frac{K}{2} + \boldsymbol{\Delta}\frac{K}{2} \tag{5.1}$$

The waypoint estimation head maps the high-level features extracted with the backbone to the output space with a 1x1 convolution. The backbone compression factor 2^{R+1} corresponds to the grid dimension K. Therefore, the output tensor of the estimation branch has a dimension of $H/K \times W/K \times 3$. We apply a sigmoid activation to the probability output and a tanh activation to the displacement outputs, in order to map each tensor to the corresponding output range. We optimize the network for the waypoint estimation task with a weighted mean squared error loss. For each output cell $\boldsymbol{u_{i,j}}$, the estimation loss is therefore computed as:

$$l_{i,j}^{\text{est}} = \mathbb{1}_{i,j}^{\text{wp}} \lambda \|\boldsymbol{u_{i,j}} - \hat{\boldsymbol{u}}_{\boldsymbol{i,j}}\|_2 + (1 - \mathbb{1}_{i,j}^{\text{wp}})(1 - \lambda)\|\boldsymbol{u_{i,j}} - \hat{\boldsymbol{u}}_{\boldsymbol{i,j}}\|_2 \tag{5.2}$$

where $\mathbb{1}_{i,j}^{\text{wp}} \in \{0,1\}$ is an indicator Boolean function evaluating 1 if a waypoint is present in that cell, and λ is the relative constant that weights differently positive and negative cells.

At inference time, we get the list of predicted waypoints by considering all the cells with confidence c over a certain threshold t_c. As in standard object detection methodologies, we also apply a suppression algorithm to decrease the number of redundant predictions that typically occur when multiple adjacent cells detect the same waypoint. The algorithm identifies all the groups of predictions with Euclidean distance within a certain threshold t_{sup} in the input reference frame. For

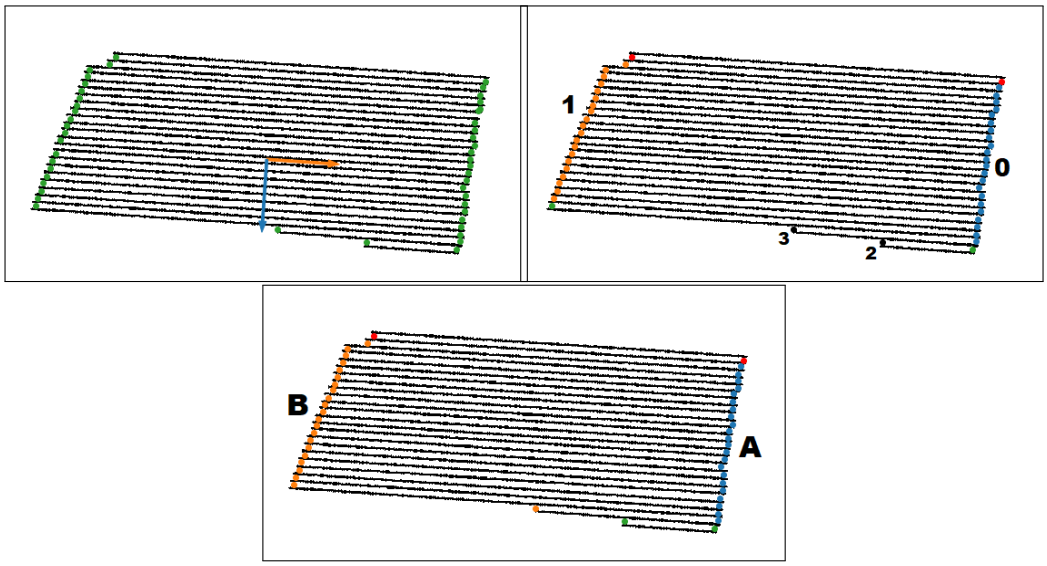


Figure 5.5: Classical solution for waypoints refinement and clustering process. Firstly, the row angle is estimated. Then, the predicted waypoints are clustered with the DBSCAN algorithm [63], that identifies two main groups (**0** in blue and **1** in orange) and two unclustered waypoints (**2** and **3** in black). The groups' starting points are in red, the ending points in green. Then, we iteratively merge the groups into the two principal clusters **A** and **B**, and we obtain the final order of the predicted waypoints.

each group, the point with highest confidence c is selected, while the remaining predictions are discarded.

5.1.3 Waypoint clustering

Once the waypoints are detected, they should be assigned to starting or ending regions. This task can be seen as a simple binary classification, in which the labels represent the two clusters. However, in this scenario, the actual assigned label is not relevant, as the only fundamental aspect is whether points of the same group are assigned the same label. The aim is to discriminate the points of the two regions without caring about which of them is classified as starting or ending. Indeed, an optimal path can be successfully planned regardless of the choice of the starting cluster. This invariance cannot be guaranteed by supervised classification.

Classical clustering

We first design a postprocessing and waypoint clustering methodology based on heuristic geometrical-based rules, that is particularly effective in handling the straight row condition. For this solution, we only use the waypoint estimation head

of the proposed model. We cluster the predicted points using the density-based clustering algorithm DBSCAN [63] (see Section 2.1.2). This approach allows to automatically cluster together points that are close to each other and can give a first subdivision of the waypoints into main groups. Depending on the geometry of the field, several clusters can be found in this way, and some points can remain unclustered, in particular for rows drastically shorter with respect to the others. To get the order of the waypoints inside each group, we project each of them along the perpendicular to the rows, and we sort them in this new reference system. This approach works well with straight rows that have a constant orientation along the whole field.

The row angle is estimated with the progressive probabilistic Hough transform technique [168]. This algorithm is a classic computer vision feature extraction method, able to detect lines in an image and return an estimate of starting and ending points. Even though this algorithm may seem enough to solve the whole problem of finding the target waypoints in the mask without the need for a neural network, this approach is too dependent on a number of hyper-parameters that cannot be well-defined a-priori and generally is not able to cope with holes and irregularities which are inevitably present in real-world field occupancy grids. We experimentally find that the application of this method leads to a high number of false-positive and false-negative detections of lines on all the considered datasets. However, we still use it to estimate the row angle by averaging the orientations of each couple of detected points. In the case of a complete failure of this approach, which can happen with the most complex masks, we estimate the angle using a probabilistic iterative process that minimizes the number of intersections with the rows starting from points close to the image center.

After ordering the points inside each cluster, we adopt a refinement approach to insert possible missing waypoints or delete duplicated ones, by counting the number of rising and falling edges in the mask along the line connecting two consecutive points. Then, to get the final order, the different groups must be merged into the two main clusters representing the beginning and the ending points of each row. We adopt a strategy to iteratively assign clusters to the groups considering their size and the values of their projections along the perpendicular to the rows. We assume that a good assignment is one that spans the same interval along the projection axis on both groups with different clusters. After the assignments, we refine the borders between the merged clusters, in order to compensate for possible mispredicted points. Once we get the final groups, we get the final order by considering a pattern A-B-B-A. Every intra-row connection is performed by checking possible intersections with the rows and correcting the order consequently. If there is a missing point in one of the two groups even after the waypoints refinement process, we remain within the same group, avoiding any possible intersection with the rows. In this way, we put the focus on building feasible paths in the field.

Algorithm 1: Classical waypoint clustering pipeline

input : Occupancy grid $\boldsymbol{X}$ of size $H \times W$
output: Waypoint clusters A,B

```
wp ← DeepWay(X);
α ← angle_estimation(X);
clusters ← DBSCAN(wp);
foreach cluster c do
    proj ← project(wp|_c, α);
    order ← sort(proj);
    wp|_c ← refine(wp|_c, order, X);
end
A,B ← [];
while wp is not empty do
    A,B ← assign(wp, α, A, B);
end
```

All these assumptions work well for straight crops only, since they consider a constant orientation of the rows. Fig. 5.5 shows all the operations performed during the waypoints refinement and clustering process. The full pipeline of the proposed approach is presented in Alg. 1.

Contrastive clustering

As already stated, the classical solution makes strong assumptions on the geometry of the crops and can only work for straight rows. To extend the approach to more generic curved crops, we model the clustering problem as a supervised representation learning process implemented in the neural network by the waypoint clustering head. Given the two sets of points $A = \{\boldsymbol{p} \mid \boldsymbol{p} \in \text{first cluster}\}$ and $B = \{\boldsymbol{p} \mid \boldsymbol{p} \in \text{second cluster}\}$, we want to find a non-linear mapping $f(\cdot)$ such that

$$d\Big(f(\boldsymbol{p_i}), f(\boldsymbol{p_j})\Big) \ll d\Big(f(\boldsymbol{p_i}), f(\boldsymbol{p_k})\Big) \quad \text{for} \quad \boldsymbol{p_i}, \boldsymbol{p_j} \in A \ , \ \boldsymbol{p_k} \in B \tag{5.3}$$

and vice versa, where d is a distance measure. In the latent space mapped by $f(\cdot)$, points of different clusters are well-separated according to distance d. This means that a simple clustering method such as K-means [157] (see Section 2.1.1) can successfully discriminate the two groups in the latent space, as shown in Fig. 5.6. Inspired by the contrastive framework used for unsupervised learning in [39], we select as distance metric d the inverse of the cosine similarity:

$$\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \frac{\boldsymbol{u}^\top \boldsymbol{v}}{\|\boldsymbol{u}\|_2 \ \|\boldsymbol{v}\|_2} \tag{5.4}$$

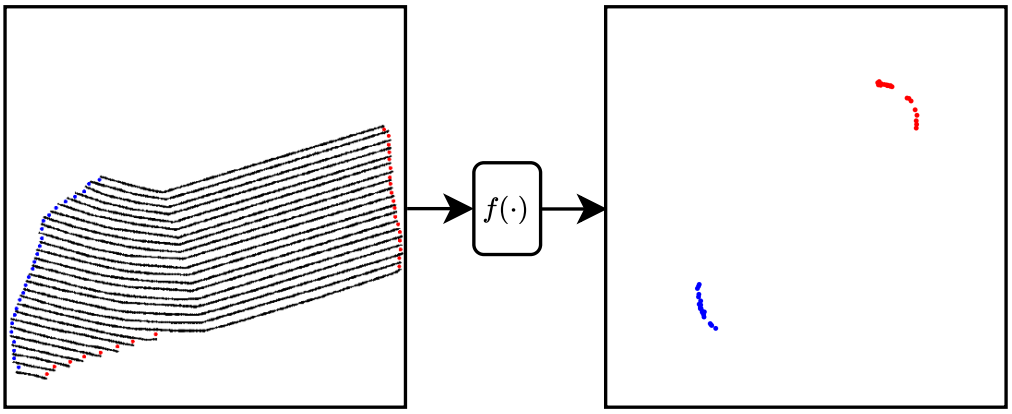


Figure 5.6: In the latent space mapped by $f(\cdot)$, points of the same cluster appear closer together with respect to points of the other cluster. The mapping function $f(\cdot)$ is implemented with the backbone and the clustering head together. In this example, the latent space has a dimensionality $D = 2$.

For each image, we consider the N ground-truth waypoints as independent samples. Given a point $\boldsymbol{p_i}$, we consider as positive examples all the other $N/2-1$ points in the same cluster, and as negative examples the $N/2$ points of the other cluster. Therefore, we define the clustering loss contribution for the sample i as:

$$l_i^{\text{clus}} = -\frac{1}{N-1}\sum_{\substack{j=1\\ j\neq i}}^{N}\Bigg[\mathbb{1}_{\substack{\boldsymbol{p_i},\boldsymbol{p_j}\in A\\ \vee\, \boldsymbol{p_i},\boldsymbol{p_j}\in B}}\log\Big(\text{sig}\Big(\text{sim}\Big(f(\boldsymbol{p_i}),f(\boldsymbol{p_j})\Big)\Big)\Big) + \\ + \Big(1-\mathbb{1}_{\substack{\boldsymbol{p_i},\boldsymbol{p_j}\in A\\ \vee\, \boldsymbol{p_i},\boldsymbol{p_j}\in B}}\Big)\log\Big(1-\text{sig}\Big(\text{sim}\Big(f(\boldsymbol{p_i}),f(\boldsymbol{p_j})\Big)\Big)\Big)\Bigg] \tag{5.5}$$

where $\mathbb{1}_{\substack{\boldsymbol{p_i},\boldsymbol{p_j}\in A\\ \vee\, \boldsymbol{p_i},\boldsymbol{p_j}\in B}} \in \{0,1\}$ is an indicator function evaluating 1 if $\boldsymbol{p_i}$ and $\boldsymbol{p_j}$ are in the same cluster and 0 otherwise, while 'sig' represents the sigmoid function. Basically, this loss computes the binary cross-entropy of the cosine similarity in the latent space mapped by $f(\cdot)$ for the pair $(\boldsymbol{p_i}, \boldsymbol{p_j})$. $f(\cdot)$ is optimized to push the cosine similarity towards the maximum +1 if the points are in the same cluster and towards the minimum -1 otherwise. The final loss is computed over all the pairs (i, j) as well as (j, i) for each input image. This loss can be seen as a variation of the one used in [264, 248, 39], but instead of N groups with 2 elements each, optimized with categorical cross-entropy and softmax, we have 2 groups with $N/2$ elements each, optimized with binary cross-entropy and sigmoid.

The mapping $f(\cdot)$ is modeled by the clustering head in the output space reference system. This means that, after selecting the predicted waypoints over the probability threshold t_c with the first channel of the estimation head, we simply select the mapped latent space features from the corresponding head tensor using the

output coordinates for each point. The head is composed of two convolutional layers with Mish activation [176] and one final 1x1 convolution with linear activation. The output tensor of the clustering branch has a dimension of $H/K \times W/K \times D$, where D is the latent space dimensionality.

At inference time, for each waypoint detected in the estimation phase, we select the corresponding feature from the clustering head output. We can predict the clustering assignment by fitting a K-means predictor with two centroids on the selected features. Since we use cosine similarity in the loss computation, we are optimizing the clustering in the normalized latent space. For this reason, the features should be divided by their Euclidean norm before clustering. This normalization decreases by one the latent space dimensionality, and therefore the minimum number of dimensions D for the clustering head is 2.

5.2 Experimental Setting

In this section, we present all the details of our experimentation. We describe the datasets used for network training and testing as well as the main hyperparameters adopted during the training phase.

5.2.1 Dataset description

Considering the lack of open datasets of row crops bird-eye maps and the time required to manually annotate a large set of real images, we define a method to build realistic synthetic occupancy grids to train the model. The proposed method is able to generate both straight and curved crop maps. The generation process can be summarized as follows:

1. sample a uniformly random number of rows $n \in$ [10,50] and angle $\alpha \in [-\pi/2, \pi/2]$;
2. generate row centers with a random inter-row distance, along the line perpendicular to α and passing through the image center;
3. generate random field borders and find starting and ending points for each row with orientation α;
4. to create curved maps, add a random displacement to the row centers, and compute a quadratic Bézier curve with the starting, ending and center points as control points; this ensures that the curves are continuous and smooth;
5. generate the occupancy grid by drawing filled circles with random radius $r \in$ [1,2] pixels to model irregularities in the row width

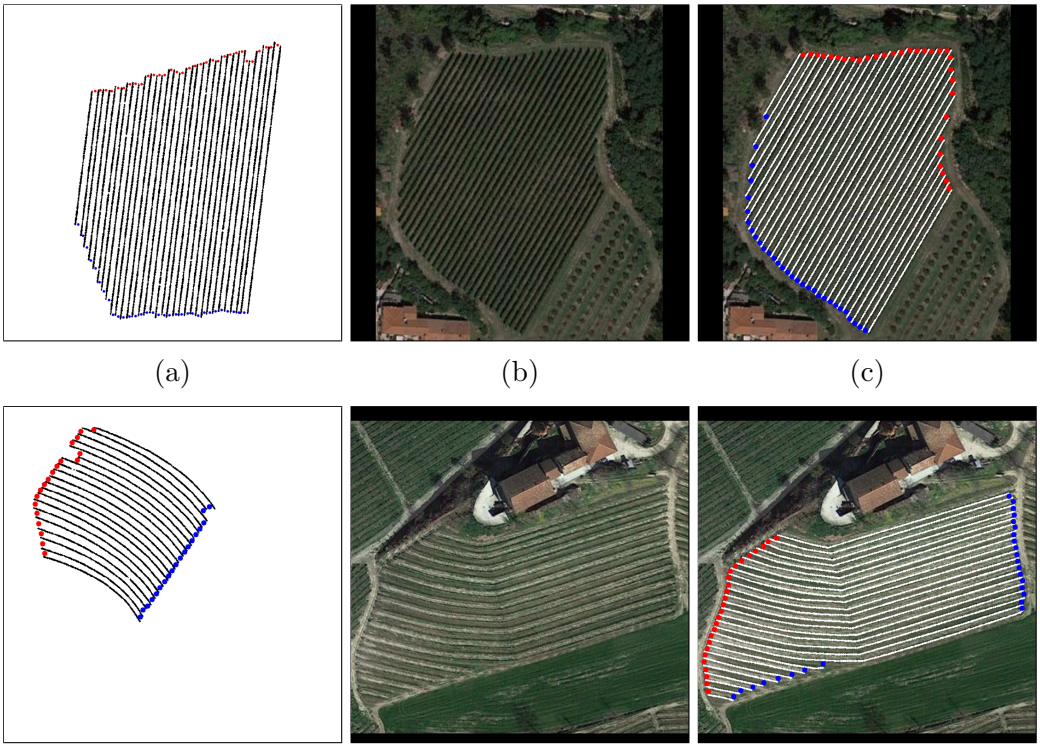


Figure 5.7: Examples of straight (top) and curved (bottom) occupancy grids: synthetic (a, d) and real-world from Google Maps satellite database without (b, e) and with (c, f) manual annotation. Red and blue points are the ground-truth waypoints divided into the two clusters.

6. create random holes in the rows to emulate segmentation errors or missing plants;

7. compute the $N = 2n$ ground-truth waypoints as the mean points of the lines connecting the ending points of the rows with the adjacent ones.

To further increase variability, we randomly add displacement noise every time we sample a point coordinate during the generation process. We select $H = W = 800$ pixels as input dimensions for all the generated images. To investigate the effect of including synthetic curved images in the training set, we randomly generate two independent datasets, one with straight rows only, and the other with both straight and curved rows. Overall, each dataset contains 3000 images for training, 300 for validation, and 1000 for testing. In addition to the synthetic data, we manually annotate real row-based images of vineyards and orchards from Google Maps (100 straight and 50 curved). These satellite images are fundamental to test the ability of the network to generalize to real-world scenarios and to prove the effectiveness

Test	Train	AP_2	AP_3	AP_4	AP_6	AP_8
Straight Synth	Straight	0.6404 ± 0.0171	0.9284 ± 0.0088	0.9856 ± 0.0021	0.9991 ± 0.0001	0.9993 ± 0.0001
	Curved	0.5751 ± 0.0241	0.8921 ± 0.0107	0.9743 ± 0.0022	0.9979 ± 0.0001	0.9984 ± 0.0001
Straight Real	Straight	0.5191 ± 0.0288	0.8155 ± 0.0109	0.9116 ± 0.0032	0.9482 ± 0.0017	0.9507 ± 0.0024
	Curved	0.4597 ± 0.0166	0.7634 ± 0.0076	0.8788 ± 0.0089	0.9391 ± 0.0052	0.9433 ± 0.0049
Curved Synth	Straight	0.5143 ± 0.0193	0.8224 ± 0.0236	0.9232 ± 0.0166	0.9726 ± 0.0078	0.9768 ± 0.0065
	Curved	0.5664 ± 0.0226	0.876 ± 0.0066	0.9632 ± 0.0009	0.9937 ± 0.0006	0.9949 ± 0.0006
Curved Real	Straight	0.4685 ± 0.0906	0.7110 ± 0.0625	0.8125 ± 0.0625	0.8802 ± 0.0374	0.8891 ± 0.0355
	Curved	0.5327 ± 0.0269	0.8010 ± 0.0095	0.8881 ± 0.0094	0.9333 ± 0.0026	0.9374 ± 0.0033

Table 5.1: Performance of waypoint estimation on both straight and curved test datasets. We first test the model on our synthetic datasets (Straight Synth, Curved Synth) and then validate the results on manually annotated occupancy grids obtained from real satellite images (Straight Real, Curved Real). For each test set, we compare the results of the model trained on straight rows with those obtained training on curved rows. We report the mean and standard deviation for the Average Precision AP_r, where r is the maximum accepted distance in pixels between predicted and ground-truth waypoints.

of the synthetic generation process. Figure 5.7 shows examples of both synthetic and real-world manually-annotated images.

5.2.2 Model training

To select the best hyperparameters, we perform a random search over a set of reasonable values. For all the convolutional layers, we set a kernel size of 5 and channel dimension $C = 16$. For the main block of the backbone, we set the number of residual reduction modules $R = 2$. Therefore, the backbone compression factor and output cell dimension is $K = 2^{(R+1)} = 8$. We set the clustering space dimensionality to $D = 3$. Thus, the output tensors have both a dimension of $100 \times 100 \times 3$. The resulting network is a lightweight model with less than 73,000 parameters. We select Adam [122] as optimizer with a constant learning rate of $\eta = 3 \times 10^{-4}$ and batch size of 16. Experimentally, we find more effective to first train the estimation head and the backbone together with the loss of Eq. 5.2. We set the loss weight to $\lambda = 0.7$ to compensate for the high imbalance in the number of positive and negative cells and stabilize the training. We then freeze the backbone weights and train the clustering head only with the loss of Eq. 5.5. To highlight the challenge posed by curved scenarios, we independently train the model on both the straight and curved training sets. We train each model for a total of 200 epochs on an Nvidia 2080 Ti GPU using the TensorFlow 2 framework [1]. To obtain significant statistics, we run each training session three times, so that the results can be presented in terms of mean and standard deviation.

5.3 Results

In this section, we report and comment the main results regarding both waypoint detection and clustering. Visual examples are included as well, to give a qualitative idea of the performance of our model. We extensively test our approach on both straight and curved rows, including a final evaluation on real satellite data.

5.3.1 Waypoint estimation

As regards waypoint estimation, we use Average Precision (AP_r) as the principal metric, considering different values of the range threshold r, such that a waypoint is considered correctly detected if its Euclidean position error in pixels is smaller than r. In this way, we can highlight the precision of the model at different levels of proximity. The AP is commonly used for evaluating object detection tasks [64, 142] and is computed as the area-under-the-curve of the precision-recall plot obtained varying the confidence threshold t_c. The waypoint estimation results are reported in Table 5.1, where each value is detailed with its mean and standard deviation. All the tests are performed setting a waypoint suppression threshold equal to the minimum inter-row distance of the synthetic datasets, $t_{sup} = 8$ pixels.

The first important result is the model trained on curved crops being able to reach an AP_8 of about 94% on all four test scenarios. This achievement confirms the effectiveness of our model far beyond the synthetic training scenario, as real satellite data does not seem to create substantial performance drops (5.7% at worst). Looking at lower values of r, the synthetic-to-real gap rises to 11.5%, showing how the model is able to estimate synthetic waypoints with higher precision. The model trained on straight crops achieves excellent performance on its corresponding test set and even on real satellite data, but generalizes poorly on curved rows: the precision drop reaches 11% on AP_8 and even 22% considering AP_3. On the contrary, the model trained on curved crops scales very well on straight scenarios. This outcome confirms the importance of training on curved crops to obtain robust models able to cope with challenging situations.

5.3.2 Waypoint clustering

As regards waypoint clustering, we adopt two separate metrics. The first is an adjusted binary accuracy, assigning a score of 0 to the worst outcome (all the points in the same cluster, meaning 50% of the points correctly clustered) and 1 to perfect clustering. However, the number of waypoints in a crop is variable and accuracy alone does not give an insight into the distribution of errors among different samples. For example, crops with a small number of waypoints tend to be easier to cluster than dense ones. Considering the fact that full-coverage path planning is possible only if every waypoint is correctly clustered, we add a

Test	Method	Train	Adjusted Accuracy	Clustering Error
Straight Synth	K-means	Straight	**1.0000 ± 0**	**0 ± 0**
		Curved	0.9913 ± 0.0076	0.6667 ± 0.5774
	Classical	Straight	**1.0000 ± 0**	**0 ± 0**
		Curved	0.9724 ± 0.0240	2.0000 ± 1.7321
	Contrastive	Straight	0.9994 ± 0.0003	0.0187 ± 0.0114
		Curved	0.9985 ± 0.0006	0.0527 ± 0.0219
Straight Real	K-means	Straight	0.4243 ± 0.1037	26.3333 ± 7.0238
		Curved	0.4635 ± 0.0873	26.0000 ± 5.1962
	Classical	Straight	0.9532 ± 0.0429	2.3333 ± 2.0817
		Curved	0.9585 ± 0.0026	2.0000 ± 0
	Contrastive	Straight	0.9707 ± 0.0135	1.0400 ± 0.5197
		Curved	**0.9716 ± 0.0123**	**0.7700 ± 0.3012**
Curved Synth	K-means	Straight	0.9714 ± 0.0336	1.0000 ± 1.0000
		Curved	0.9885 ± 0.0199	0.3333 ± 0.5774
	Classical	Straight	0.9563 ± 0.0757	1.3333 ± 2.3094
		Curved	0.8898 ± 0.0337	3.0000 ± 1.0000
	Contrastive	Straight	0.9823 ± 0.0138	0.3414 ± 0.3278
		Curved	**0.9992 ± 0.0006**	**0.0127 ± 0.0038**
Curved Real	K-means	Straight	0.2443 ± 0.0984	73.3333 ± 29.2632
		Curved	0.2721 ± 0.1493	70.0000 ± 19.5192
	Classical	Straight	0.7247 ± 0.2734	27.0000 ± 25.5343
		Curved	0.5181 ± 0.1061	45.3333 ± 6.6583
	Contrastive	Straight	0.8571 ± 0.0924	3.4667 ± 2.4437
		Curved	**0.9344 ± 0.0116**	**1.1933 ± 0.1858**

Table 5.2: Performance of waypoint clustering on both straight and curved datasets, comparing our two approaches, the classical one based on the DBSCAN pipeline and the contrastive one, with K-means. We first test models on our synthetic datasets (Straight Synth, Curved Synth) and then validate the results on real occupancy grids obtained from satellite images (Straight Real, Curved Real). For each test, we compare the results of models trained on straight rows with those obtained training on curved rows. We report the mean adjusted accuracy and clustering error with their standard deviations.

clustering error metric computing the average number of wrongly labeled points per image. The results are detailed in Table 5.2. We compare our approaches, the classical clustering based on DBSCAN and the contrastive clustering, with the K-means algorithm directly applied in the image reference system. All the clustering tests are performed setting the confidence threshold to $t_c = 0.4$ and the waypoint suppression threshold to $t_{sup} = 8$ pixels. As for the previous results, each value is reported with its mean and standard deviation.

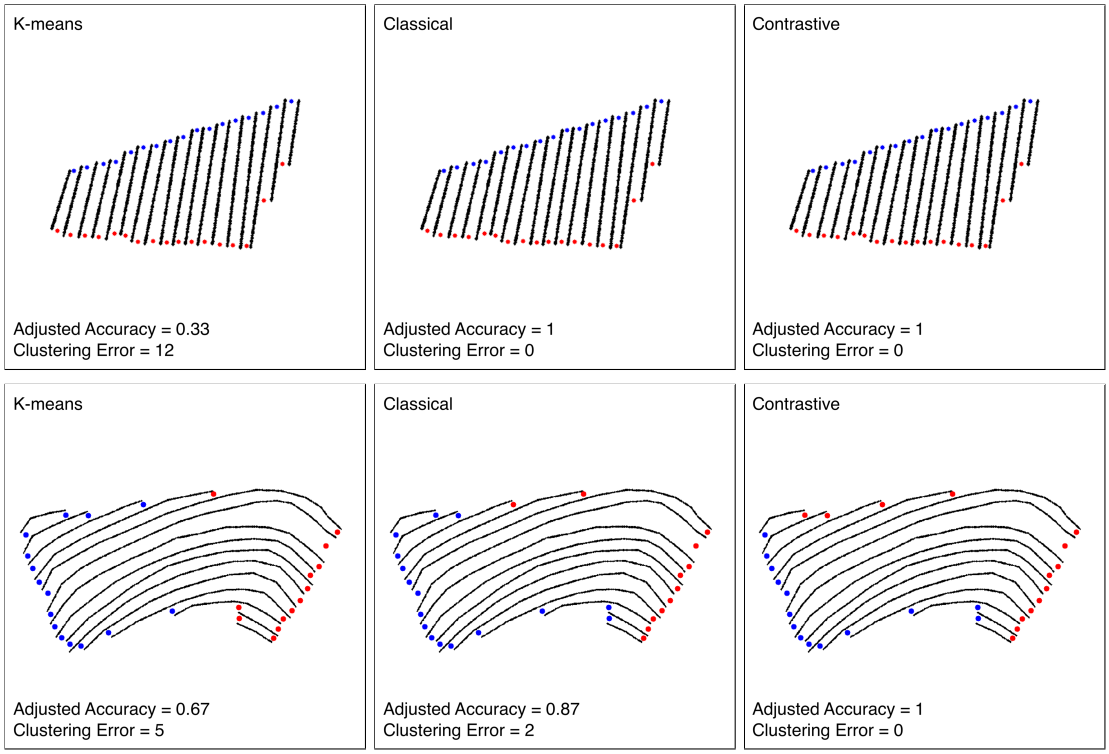


Figure 5.8: Examples of clustering on real-world straight (top) and curved (bottom) samples: K-means struggles in both conditions; the classical clustering based on the DBSCAN pipeline correctly deals with the straight field, but fails with the curved one; the contrastive clustering method correctly assigns the points in both conditions.

Straight test set The proposed methodologies achieve remarkable results, outperforming the baseline solutions in all the testing scenarios. In particular, for the straight test set, the classical clustering is able to generalize to straight satellite crops, since the methodology is specifically designed to cope with real-world straight rows. Also, contrastive clustering, with both training strategies (based on straight and curved crops), approaches perfect clustering on the synthetic straight dataset and generalizes well to real crops. On the contrary, K-means, which perfectly works for the well-separated synthetic samples, loses more than half of its adjusted accuracy and presents a very high clustering error when switching to real test rows, mainly due to the irregular shapes typical of real-world vineyards.

Curved test set As regards curved test sets, K-means clustering is totally unable to generalize to the real dataset. At the same time, also the classical DBSCAN pipeline results drop significantly when switching to real samples, due to its heavy dependence on angle estimation. The contrastive clustering, trained on straight

rows, obtains 0.98 adjusted accuracy and 0.34 clustering error on synthetic data, outperforming both baselines. However, it struggles to generalize to real crops, reaching an adjusted accuracy of 0.86. On the other hand, the model trained on curved data outperforms the baseline and the classical clustering in synthetic and real data, where it achieves an adjusted accuracy of 0.93. This result can be considered extremely positive, taking into account the strong challenges present in satellite data. In particular, a clustering error of 1.19 is remarkably smaller than those obtained by K-means and DBSCAN. In conclusion, these results confirm how the proposed methodology, combined with a well-devised generation process of curved synthetic samples, allows waypoint generation and clustering even in challenging scenarios.

5.3.3 Qualitative results

To give further insight into the performance of the proposed methodology, we present some qualitative examples of real-world curved samples. Fig. 5.8 shows a comparison between the three clustering methodologies. While both proposed approaches can deal with the straight scenario, only the contrastive clustering correctly assigns the points in the curved one. Finally, Fig. 5.9 shows some examples of full-coverage path planning. The trajectory can be obtained by connecting the waypoints with any path planning algorithm, such as the popular A* [86], RRT* [131] and D* [234, 123]. The proposed examples are all obtained with the Adaptive Row Crops Path Generator (ARC-PG) planner [32], which is specifically proposed to solve the row crop planning problem. The procedure exploits the gradient descent principle in case the partial path to be computed should lay inside the row crop, while it uses the vanilla A* algorithm to switch between different rows. By modifying the costmap for the search algorithm, ARC-PG is able to produce a centered and smooth path with on average less processing time compared with the other planning algorithms. With geo-referenced maps, the planned path can be then converted from the image reference system to a GNSS reference frame to be used in real-world navigation. All the examples are produced with the model trained on the curved dataset and setting the confidence threshold to $t_c = 0.4$ and the waypoint suppression threshold to $t_{sup} = 8$ pixels.

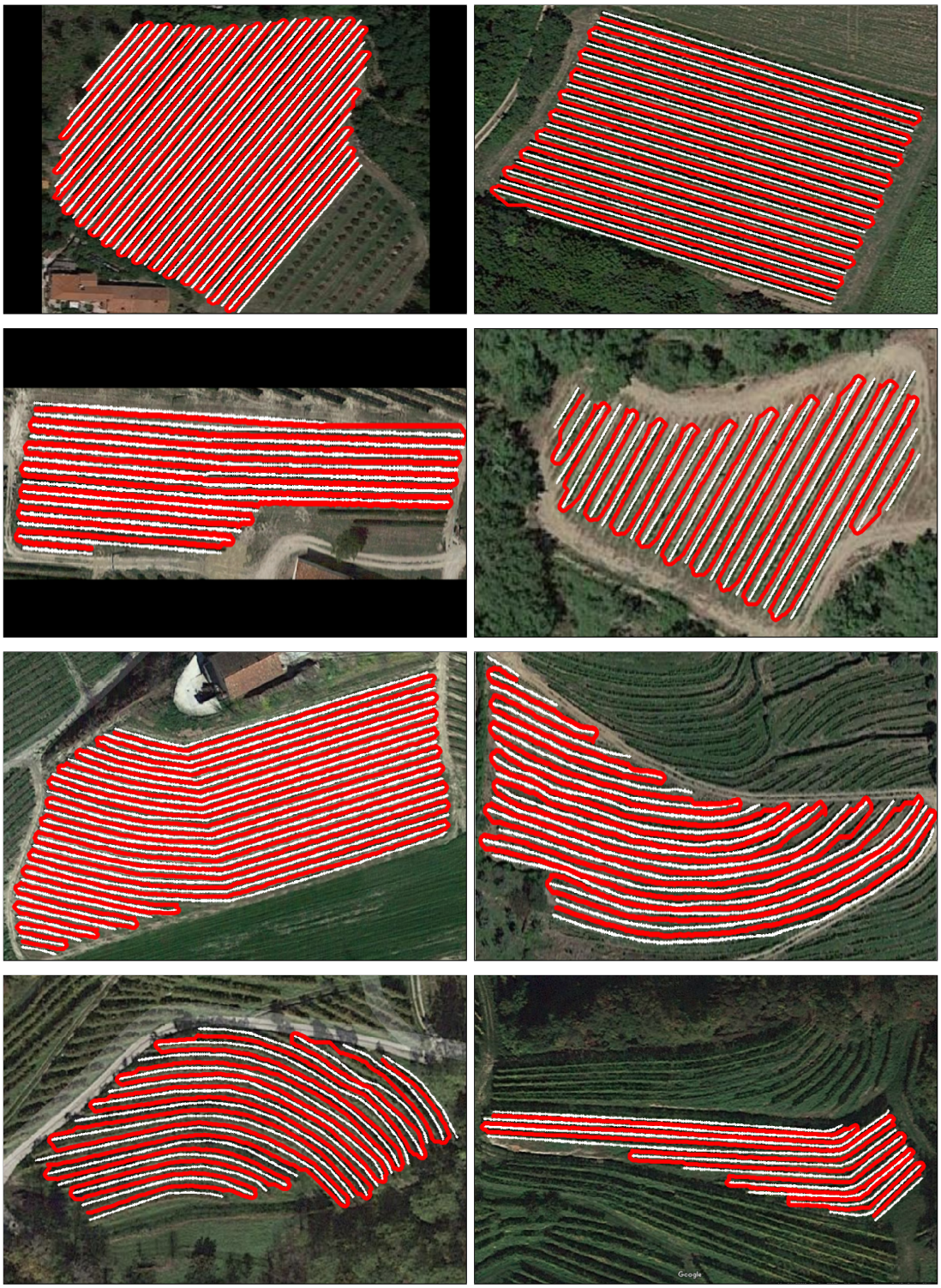


Figure 5.9: Examples of full-coverage path planning in straight and curved real-world vineyards taken from Google Maps satellite database.

Part III

Perception

Chapter 6

A Real-Time Apple Detection System at the Edge

Monitoring agricultural farms and orchards mainly rely on skilled farmers and workers who are responsible for assessing several growth stages before performing farming-related actions in order to maximize the quality and yield. The manual work of these farmers consumes time and increases production costs, and workers with less knowledge and experience make unnecessary mistakes. With the advancements in precision agriculture and information technology, crop imaging has become an important source of information that can be used to assess the vegetation status of the crops, fruit growth, yield, and quality. Two important features that enable the farmers to estimate crop load and yield mapping in tree fruit crops are fruit counting and size estimation. Several studies have proposed fruit detection in orchards using machine vision systems for automatic growth assessment, robotic harvesting, and yield estimation [44, 110]. It has been the primary problem to develop algorithms that enable the apple harvesting robot to directly, quickly, and accurately recognize fruits in real-time [8]. In the natural environment, for the visual systems, apple fruit detection is typically more difficult because of the influence of lights and shadows, branches, and leaf coverings. Apples visual appearance in the natural environment may be categorized as occluded and non-occluded. Occlusion of fruits due to leaves, branches, and other fruits and variable lighting conditions are some of the main reasons that make it more challenging to achieve good accuracy and robustness in fruit detection [44].

With the upsurge of machine learning, deep learning algorithms have been extensively used in agriculture-related applications [115]. YOLO [197, 198, 199] is a CNN-based method that deals with object detection as a regression problem. However, it is not suitable for real-time applications such as harvesting robots due

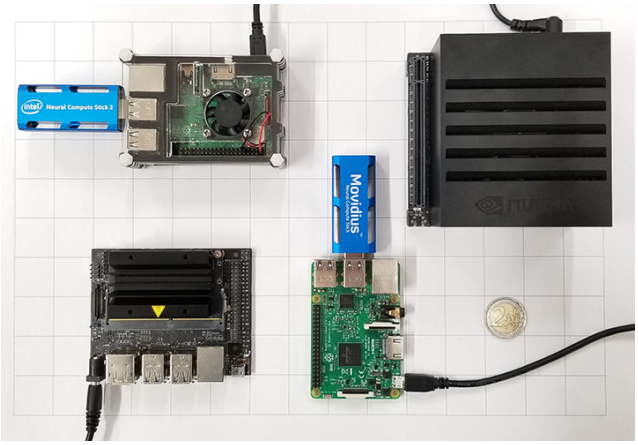


Figure 6.1: The analyzed embedded platforms. Top left: Raspberry Pi 3 B+ with Intel Movidius NCS2. Top right: Nvidia Jetson AGX Xavier. Bottom left: Nvidia Jetson Nano. Bottom right: Raspberry Pi 3 B+ with Intel Movidius NCS.

to its complex architecture that requires more processing power. Optimization of the parameters of the model reduces the computational complexities and thus is needed for deployment on edge platforms.

In this work, we apply transfer learning [152] to re-train and fine-tune a custom version of the YOLOv3-tiny model, specifically optimized for the accurate detection of small objects. After training, the resulting network has been benchmarked on all the images of the *apple* class from the OIDv4 [127] dataset, producing a reproducible metric that can be easily compared with future works. Finally, we deploy the model on embedded platforms such as Raspberry Pi 3 B+ in combination with Intel Movidius Neural Computing Stick (NCS), Nvidia Jetson Nano, and Jetson AGX Xavier, assessing their performance in terms of speed and power consumption. Fig. 6.1 shows the analyzed platforms, comparing their physical scale.

6.1 Methodology

YOLO is a network specifically designed for fast and accurate real-time object detection. It has comparable performance in terms of accuracy with other popular object detection algorithms like Faster R-CNN [200], SSD [146] and RetinaNet [141], but it is much faster and compact, thus being an optimal choice for real-time embedded applications. It is a single fully convolutional neural network that takes as input a raw image and gives as output bounding boxes and related classes of recognized objects inside the presented scene. Different versions of the model have been released [197, 198, 199] that gradually increased the accuracy of the general framework without giving away too much of its inference speed. At the same time, all different versions have been released with a lighter counterpart defined

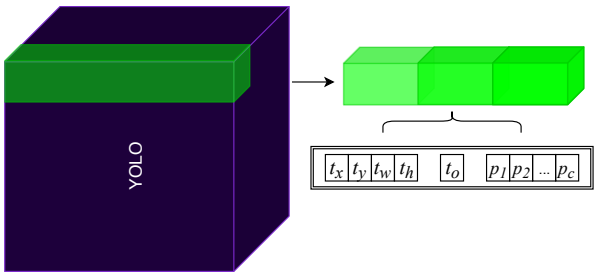


Figure 6.2: In a YOLO layer, a 1x1 convolution predicts, for each spatial location, B arrays with $(5+C)$ elements, that are used to compute the bounding box center coordinates (t_x, t_y), width t_w, height t_h and confidence t_o, plus C classes probabilities $(p_1, \ldots, p_C)$.

as *tiny* that has a simplified and optimized structure without losing too much accuracy. The intrinsic characteristics of the *tiny* version make it suitable for Edge AI applications. This work takes the YOLOv3-tiny architecture as a starting point for designing an embedded apple detector system.

6.1.1 Network architecture

YOLOv3-tiny is a fully convolutional network that can accept inputs of different sizes during and after training. It can be divided into two main blocks: the first is the feature extractor or backbone, referred to as *darknet-19*. Its principal and fundamental role is to extract features in a hierarchical fashion, starting from raw pixels coming from the input layer. The extracted representations are later used as starting point by the other modules of the network. Darknet-19 is a light and efficient feature extractor, but can be easily swapped with any other backbone, such as, for example, ResNets [87]. It features a standard architecture greatly inspired by VGG [229], making use of only 3x3 filters throughout the entire structure, max-pooling layers to reduce the spatial dimensionality and obtain local invariance. Finally, darknet-19 exploits Batch Norm layers [104] to regularize the network, reducing the internal covariance shift. All backbone convolutional layers use Leaky ReLU [155, 269] as activation function.

On the other hand, the second block of the YOLOv3-tiny architecture analyzes the extracted representations, and it predicts the bounding box position and class of the different objects present in the input raw image. That is achieved with 1x1 convolutions arranged as a Pyramid Network [140] structure. Thus, the output of the network is a feature map tensor that has the same spatial dimensions as the previous layer.

The network predicts center coordinates (t_x, t_y), width t_w and height t_h, in order to obtain the bound box dimensions. Moreover, each bounding box is linked to a confidence score t_o. This score reflects how confident the network is that the

box contains an object and has the right dimensions. Moreover, the model also outputs C conditional class probabilities with independent sigmoid activations. The classical softmax activation has been replaced because it assumes mutual exclusion between different classes. That is not true with larger datasets like Open Image Dataset (OID) [127] or ImageNet [50], where a multi-label classification approach is adopted.

We denote the last layer as a YOLO layer, whose 1x1 convolution produces for each spatial location an array of $B \cdot (5 + C)$ elements, where C is the number of classes and B is the number of concurrently predicted bounding boxes, that for YOLOv3-tiny is set to 3. Therefore the overall output tensor dimension is $N \times N \times (3 \cdot (5 + C))$, where N is the output grid spatial dimension. A graphical representation of a YOLO layer is depicted in Fig. 6.2.

It is essential to point out that, YOLO, starting from its second version, does not predict coordinates directly, but it refines hand-picked priors known as *anchors*. This approach simplifies the problem and makes it easier for the network to learn. Moreover, an ablation study made by the author [198] demonstrated how predicting offsets increases the recall of the network and stabilizes gradients during training. So, for each output spatial location, there are B bounding boxes with dimensions decided a priori. The network, during the learning phase, learns how to adjust these bounding boxes when an object is detected. Anchors dimension is critical and, in order not to manually guess their prior dimension, authors suggest using K-means clustering over the data set ground truth bounding boxes in order to predict the width and height of the box as offsets from cluster centroids. That allows generating prior bounding boxes that initialize the model with better representations and make the task easier to learn.

As shown in Fig. 6.3, the center of a predicted object falls in a certain cell of the output grid. That cell is responsible for detecting the presence of the object. During training, we force the network to use the anchor box that has the prior dimension with the higher intersection-over-union (IoU) with the ground truth bounding box. So, the network will generate t_x, t_y, t_w and t_h in order to refine one of the three anchors present in that cell to perform the detection of the object. The actual bounding box center coordinates (b_x, b_y) and dimensions (b_w, b_h) can be computed with the following transformations:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{6.1}$$

where (c_x, c_y) are the top-left coordinates of the cell responsible for the prediction, (p_w, p_h) the anchor prior dimensions and $\sigma(\cdot)$ the sigmoid function. Moreover, a non-maximum suppression (NMS) algorithm is used to remove redundant

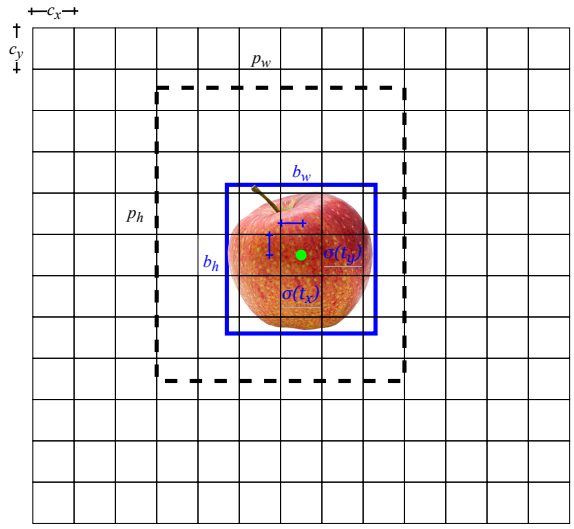


Figure 6.3: YOLOv3-tiny predicts the dimension of the bounding boxes as offsets to predefined hand-picked prior boxes known as *anchors*, preventing unstable gradients during training and making easier for the network to learn the task.

detections that have very high IoU and produce the final output bounding box predictions.

6.1.2 A custom YOLOv3-tiny for small objects detection

The two first versions of YOLO greatly struggle to detect small objects in the input image. That is mainly because that YOLO imposes a strong spatial constraint on bounding box predictions since each grid cell can only predict three bounding boxes. Moreover, the input dimension reduction performed in the backbone induces the loss of low-level features, which are instrumental for detecting small objects. For this reason, the authors proposed with the third version of the tiny network [199] a pyramid structure producing detection at two different scales. The backbone, with the pooling layers and convolutional strides, spatially downsamples the input image until the first YOLO layer, where a prediction is made with a $\frac{1}{32}$ spatial dimensionality reduction with respect to the input. Then, layers are upsampled by a factor of 2 and concatenated with representation maps of a previous backbone layer reaching a $\frac{1}{16}$ reduction factor.

We take this concept even further, building a YOLOv3-tiny architecture with predictions across three different scales. So we add another upsampling by a factor of 2, reaching a $\frac{1}{8}$ reduction factor. Since we adopt an input dimensionality of (608×608), our output dimensionality is (19×19), (38×38), and (76×76).

Fig. 6.4 presents the overall architecture of the proposed network. The pyramid structure is clearly represented: extracted features from the backbone are sampled

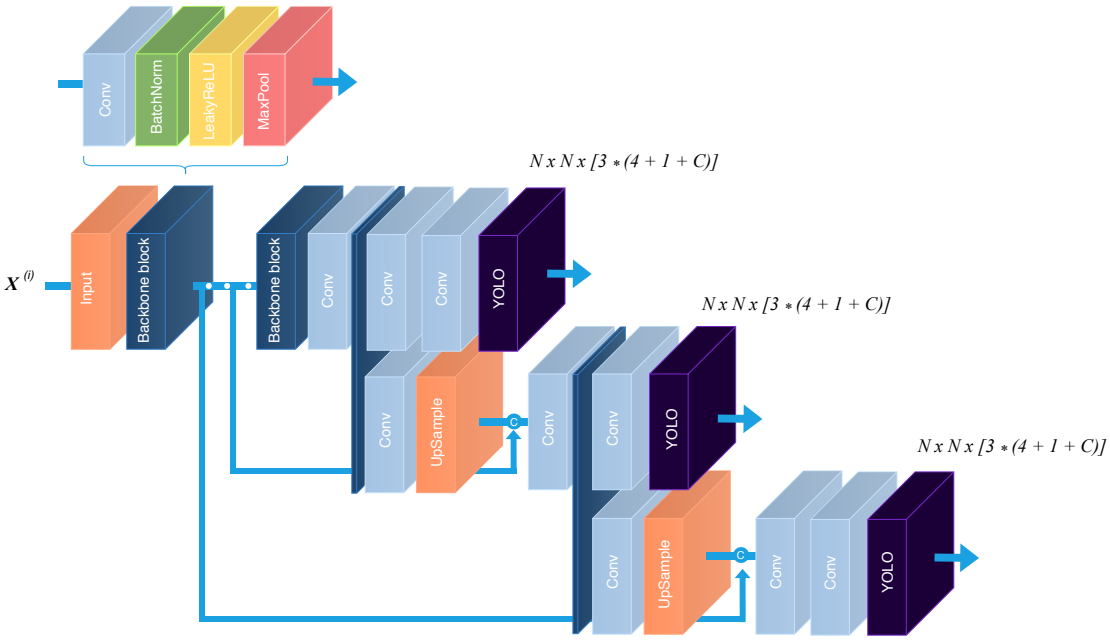


Figure 6.4: Overview of the modified version of the model YOLOv3-tiny. The original architecture features only the two first branches and makes detection at two different scales, instead of three. The detection layers exploit feature maps of three different sizes, having reduction factors of $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$. Each YOLO layer predicts for each cell $B = 3$ bounding boxes using 3 different anchors. The blue vertical planes mean that the specific layer output is fed to both the subsequent graph branches.

at two different points and are lately concatenated with upsampled higher-level features. In this way, high-level features are enriched with low-level information that helps the network learn fine-grained features that are fundamental for detecting small objects. At each scale, YOLO layers predict 3 bounding boxes per cell using 3 different anchors. So, the total number of anchors used is 9, and the total number of possible bounding boxes predicted by the modified network is equal to $3 \cdot 19 \cdot 19 + 3 \cdot 38 \cdot 38 + 3 \cdot 76 \cdot 76 = 22743$. Confidence scores and NMS drastically reduce this number, removing low confidence and redundant detections.

Fig. 6.5 shows a graphical representation of the grids used for the predictions at the three different scales. Each grid represents the tensor used for the prediction. It is noticeable how a finer grid can greatly improve the prediction capabilities of small objects. Indeed, for our specific case, this simple modification largely improves the accuracy of the model with a small overhead in terms of speed and power consumption.

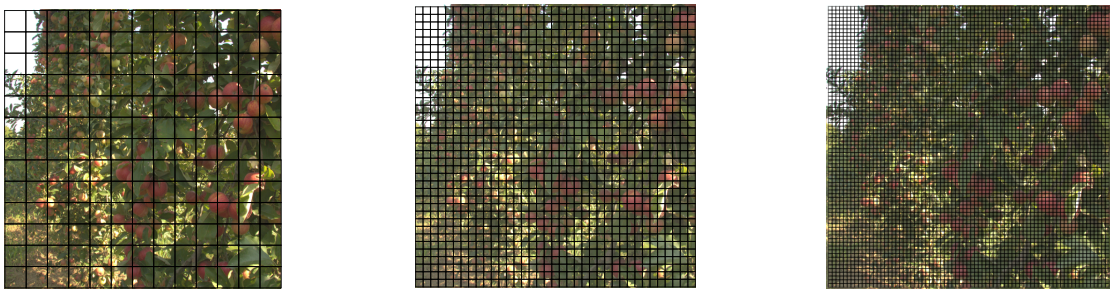


Figure 6.5: Our custom version of YOLOv3-tiny predicts at three different scales. The YOLO layers make detections with grids of three different sizes, having a reduction factor with respect to the input image of $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$, respectively. Having an input size of (608×608), detections are made with dimensions (19×19), (38×38), and (76 × 76).

6.2 Experiments

In this section, firstly we present the dataset and some technical details of the training process, then we discuss experimental evaluations for both the model and its deployment on the selected embedded platforms. Finally, quantitative and qualitative results are reported with a comparison between the different devices.

6.2.1 Dataset description

An orchard is considered in order to acquire a custom data set for the training process. Two popular types of apple (Braeburn and Fuji) are considered in this study, which are the most common types found in the northern part of Italy. The image acquisition campaign is conducted in randomly selected healthy apple trees in orchards, using a reflex digital camera with 18 megapixels at different times. Image acquisition is performed for separate/non-overlapped fruits, overlapping fruits/occluded fruits under variable lighting conditions such as fully exposed to sun from the front, full sun influencing from the back of the fruits, and fruits covered by the shades of leaves/branches or other apples. It must be underlined that, even though for the specific application a quite low image resolution is needed (608 × 608 pixels), a reflex camera is adopted in order to guarantee the best image quality for other possible future uses of the same dataset.

We create a set of 618 sample images. Subsequently, inspired by active learning techniques [150], in order to speed up label generation, we use an extensive and accurate version of YOLOv3 known as YOLOv3-spp, pretrained on the COCO dataset [142], to create a draft of the ground truth labels. In order to increase the

recall of the model, we set the network with a low value of confidence. Then, we perform a manual inspection of the generated ground truth, using LabelMe [211], refining and adjusting the bounding boxes. Finally, we apply a straightforward pipeline to preprocess our training dataset: each image is first resized from 5202×3465 pixels to the input size 608×608, then it is normalized by subtracting its mean.

6.2.2 Experimental setting

Using transfer learning, we start from a pretrained backbone of the original model. In particular, we adopt the *darknet-19* model trained on the COCO dataset. That greatly speeds up the training process, drastically reducing the number of samples required to achieve a high level of accuracy.

It is worth noticing the dimension of the prediction tensors at the 3 different scales; having only one class ($C = 1$) and three anchors per location ($B = 3$), each output cell produces vectors with $B \cdot (5 + C) = 18$ elements. This low output dimensionality further decreases the inference time required by the network.

We train the network for 100 epochs using AdamW optimizer [153] and setting $\beta_1 = 0.89$ and $\beta_2 = 0.99$ and $\epsilon = 10^{-9}$. AdamW is a slight modification to the original Adam updating rule:

$$\Delta \boldsymbol{p} = -\eta \left(\frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{E}}_t + \epsilon}} + \gamma \boldsymbol{p} \right) \tag{6.2}$$

where $\Delta \boldsymbol{p}$ is the parameters update, $\hat{\boldsymbol{E}}_t$ and $\hat{\boldsymbol{m}}_t$ the corrected running averages of squared and plain gradients, respectively, ϵ an arbitrary little value to avoid zero-division error and η the learning rate, like in the original Adam formulation (Eq. 2.30). AdamW adds a little portion of the parameters $\gamma \boldsymbol{p}$, regularizing large weights inside the network and giving advantage to rare features, where γ is a regularization parameter. In all our experimental evaluations, it has given better results than classical L_2 regularization. We set $\gamma = 10^{-4}$. All the training processes have been carried out on a workstation with an Nvidia RTX 2080Ti and 64GB of DDR4 SDRAM. Each training takes an average of one hour using the TensorFlow framework [1] and CUDA 10.

6.2.3 Quantitative results: detection performance

To understand the model performance, mean average precision (mAP) is computed on the test dataset. Mean average precision is a popular object detection scoring method that assesses the network performance in detecting the target objects for different values of target intersection-over-union $\text{IoU}_{\text{target}}$. This methodology has been presented for the PASCAL Visual Object Classification (VOC) challenge

IoU_{target}	Proposed architecture			Original Yolov3-Tiny	Gain
	Recall ($c = 0.25$)	Precision ($c = 0.25$)	mAP	mAP	
0.50	0.83	0.69	83.64%	77.02 %	6.62 %
0.75	0.55	0.46	47.97%	42.50 %	5.47 %

Table 6.1: Detection performance of the network on the test dataset from OIDv4 [127] for 0.5 and 0.75 IoU thresholds. The same computation is made with the original YOLOv3-tiny architecture retrained on the same training dataset. The results show how the proposed architecture can boost recognition in agricultural contexts by allowing small fruit detection.

2012 [64]. Each predicted bounding box is compared with the ground truth and marked as correct (true positive TP) if the apple is present and $IoUpred > IoU_{target}$. If IoUpred is lower than the target or the apple is not present, the prediction is marked as incorrect (false positive FP). Finally, all the apples not detected are marked as missing predictions (false negative FN). Since the predicted bounding boxes are given as output only if they have a level of confidence above a certain threshold c, it is possible to compute the precision (p) and the recall (r) of the network over the test dataset as a function of c:

$$p(c) = \frac{\mathrm{TP}(c)}{\mathrm{TP}(c) + \mathrm{FP}(c)} \qquad r(c) = \frac{\mathrm{TP}(c)}{\mathrm{TP}(c) + \mathrm{FN}(c)} \tag{6.3}$$

Computing $p(c)$ and $r(c)$ for all the possible confidence thresholds $0 \leq c \leq 1$, it is possible to get the precision/recall curve. The graph is then usually smoothed in order to get a monotonically decreasing precision curve by setting $p(r) = \max_{r' \geq r} p(r')$. The average precision of the network is computed as the area-under-the-curve of the precision/recall and is always a number between 0 and 1:

$$\mathrm{AP} = \int_0^1 p(r)dr \tag{6.4}$$

An average precision equal to 1 means that the detector is able to reach a perfect precision (100%) for all the values of recall. Thus it is possible to find a value of c such that we are able to detect all the objects with correct bounding boxes. On the other hand, an average precision of 0 means that we cannot detect any object correctly whatever value of c we choose, thus both $p(c)$ and $r(c)$ are always equal to 0.

For a multi-class object detection algorithm, the mean average precision is the mean of the AP over all the classes. In our specific context, we have a single class

Device	Mode	$V_{al}[V]$	$I_{mean}[A]$	$P[W]$	fps
Jetson AGX Xavier	IDLE 30W	19	0.35	6.65	-
	RUNNING 30W	**19**	**1**	**19**	**30**
	RUNNING 15W	19	0.88	16.72	25
	RUNNING 10W	19	0.66	12.54	13
Jetson Nano	IDLE 10W	5	0.32	1.6	-
	RUNNING 10W	5	2.04	10.20	8
	RUNNING 5W	5	1.42	7.1	6
Raspberry Pi 3B+	IDLE	5	0.61	3.075	-
RP3 + Movidius NCS 1	RUNNING	5	1.2	6	4
RP3 + Movidius NCS 2	RUNNING	5	1.12	5.6	5

Table 6.2: Comparison between different devices power consumption and performances achieved with our customized version of YOLOv3-tiny. Jetson series boards can be run at different power modes reducing current absorption at the expense of lowering computational capabilities. The **Mode** column shows the theoretical maximum absorbed power in the different working modalities, which is different from the actual power consumption during the execution of the algorithm. The best performance, in terms of fps, is in **bold**.

($C = 1$), thus AP = mAP. The mean average precision gives a piece of information on the quality of the network detection independent from the chosen confidence c, which can be chosen depending on which metric, precision or recall, is more relevant for the specific application. mAP can be computed at different values of $\text{IoU}_{\text{target}}$. Usual values are 0.5 and 0.75 in order to evaluate the model performance with different requirements on object localization accuracy.

Tab. 6.1 presents the recall and precision for our default confidence threshold $c = 0.25$ and the mAP for the two selected values of $\text{IoU}_{\text{target}}$. The evaluation is performed with both the modified and the original YOLOv3-tiny architecture, both fine-tuned on the training dataset. Results show how the change in the architecture can boost the mAP on the test dataset by up to 6.6%, thanks to the improved detection of small objects.

6.2.4 Quantitative results: embedded implementation

After training, we deploy the model on the different hardware platforms:

- Raspberry Pi 3 B+ in combination with Intel Movidius Neural Computing Stick (NCS)
- Raspberry Pi 3 B+ in combination with Intel Movidius Neural Computing Stick 2 (NCS 2)

Device	Price	fps	price/fps
Movidius NCS†	70 $	4	17.5
Movidius NCS 2†	74 $	5	14.8
Jetson Nano	**99 $**	**8**	**12.4**
Jetson AGX Xavier	699 $	30	23.3

Table 6.3: Price/performance analysis of the different embedded platforms. The best price/fps ratio is in **bold**. Prices are at the time of writing.
†: this device requires an additional embedded computer

- Nvidia Jetson Nano
- Nvidia Jetson AGX Xavier

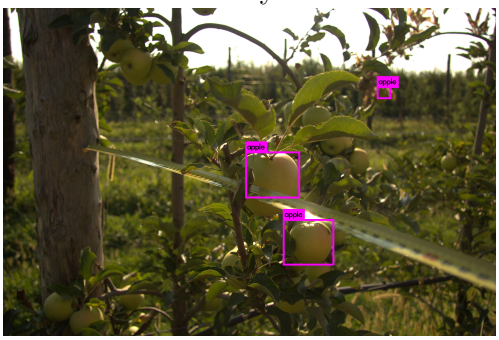
For details on the HW specifications of the selected devices, refer to Tab. 2.1. The selected platforms are depicted in Fig. 6.1. We test the performance in terms of absorbed power and frame rate.

Firstly we measure the power consumption of the different boards (Jetson AGX Xavier, Jetson Nano, Raspberry Pi 3B+) in idle condition, and then we run the algorithm for nearly 5 minutes to be sure to be at steady state. We measure directly the current absorbed from the power source, thus obtaining the power consumption of the entire system. Since the Jetson boards allow the user to select different working conditions, we test all of them. The results are presented in Tab. 6.2.

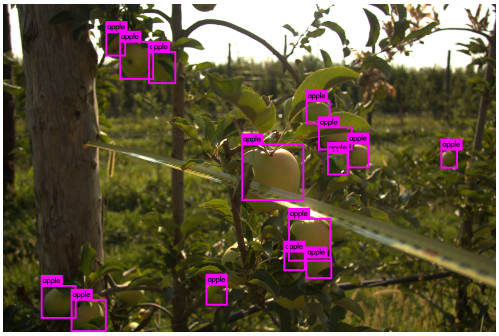
The Nvidia Jetson AGX Xavier is the most performing platform, being able to reach 30 fps in the 30W operational mode. Also, in the other modalities, it is able to reach frame rates suitable for strong real-time applications. With the Jetson Nano, the frame rate drops to 8 fps in 10W mode, which can still be an acceptable value for soft real-time contexts. With the Raspberry Pi and the Intel NCS accelerators, the performance is further lowered: the more advanced NCS2 is able to outperform the basic NCS both in terms of frame rate and power consumption. However, despite being more flexible, these USB accelerators cannot go beyond 5 fps in the best case.

An interesting comparison between the different platforms is the price/fps ratio, shown in Tab. 6.3. The Jetson Nano appears to be the best choice if we are looking for a tradeoff between performance and cost. On the other hand, the AGX Xavier has the higher ratio, since it is a board with the highest quality, but certainly not suitable for low-cost solutions. The Intel NCS accelerators come in second and third place for price/fps ratio, but it must be underlined that, since they are USB accelerators only, an additional embedded computer must be purchased, increasing the final cost.

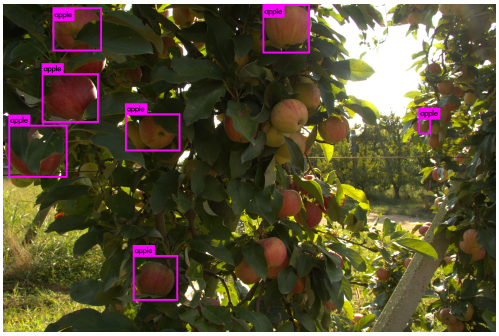
YOLOv3-tiny retrained | Our architecture



(a) $p = 0$ $r = 0$ mAP = 9.9%



(b) $p = 0.87$ $r = 0.53$ mAP = 59.33%



(c) $p = 1$ $r = 0.08$ mAP = 32.73%

(d) $p = 0.75$ $r = 0.42$ mAP = 60.86%

Figure 6.6: Comparison between the original YOLOv3-tiny and our modified version that performs predictions at three different scales. Fig. (b). and (d). produced by our custom version demonstrate a considerable improvement over the predictions of the original model. The additional detection at a scale with stride 8 largely improves the recall of the model and make it much more robust to scale variations. All the predictions are done with default confidence threshold $c = 0.25$. Precision and recall are computed for $IOU_{\text{target}} = 0.5$.

6.2.5 Qualitative results

A qualitative comparison with the original architecture is presented in Fig. 6.6. We process two images excluded from the training dataset with default confidence threshold $c = 0.25$, and we compute precision and recall with $\text{IoU}_{\text{target}} = 0.5$. It is interesting to notice that precision and recall for image (a) are both 0 since no bounding box has sufficient intersection-over-union to be considered as a TP. Image (c) has a precision equal to 1, but very poor recall, since the network is able to detect only 8% of the apples. The proposed modified architecture, on the other hand, is able to strongly increase the recall by detecting very small fruits, boosting the quality of the predictions. In this scenario, the mAP gain is a lot higher with respect to the test dataset taken from OIDv4. This is due to the

Figure 6.7: Qualitative results of some additional test images acquired from the same study site of the training dataset. It is possible to notice how our custom version of YOLOv3-tiny is robust to different factors of variation in how apples appear. Simultaneously handling variations in illumination, viewpoint, scale, occlusion, and background clutter is a challenging task that our system has to tackle in real-time with limited computational capabilities.

fact that the images of the apple class in the OIDv4 dataset present, on average, bigger apples with respect to the training dataset taken on a real orchard, so the difference between the two architectures is less visible. On the other hand, on the test images taken from the same dataset used for training, the ability to detect little apples becomes fundamental for reaching a high recall value. However, in the results section, we present our results for OIDv4 in order to make the experiments repeatable and allow future direct comparisons with our work.

Finally, Fig. 6.7 presents some other test images acquired from the same study site of the training dataset, to give a qualitative idea of the model detection performance. It is possible to see how our network is able to recognize a great number of apples in several image conditions such as different illumination and contrast. The network is able to detect fruits on different scales, and, in particular, it can recognize very small apples, even in bad lighting conditions.

Chapter 7

Robust Ultra-wideband Range Error Mitigation with Deep Learning at the Edge

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 [252, 276]. As Global Navigation Satellite System (GNSS) is the benchmark solution for outdoor positioning, Ultra-wideband (UWB) real-time locating systems have recently become the state-of-the-art technology for localization in indoor environments [241], offering decimeter-level accuracy and increasingly smaller and cheaper transceivers [158]. 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 when compared to other radio-frequency technologies [219].

Nevertheless, in a real-world scenario, the complexity of the environment often leads to partial or total obstruction of the signal between the transmitter and the receiver, thus causing a substantial degradation of the positioning performances. UWB is still primarily affected by this effect, defined as a non-line-of-sight (NLoS) condition, in which the range estimates based on time-of-arrival (ToA) is typically positively biased [217, 187], as shown in Fig. 7.1. That is particularly true for indoor localization, where ranging errors introduced by multipath and NLoS conditions can quickly achieve large deviations from the actual position [40]. Robust and effective mitigation is therefore necessary to prevent large localization errors.

Several approaches have been proposed to address the NLoS problem. In the presence of a large number of anchor nodes available, NLoS identification is the preferable choice so far. Indeed, once an NLoS anchor is identified, it can be easily eliminated from the pool of nodes used for the trilateration algorithm [226]. The majority of the proposed methodologies found in the literature make use of channel

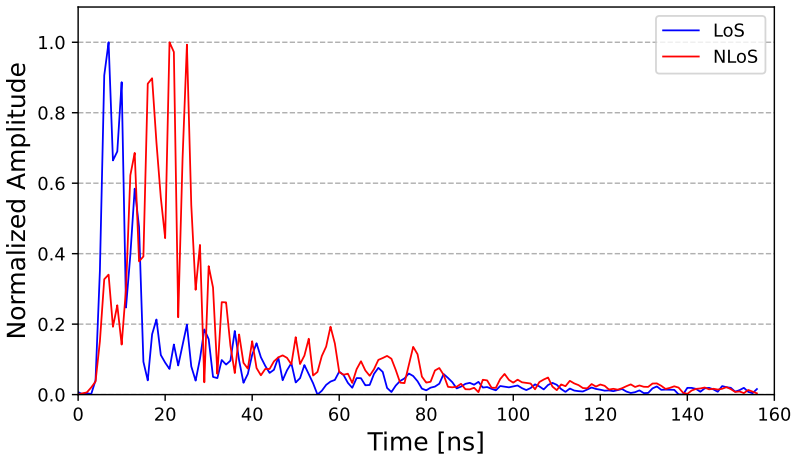


Figure 7.1: Examples of a LoS and a NLoS (with an aluminum obstacle) channel impulse response (CIR) with normalized amplitude in an indoor environment. In the NLoS case, the signal travels along many routes until it reaches the receiver. That makes the ToA estimation ambiguous.

and waveform statistics [162, 220, 19], likelihood ratio tests or binary hypothesis tests [226, 179] and ML techniques. In the latter case either hand-designed techniques, such as support vector machine (SVM) [271], Gaussian processes (GP) [268], or representation learning models have been investigated [111, 233].

Despite the simplicity of applying NLoS identification [81], in almost all practical situations, there is no sufficient number of anchors available to exclude some of them. So, the majority of research community efforts focus on range mitigation and direct localization mitigation. Regarding the latter, even if there are studies that show excellent position estimation in multipath environments [182, 98, 194], the collected training data are incredibly site-specific. Therefore, conducting the data collection on one site does not allow exploiting the resulting model 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 [217]. Range error mitigation is mostly performed with similar techniques as NLoS identification [268, 279, 161, 265] and also with more extreme tasks such as error mitigation for through-the-wall (TTW) conditions [227]. Moreover, Bregar & Mohorčič [27] attempted to perform range error estimation directly from the channel impulse response (CIR) using a DL model. Nevertheless, being a preliminary study, no relevance has been given to studying the network, optimizing it, and making it able to generalize to different environments.

This work focuses specifically on investigating a novel efficient DL model that performs an effective range error mitigation, using only the raw CIR signal as input

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. We adopt the latest advancements in DL architectural techniques and graph optimization to improve nearly 45% and 34% the NLoS and LoS conditions, respectively, in an unknown indoor environment up to barely 0.4 mJ of energy absorbed by the network during inference. Moreover, our proposed methodology does not require additional NLoS identification models. Still, it is able to extract valuable features to estimate the correct range error directly from the CIR in both LoS and NLoS conditions. The main contributions of this work can be summarized as:

1. design and train a highly efficient deep ANN for UWB range mitigation in NLoS and LoS conditions using only raw CIR data points as input;

2. apply weight quantization and graph optimization techniques for power and latency reduction in range error mitigation;

3. evaluate and compare several devices and hardware accelerators, measuring power and computational request for different optimized networks;

4. collect and analyze a novel open-source dataset with different NLoS scenarios and settings, highlighting the generalization limitations of a hierarchical learning model.

7.1 The DeepUWB dataset

The measurements are taken in five different environments to cover a wide variety of LoS and NLoS scenarios: an outdoor space, in which the only source of error is the presence of obstacles, and three office-like rooms, to include the effect of multipath components. In particular, the biggest room is approximately 10m x 5m large, the medium one is 5m x 5m, and the smallest is 5m x 3.5m. Moreover, to analyze the TTW effect, some measurements are acquired across different rooms. Taking range measurements in different conditions allows performing training, validation, and testing on entirely different datasets, avoiding overfitting and encouraging a representation learning model to learn domain-independent features.

The Decawave EVB1000 boards mounting the DWM1000 UWB transceiver are configured to guarantee precise ranging and high update frequency according to the constructor manual, and antenna delays are tuned to compensate for measurement bias. The measurements are taken using a Leica AT403 laser tracker as ground truth. First, we measure the anchor position to create a precise reference system; then, the laser follows the reflector placed on the moving tag estimating its position ten times per second. Meanwhile, tag and anchor perform two-way ranging at approximately the same frequency. The tag follows a path in an environment filled

	SR	MR	BR	OD	TTW	**Tot**
Size	5m x 3.5m	5m x 5m	10m x 5m	-	-	-
Total samples	17601	13210	18422	4971	954	55158
NLoS samples	15739	12151	16797	4826	954	50467
LoS samples	1862	1059	1625	145	0	4691

Table 7.1: Number of data points collected in the small room (SR), medium room (MR), big room (BR), outdoor (OD), and trough-the-wall (TTW) during the dataset construction.

with obstacles to generate NLoS measurements. When no obstacles are present, the samples are explicitly labeled as LoS. After a satisfying number of data points are obtained, the configuration is changed by modifying the anchor position or the type and position of the obstacles. The total samples-per-environment distribution is presented in Tab. 7.1.

Each data point consists of the CIR vector and the corresponding error between the board range estimation and the ground truth, matched by comparing timestamps. For each CIR vector, only 152 samples after the first detected peak are retained, as suggested in [27]. Moreover, five additional samples before the peak are included to compensate for eventual errors in the detection, for a total of 157 temporal samples. Finally, the environment and obstacles used for the measurements are reported to study their effect on the proposed method. The collected dataset is open source and publicly available[1] for future work comparison.

7.1.1 Dataset analysis

To visualize the distribution of the acquired instances in the data space, we 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. 7.2, the first analysis highlights the correlations between data points in the different analyzed environments. A prevalence of samples from the big room can be found in the lower central part of the plot, while the medium and small room samples are more present 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 completely separated and concentrated on the right side of the plot.

The same procedure is followed for materials, considering four object classes for clearness: aluminum plate, plastic bin, wooden door, and glass. In this case, a

[1] https://doi.org/10.5281/zenodo.6611037

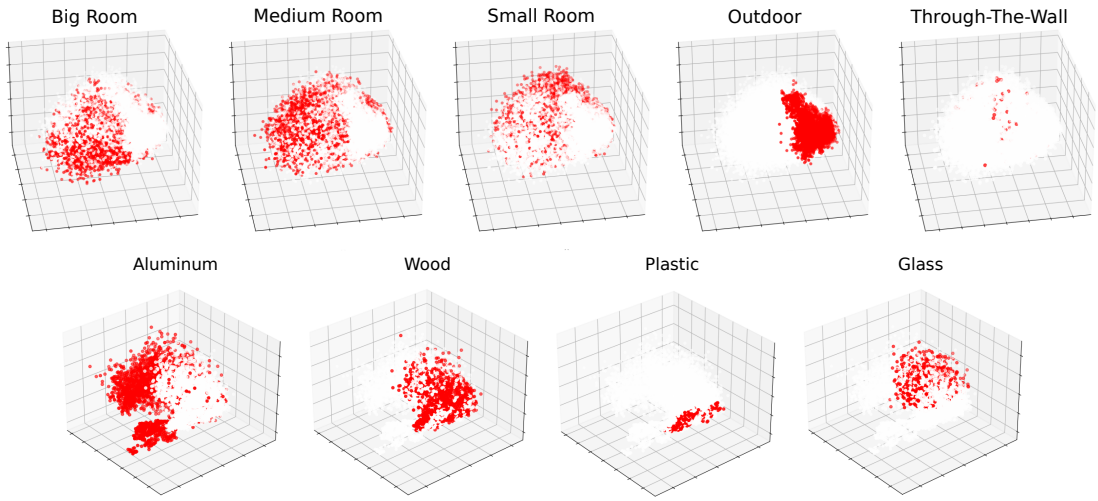


Figure 7.2: Principal Component Analysis (PCA), projecting the original 157 CIR dimensions into a three-dimensional space. It is clear how rooms cover a similar data space, completely separated by the outdoor scenario. Moreover, the same applies to materials, where more dense molecular structures affect the signal differently.

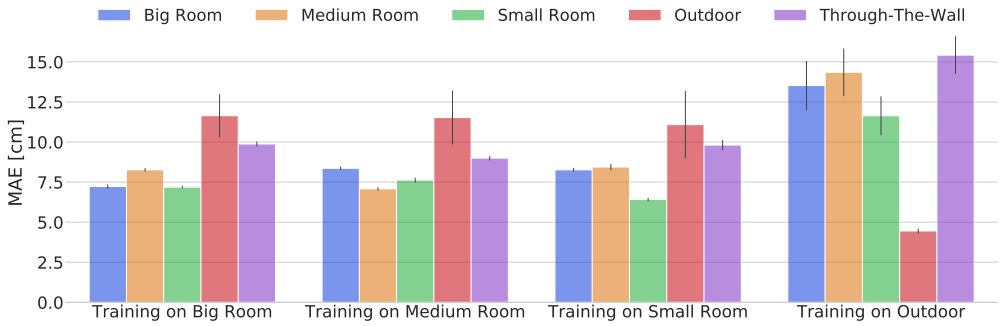


Figure 7.3: Analysis of generalization capabilities in different environments of a baseline representation learning model trained directly on the CIR waveform. Each bar represents the average MAE on the ranging estimation of 20 independent MLP models. Results show that rooms with different sizes and configurations lead to very similar results. Moreover, outdoor and TTW scenarios should be considered as separate settings and cannot be corrected without including appropriate samples in the training set.

remarkable separation is noticeable, as the metal samples occupy all the left part of the graph and light objects like plastic, wood, and glass take the right area. Moreover, the spatial distribution of wood occupies specific zones showing different features from plastic and glass. The presented qualitative analysis allows a first visual proof of the meaningfulness of 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 less accurate estimations for metal samples.

Finally, a Multi-layer Perceptron (MLP) is trained and tested on different splits of the dataset to assess the generalization capabilities of a baseline representation learning model. The model tries to compensate for the ranging measurement error directly from the CIR waveform, and its performance is measured as the mean absolute error (MAE) with respect to the ground truth. After the validity of the method is first verified on the whole dataset, a series of tests are conducted to study the effect of different environments and obstacles on the model performance. The network is trained on a specific set of data from the same room or with the same material and tested on other possible conditions. In this way, it is possible to state whether the approach holds an absolute generality concerning such factors. For what concerns environmental influence, resumed in Fig. 7.3, metrics show that rooms with different sizes and configurations lead to minimal changes in results (less than 2 cm) compared to those caused by outdoor measurements or more extreme conditions as TTW. Indeed, samples taken in open space show the worst generalization, because they are taken in a completely different scenario. The model struggles to adapt to a situation in which multipath components are completely absent, but an improvement is achieved in almost all cases.

For what concerns obstacles, we find a more marked distinction. As already emerged from PCA analysis, heavy materials have a very different impact on UWB signals with respect to wood, plastic, and glass. However, there is almost always an improvement in the raw MAE. That means that models can learn a way to compensate for part of the error independently from the obstacles. A dataset containing a sufficient number of examples for a wide variety of materials can lead to excellent results in many different scenarios.

7.2 Methodology

In this section, we propose a Deep Neural Network (DNN) to solve the range error mitigation problem. Moreover, we present some optimization and quantization techniques used to increase the computational efficiency of the network and deploy it on low-power devices. Since UWB are low-power localization devices directly connected to the mobile robot board, any error compensation technique should be applied locally on the platform to ensure real-time execution with a latency compatible with the control frequency of the robot. The method should also be as efficient as possible to ensure a low impact on the system overall energy and computational demand. In designing our solution, we mainly focus on optimizing the model to reduce memory occupancy and computational efforts during inference.

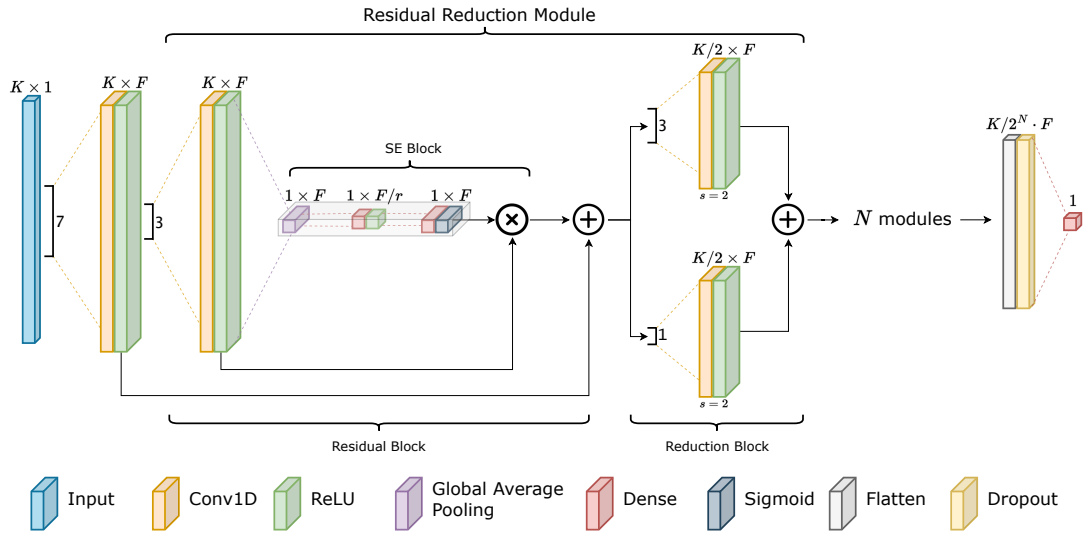


Figure 7.4: Overview of the REMNet architecture. The input of the model is the $K \times 1$ tensor representing the CIR of the measurement. The dimensionality is reduced by N subsequent RMMs, with a feature attention mechanism performed with an SE block. Finally, an FC layer composes the high-level extracted features and outputs the range error estimation.

7.2.1 Network design

We consider the following model for a generic UWB range measurement:

$$\hat{d} = d + \Delta d \tag{7.1}$$

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 the obstacles, giving, in general, worse performance in NLoS conditions.

We formulate the mitigation problem as a regression of the compensation factor Δd that should be subtracted from the measured range to obtain the actual distance between the two sensors. Therefore, we 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 $\boldsymbol{X} \in \mathbb{R}^{K \times 1}$, where K is the number of temporal samples. We call the proposed architecture Range Error Mitigation Network (REMNet). It is important to underline that we do not distinguish between LoS and NLoS measurements, but we let the network learn how to compensate for both conditions autonomously. Therefore, a classification of the measurements is not necessary, but the model implicitly performs it during the mitigation process. Such an approach allows obtaining an algorithm that is always beneficial and can be applied continuously onboard without the need for an additional classification step.

Due to the one-dimensional nature of the data, we select 1D convolutional layers as building blocks of the network. We first extract F low-level features with a 1D convolution from the input $\boldsymbol{X}$, obtaining a feature tensor with shape $K \times F$. The network architecture is then made of a stack of N Residual Reduction Modules (RRMs) that learn deep features while reducing the temporal dimensionality K. We develop this module adopting well-known strategies used in DL literature such as residual connections [87] and attention mechanism [99, 262]. All these methodologies have been proven to be effective in guaranteeing trainable and well-converging networks and are therefore suitable to be applied to the range error mitigation problem.

The core of the RRM is composed of a residual unit followed by a reduction block. Given a generic feature tensor $\boldsymbol{x_1} \in \mathbb{R}^{K \times F}$, an RRM performs the following operation:

$$\mathrm{RRM}(\boldsymbol{x_1}) = \mathrm{Red}(\mathrm{Res}(\boldsymbol{x_1})) = \mathrm{Red}(\boldsymbol{x_2}) \tag{7.2}$$

where Res and Red represent the residual and reduction blocks, respectively. In particular, the residual unit has a 1D convolution followed by a Squeeze-and-Excitation (SE) block [99] on the residual branch:

$$\boldsymbol{x_2} = \mathrm{Res}(\boldsymbol{x_1}) = \mathrm{SE}(\mathrm{Conv1D}(\boldsymbol{x_1})) + \boldsymbol{x_1} \tag{7.3}$$

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 themselves. It first squeezes the feature tensor with a GAP layer that aggregates the tensor along the temporal dimension K, obtaining a single statistic for each of the F features. The excitation step is then performed with a stack of one bottleneck FC layer that reduces the feature dimension F of a factor r, with ReLU activation, and another FC layer that restores the dimensionality to F with sigmoid activation. This activation outputs F independent scaling factors between 0 and 1 that are then multiplied with the input $\boldsymbol{x}$, allowing the network to focus on the most prominent features. Overall, the SE output is computed as:

$$\mathrm{SE}(\boldsymbol{x_1}) = \mathrm{FC}_2\Big(\mathrm{FC}_1\Big(\frac{1}{K}\sum_{i=1}^{K}\boldsymbol{x_1^{(i)}}\Big)\Big) \cdot \boldsymbol{x_1} \tag{7.4}$$

where $\boldsymbol{x_1^{(i)}}$ represent the i-th temporal feature of the tensor $\boldsymbol{x_1}$ and FC_1 and FC_2 represent the two FC blocks, that have $\boldsymbol{W_1} \in \mathbb{R}^{F \times F/r}$ and $\boldsymbol{W_2} \in \mathbb{R}^{F/r \times F}$ weight matrices respectively.

The residual unit is then followed by a reduction block, which halves the temporal dimension K with a convolution with stride $s = 2$. The reduction block again has a residual connection characterized by a 1D convolution with a kernel size of 1 and stride $s = 2$ to match the temporal dimension, with the main branch.

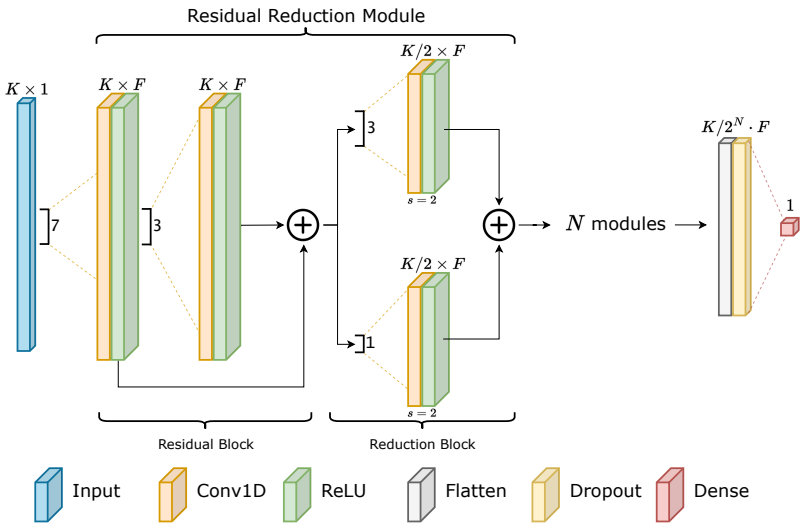


Figure 7.5: Overview of the modified REMNet architecture to ensure compatibility with the Arduino Nano 33 BLE Sense micro-controller. We remove the SE block to ensure compatibility with the target embedded board.

$$\text{Red}(\boldsymbol{x_2}) = \text{Conv1D}_1(\boldsymbol{x_2}) + \text{Conv1D}_2(\boldsymbol{x_2}) \tag{7.5}$$

where both Conv1D_1 and Conv1D_2 output F channels.

After N RRMs, we end up with a tensor with shape $K/2^N \times F$. We flatten it into a single vector, and we apply a Dropout layer to regularize the network and help generalization. Finally, an FC regression head with linear activation computes an estimate $\hat{y}$ of the compensation value $y = \Delta d$. An overview of the overall network architecture is presented in Fig. 7.4. It is worth noting that all the Conv1D layers in the model have ReLU non-linearity as activation function and are zero-padded so that the temporal dimension is reduced by the non-unitary strides of the reduction block, only.

7.2.2 Network optimization and quantization techniques

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, we investigate different graph optimization and quantization methods to both decrease model size and computational cost. An overview of optimization methods present in the literature is presented in Section 2.4. We produce five different versions of REMNet, depending on the adopted techniques:

1. the plain float32 network with no modifications;

2. graph optimization (G.O.) without quantization;
3. weight quantization to 16 bits, while activations are still represented as 32 bits floats;
4. 8-bit full integer quantization of both weights and activations;
5. 8-bit full integer quantization with modified architecture.

The full integer quantization strategy is the most radical to increase network efficiency by changing the representation of both weights and activations to 8-bit integers, greatly reducing memory and computational demands due to the high efficiency of integer computations. For details on this technique, refer to Section2.4. Moreover, we modify the network architecture to ensure compatibility with the ultra-low-power Arduino Nano 33 BLE Sense microcontroller, obtaining a second full-integer network (point 5). In particular, we simply remove all feature-attention SE blocks that boost the accuracy performance but compromise compatibility, obtaining the architecture shown in Fig. 7.5.

7.3 Experiments

In this section, we perform an experimental evaluation of the proposed architecture for efficient range error mitigation. Moreover, we test the accuracy and performance of different optimized versions of the network on disparate heterogeneous devices collecting energy and computational requirements.

7.3.1 Experimental setting

In the following experiments, we employ the presented dataset of Section 7.1 keeping aside the medium size room as the testing set. Indeed, instead of performing a stratified sampling of the available data, in the light of the evidence of Section 7.1.1, we decide 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 36023 and 13210 training and testing data points, respectively, keeping aside TTW and outdoor measurements. Moreover, due to the LoS very different nature, we report results on both the overall testing set and on the test LoS samples only, in order to evaluate the network capability to recognize this condition and act accordingly.

A final test consists in using the best-developed model for a 3D positioning task to assess the range mitigation effect on localization accuracy. The medium room is chosen as the testing environment, as its samples have not been used to train the network. Four UWB anchors are placed in the room, and a fixed tag is put in the center. First, the laser tracker precisely measures the position of all the nodes to

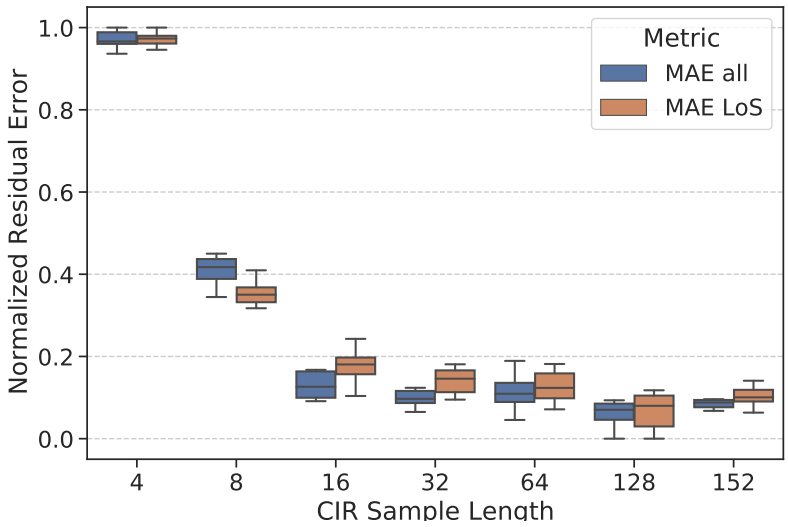


Figure 7.6: Network performance with different CIR sizes K, starting from the dimension suggested by [27]. For each test, we report overall and LoS MAE on the medium size room test set, normalized with respect to the worst results at $K = 4$: 0.0946 (all) and 0.0542 (LoS). Progressively training with a reduced number of input features degrades the performance of the network. An input with 8 dimensions appears to be the minimum required to obtain an acceptable range error estimation.

provide ground truth, then the acquisition of the data begins. Two situations are taken into consideration, 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 3D position of the tag employing a simple Gauss-Newton non-linear optimization algorithm.

All network hyperparameters are selected with an initial random search, followed by a grid search exploration of the most promising ones, to fine-tune them and find a compromise between accuracy and efficiency. Indeed, working at the architecture level is crucial to satisfy the constraints given by the studied application. We set the number of filters $F = 16$, and the number of reduction modules $N = 3$ with $r = 4$. We always use a kernel size of 3 for all the Conv1D layers, unless where otherwise shown in Fig. 7.4. We set the dropout rate in the regression head at 0.4. The resulting network has an efficient and highly optimized architecture with 6349 trainable parameters. On the other hand, the modified architecture to target the Arduino Nano 33 BLE Sense microcontroller has 5905 parameters, since it lacks the SE blocks.

Finally, to select the optimal number of input features, as shown in Fig. 7.6, we progressively reduce the number of input temporal dimensions K, while training and testing the network. All points are the average result of ten consecutive independent trials. The experimentation shows that $K = 8$ is the minimum number of

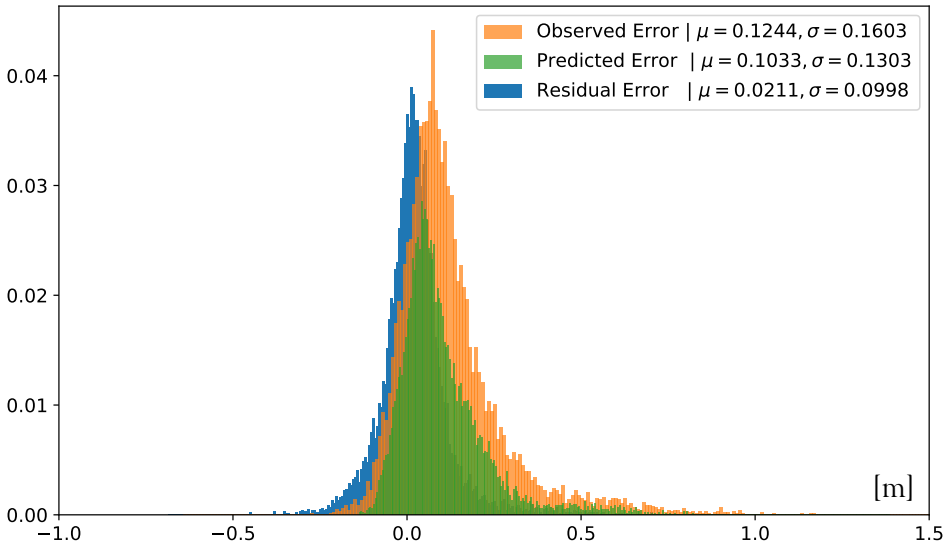


Figure 7.7: REMNet test samples error distribution in meters. It is possible to notice how the residual range error distribution (blue histogram) is almost Gaussian. That greatly improves the optimality and simplicity of the subsequent iterative localization algorithm [216]. The histograms are normalized and with 300 bins each.

temporal dimensions required for the network to obtain an acceptable range error estimation. Moreover, we empirically find that an input CIR of 152 elements, as suggested by [27], is redundant and could even slightly reduce the model performance. On the other hand, fewer dimensions of 128 tend to almost linearly degrade the network performance. For this reason, we refer to $K = 128$ CIR as the standard REMNet model.

The Adam optimization algorithm [122] is used for all the training processes, with default parameters. The optimal learning rate $\eta = 3 \times 10^{-4}$ is experimentally selected using the methodology described in [230]. We train for a total of 30 epochs with a batch size of 32 and using MAE as both loss function and metric. We employ the TensorFlow 2 framework [1] to train the network on a workstation with 32GB of RAM and an Nvidia 2080 Super GPU. The overall training process can be performed in less than 10 minutes.

7.3.2 Quantitative results

The medium room data samples have an error distribution with mean $\mu_{\text{obs}} = 0.1204$ m and standard deviation $\sigma_{\text{obs}} = 0.1602$ m. Fig. 7.7 shows the error distribution, along with the one estimated by the standard REMNet architecture and the residual. It is possible to notice how the network is able to almost completely compensate for the offset of the original range error and reduce the standard deviation

	All				LoS			
	MAE	$\boldsymbol{R^2}$	$\boldsymbol{\mu_{\text{res}}}$	$\boldsymbol{\sigma_{\text{res}}}$	**MAE**	$\boldsymbol{R^2}$	$\boldsymbol{\mu_{\text{res}}}$	$\boldsymbol{\sigma_{\text{res}}}$
Observed	0.1391	-0.6021	0.1244	0.1603	0.0841	-0.3537	0.0553	0.0929
Mean prediction	0.1096	0.0	0.0	0.1603	0.07297	0.0	0.0	0.0929
SVM	0.0778	0.4326	0.024	0.1183	0.0592	0.3231	0.0085	0.0756
MLP	0.0766	0.4635	0.0172	0.1161	0.0592	0.3468	0.0076	0.0747
CNN-1D [27]	0.0918	0.1750	0.0429	0.1391	0.0659	0.1581	0.0111	0.0846
REMNet_1 (float32)	0.0678	0.5946	0.0211	0.0998	0.0550	0.4183	0.0172	0.0688
REMNet_2 (float32 G.O.)	0.0678	0.5946	0.0211	0.0998	0.0550	0.4183	0.0172	0.0688
REMNet_3 (float16)	0.0678	0.5946	0.0211	0.2066	0.0560	0.3770	-0.0105	0.1302
REMNet_4 (int8)	0.0709	0.5320	0.0148	0.1086	0.0567	0.3925	0.0062	0.0722
REMNet_5 (int8 modified)	0.0706	0.5338	0.0116	0.1088	0.0572	0.3876	0.0061	0.0725

Table 7.2: Performance of the five variants of REMNet, and other baseline methods. It is possible to notice how the different transformations barely affect the range error estimation capability of the network. All errors are in meters.

of 37.7%.

Tab. 7.2 shows the performance of REMNet in its five different variations, together with three simple models (SVM, MLP, and CNN-1D) included as a reference. Moreover, as baselines, we report the result of zero prediction, which coincides with the observed error distribution, and the test average prediction, which we refer to as the worst possible regressor. As metrics, we report the MAE loss of the predictions, the coefficient of determination R^2 (that is 0 for a result as good as the mean prediction), and mean and standard deviation of the residual error distribution in meters. We report the evaluation on all the test samples and on the LoS samples only. For support vector machine (SVM) and MLP, we adopt 128 CIR input dimensionality for a fair comparison. We use radial basis function (RBF) as SVM kernel, and a 3-layer architecture with 64 hidden neurons, ReLU activation and Batch Norm for the MLP. Instead, for the CNN-1D proposed by [27], we feed the network with 152 bins as suggested in the paper. The training hyperparameters are the same used for REMNet. It is noticeable how REMNet has better performances than other methodologies even with a highly efficient architecture.

Moreover, the network can easily detect LoS input signals and apply a small correction factor that takes into account the multipath effect. That is proved by the MAE which has a percentage improvement of 34.6%. On the other hand, MAE for the overall test samples is improved by 51.3%, reducing the average error to 0.0678 m.

Graph optimization techniques and different weight quantization levels are both examined, starting from the reference network. Even if there is a slight degradation of the overall metrics, these changes are mostly negligible. Moreover, it is possible to notice that the full-integer quantization (REMNet_4) decreases the NLoS MAE only of the 4.6%. That opens the possibility to achieve effective range mitigation

	All			LoS		
	SR	**MR**	**BR**	**SR**	**MR**	**BR**
Training samples	31632	36023	30811	31632	36023	30811
Test samples	17601	13210	18422	1862	1059	1625
μ_{obs}	0.0881	0.1244	0.1058	0.0186	0.0553	0.0318
μ_{res}	0.0089	0.0211	0.0167	0.0099	0.0172	0.0024
σ_{obs}	0.1508	0.1603	0.1851	0.0643	0.0929	0.0574
σ_{res}	0.1137	0.0998	0.1497	0.0555	0.0688	0.0534
MAE_{obs}	0.1069	0.1391	0.1253	0.0841	0.0841	0.0528
MAE_{res}	0.0634	0.0678	0.0734	0.0418	0.0550	0.0415
R^2	0.4282	0.5946	0.3374	0.2319	0.3770	0.1335

Table 7.3: REMNet cross-validation results with the three different rooms, small room (SR), medium room (MR) and big room (BR). Each column presents NLoS and LoS results for the room excluded by the training procedure and used as a test. All errors are in meters.

with integer operations only and almost negligible impact on the overall application. Indeed, extreme weight quantization implies a smaller model size with less memory usage, an important latency reduction, and the possibility of using highly efficient neural accelerators.

Finally, for a matter of completeness, Tab. 7.3 presents results obtained with cross-validation on the three different room sizes. We independently train the standard REMNet on two rooms and test on the third and report the NLoS and LoS performance. In accordance with the conclusions of Section 7.1.1, REMNet achieves comparable range error mitigation in the three different configurations.

Power and latency results

In this section, we test the different optimized networks on several devices and hardware accelerators, annotating power, and computational request. We consider the Nvidia Jetson Nano, the Coral Dev Board, and the Arduino Nano 33 BLE Sense, as deploy platforms, testing both CPUs and ML accelerators, when available. For details on the HW specifications of the selected devices, refer to Tab. 2.1. Moreover, we report results on an Nvidia RTX 2080 GPU as a reference. We adopt two standard libraries for network deployment, TFLite[2] and TensorRT[3] to produce the optimized models. Both are directly integrated into the TensorFlow framework and are specifically conceived to target different hardware platforms. In particular,

[2] https://tensorflow.org/lite

[3] https://developer.nvidia.com/tensorrt

Device	G.O.	W.P.	Latency [ms]	V_{cc} [V]	I_{run} [A]	P_{run} [W]	E_{inf} [mJ]	Size [kB]
RTX 2080	N	FP32	19.7 ± 0.23	-	-	32	617.6	250.0
	Y	FP32	0.69 ± 0.13	-	-	20	138.0	613.0
	Y	FP16	0.54 ± 0.09	-	-	18	97.2	615.0
Coral Dev Board (CPU)	N	FP32	16.9 ± 0.03	5.0	0.60	3.0	50.7	250.0
	Y	FP32	12.2 ± 0.03	5.0	0.60	3.0	36.6	40.7
	Y	FP16	11.2 ± 0.03	5.0	0.60	3.0	33.6	33.9
	Y	INT8	6.23 ± 0.02	5.0	0.60	3.0	18.7	32.7
Jetson Nano (CPU)	Y	INT8	4.71 ± 0.01	5.0	0.70	3.5	16.5	32.7
Jetson Nano (GPU)	Y	FP32	5.36 ± 0.05	5.0	0.86	4.30	23.0	615.0
	Y	FP16	5.18 ± 0.04	5.0	0.85	4.25	22.0	613.0
Coral Dev Board (Edge TPU)	Y	INT8	0.51 ± 0.01	5.0	0.73	1.8	0.9	70.54
Arduino Nano 33 BLE Sense	Y	INT8	47.17 ± 0.01	3.3	0.016	0.053	2.5	23.02

Table 7.4: Comparison between different devices energy consumption and inference performances. Graph optimization (G.O.) and weight precision (W.P.) reduction further increase the efficiency of REMNet, helping to deal with energy, speed, size, and cost constraints.

we target Cortex-A53 (Coral Board), A57 (Jetson Nano) CPUs, Edge TPU and Arduino with TFLite, and the Nvidia RTX 2080 and 128-core Maxwell (Jetson Nano) GPUs with TensorRT.

Experimentation results are summarized in Tab. 7.4, where latency, run-time voltage, current and power are shown, together with the energy required for a single inference and the size of the model in kB. It is possible to notice that, due to the high efficiency of the proposed architecture, all configurations satisfy a sufficient inference speed compliant for an effective range error mitigation solution. Nevertheless, the different optimization techniques applied have a high impact on the energy consumed by the network. Indeed, considering experimentation performed with the Cortex-A53 (Coral Board CPU), optimization can reduce the energy consumption by a factor of more than three, starting with an initial value of 50.7 mJ down to 18.7 mJ, with a reduction of 63%. Moreover, the model size is greatly reduced from 250 kB to 32.7 kB. That implies a smaller storage size and less RAM at run-time, freeing up memory for the main application where UWB localization is needed. Moreover, as further highlighted by Fig. 7.8 and the results of the previous subsection, the Edge TPU neural accelerator with a full-integer model appears the preferable solution for deployment. With only 0.51 ms of latency, 1.8 W power, and 0.9 mJ energy consumption, it barely impacts the performance of the overall application, allowing to exploit duty cycling and energy-saving techniques. Finally, if there are strong constraints in terms of size and low-power requirements, the Arduino-based solution appears to be very interesting, since it is based on a 3.3 V microcontroller and only needs 53 mW at run-time. The only drawback is that, since it does not have a specific ML accelerator, it is the slowest device analyzed, with its 47 ms required per inference. Overall, the already efficient design of our architecture, in conjunction with 8-bit weight precision, graph optimization techniques, and embedded deployment, makes DL a feasible solution for an effective

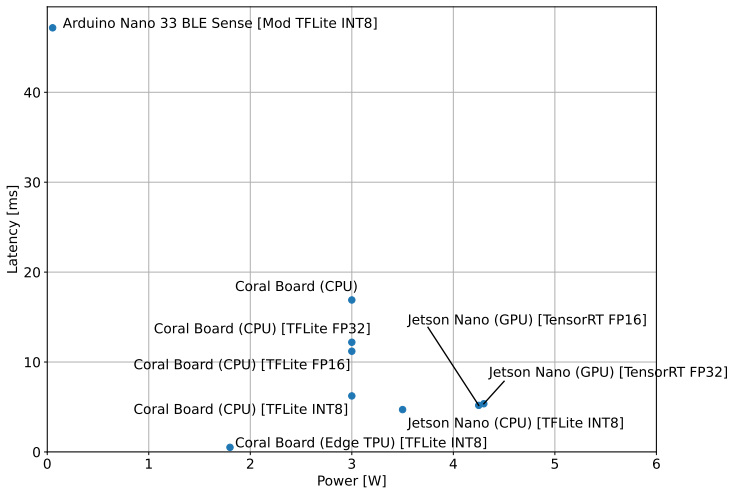


Figure 7.8: Power and latency are two important constraints for effective range error mitigation. Indeed, error correction is performed progressively over all the received anchor signals on board the platform connected with the tag. Without a highly optimized and efficient correction model, range error mitigation would not be applicable.

range error mitigation for UWB at the edge.

Trilateration results

As described in Section 7.3.1, the effect of the proposed method is lastly verified by using the full-integer model REMNet_4 deployed on the Coral Dev Board Edge TPU for a 3D positioning task, in which the results obtained from raw range estimates are compared to those achieved with our mitigation model in the loop. We perform an experiment in pure LoS and one with strong NLoS conditions. The results are summarized in Tab. 7.5, while Fig. 7.9 gives a graphical representation of the NLoS results.

Regarding the LoS case, the positioning system already achieves good precision by itself with a very low range MAE and, consequently, a low position MAE. In this case, the effect of mitigation is irrelevant, causing a slight increment of ranging error but a slight decrease in positioning error. So, as expected, the model learns to apply very slight corrections to LoS samples, avoiding worsening already good measurements. Instead, the NLoS scenario shows a significant improvement, as the range MAE is more than halved, reaching a value that is comparable to the LoS case and confirming previous results.

Consequently, the error in the position estimation is strongly reduced, going from 57.7 cm to 18.2 cm. Although the final accuracy is still significantly higher than the one found in the LoS case, a reduction of 68% is considered a significant

	LoS		NLoS	
	Range MAE	**Position MAE**	**Range MAE**	**Position MAE**
Raw UWB	0.0388	0.0703	0.1129	0.5772
REMNet_4 (int8)	0.0465	0.0679	0.0571	0.1817

Table 7.5: Results obtained from the positioning test in the test medium room. For each test, the MAE is reported for both the range estimates and the final position result, in order to highlight the effect of the former on the latter. All errors are in meters.

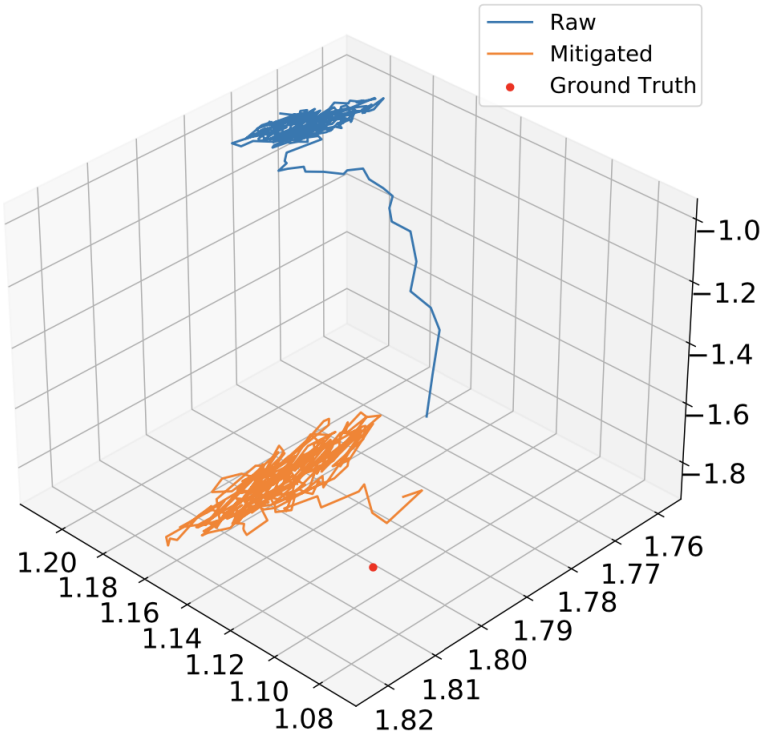


Figure 7.9: Comparison between 3D position estimation of a fixed tag in strong NLoS conditions. In light blue are the results obtained from raw range measurements, and in orange the ones achieved with our full-integer model.

result. Indeed, our approach allows to achieve suitable precision for many kinds of indoor robotic applications showing good generalization to unknown environments.

Chapter 8

Action Transformer: A Self-Attention Model for Short-Time Pose-Based Human Action Recognition

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 [203], surveillance and security [258], autonomous vehicles [22], and automatic video captioning [254]. Most previous works dealing with HAR adopted datasets characterized by samples with a long temporal duration [103, 249, 154]. Indeed, traditional datasets used by previous works, such as 3D NTU RGB+D [222, 144], or Kinetics-Skeleton [31, 270], include long temporal sequences which must be entirely scanned to make the correct classification. 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, in this work, we focus on short-time HAR, which aims at continuously classifying actions within short past time steps (up to a second). This approach is fundamental to target real-time applications: in robotics, for example, the HAR problem should be solved promptly to react to sudden behavioral changes, only relying on near past information.

Works in the HAR field can be mainly subdivided into two broad categories: video-based and depth-based methodologies [231]. In both, the input consists of a sequence of points representing body joints, extracted from the corresponding RGB frame [137], 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

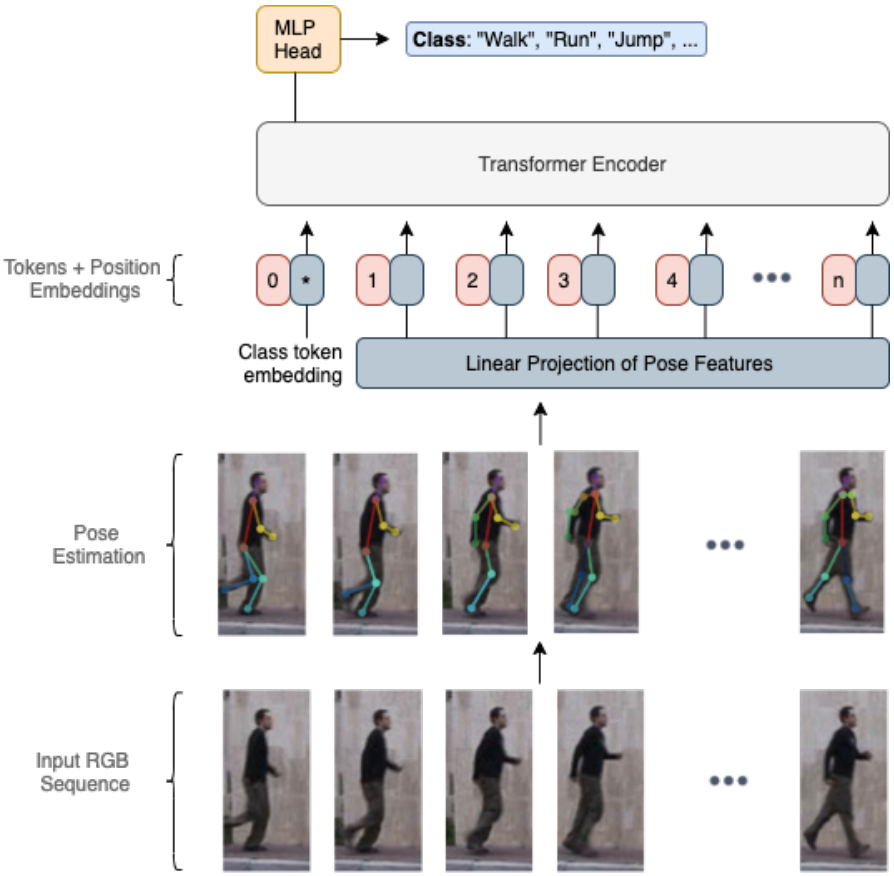


Figure 8.1: Overview of the Action Transformer architecture. Pose estimations are linearly projected to the dimension of the model, and together with the class token, they form the input tokens of the transformer encoder. As for Vision Transformer models [59, 242, 47], a learnable positional embedding is added to each input token. Then, only the output class token is passed through an MLP head to obtain the final class prediction.

(skeletal data), such as those captured by Kinect sensors [222, 144], that can be possibly projected onto the 2D space [31, 270]. On the opposite, in this work, we focus on a video-based analysis, where body joints are already provided as 2D coordinates (pose data) by pose detection algorithms such as OpenPose [30] and PoseNet [190]. This characteristic makes 2D HAR methodologies applicable in a great range of applications using a simple RGB camera. Conversely, skeletal data require particular sensors to be acquired, such as Kinect or other stereo-cameras. That raises substantial limitations, such as availability, cost, limited working range (up to 5-6 meters in the case of Kinect [130]), and performance degradation in outdoor environments.

Angelini et al. first collected the MPOSE dataset for video-based short-time

HAR [13], 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 [117], which exploits a combination of 1D convolutions, LSTM [94], and Squeeze-and-excitation attention [99]. The same authors successively applied their approach to anomaly detection [15, 14] and expanded MPOSE with the novel ISLD and ISLD-Additional-Sequences datasets. Conversely, Yan et al. [270] first applied OpenPose to extract 2D poses from the Kinetics-400 RGB dataset [31] and used graph convolutions to capture spatial and temporal information.

Recently, many models for HAR have proposed the integration of self-attention mechanisms with convolutional and recurrent blocks to improve the accuracy of models. For example, Cho et al. [41] first applied self-attention [250] to the skeletal-based HAR problem. More recently, Plizzari et al. [192], inspired by Bello et al. [21], 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. However, solutions that rely exclusively on self-attention blocks have not been investigated for this task yet.

In this work, we propose a new model for HAR called the Action Transformer (AcT), schematized in Fig. 8.1, inspired by the simple and prior-free architecture of the Vision Transformer (ViT) [59]. Optimized versions of the Transformer have been developed in the literature for real-time and embedded applications [24], proving that this architecture is suitable for Edge AI purposes. With AcT, we apply a pure Transformer encoder architecture for HAR, obtaining an accurate and low-latency model for real-time applications. We study the model at different scales to investigate the impact of the number of parameters and attention heads. We propose the new single-person, short-time action recognition dataset MPOSE2021 as a benchmark and exploit 2D human pose representations provided by two existing detectors: OpenPose [30] and PoseNet [190]. We compare AcT with other state-of-the-art baselines to highlight the advantages of the proposed approach. To highlight the effectiveness of self-attention, we conduct a model introspection providing visual insights of the results and study how a reduction of input temporal sequence length affects accuracy. We also conduct extensive experimentation on model latency on low-power devices to verify the suitability of AcT for real-time applications.

8.1 The MPOSE2021 dataset

We introduce MPOSE2021, a dataset for real-time short-time HAR, suitable for both pose-based and RGB-based methodologies. It includes 15429 sequences from 100 actors and different scenarios, with limited frames per scene (between 20 and 30). In contrast to other publicly available datasets, the peculiarity of

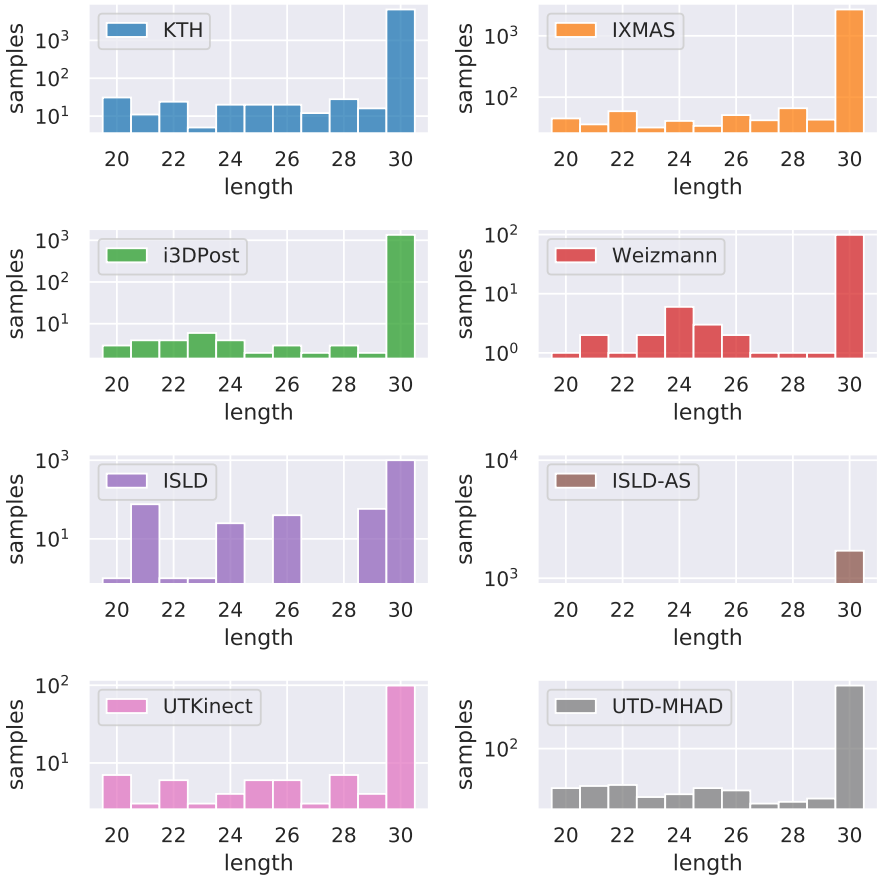


Figure 8.2: MPOSE2021 sequence length distribution for the different precursor datasets. It is possible to notice how most of them present a mode of 30 frames per scene.

having a constrained number of time steps stimulates the development of real-time methodologies that perform HAR with low latency and high throughput. As in [13, 14, 15], video data have been previously collected from popular HAR datasets (defined as *precursors*), i.e. Weizmann [77], i3DPost [71], IXMAS [260], KTH [221], UTKinetic-Action3D (RGB only) [266], UTD-MHAD (RGB only) [34], ISLD, and ISLD-Additional-Sequences [13].

Due to the heterogeneity of actions across different datasets, labels are remapped to a list of 20 common 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 retaining tail samples with more than 20 frames. A visual representation of the number of frames per sample for each sub-dataset of MPOSE2021 is shown in Fig. 8.2. The peculiarity of a reduced number of time steps contrasts with other publicly available datasets and

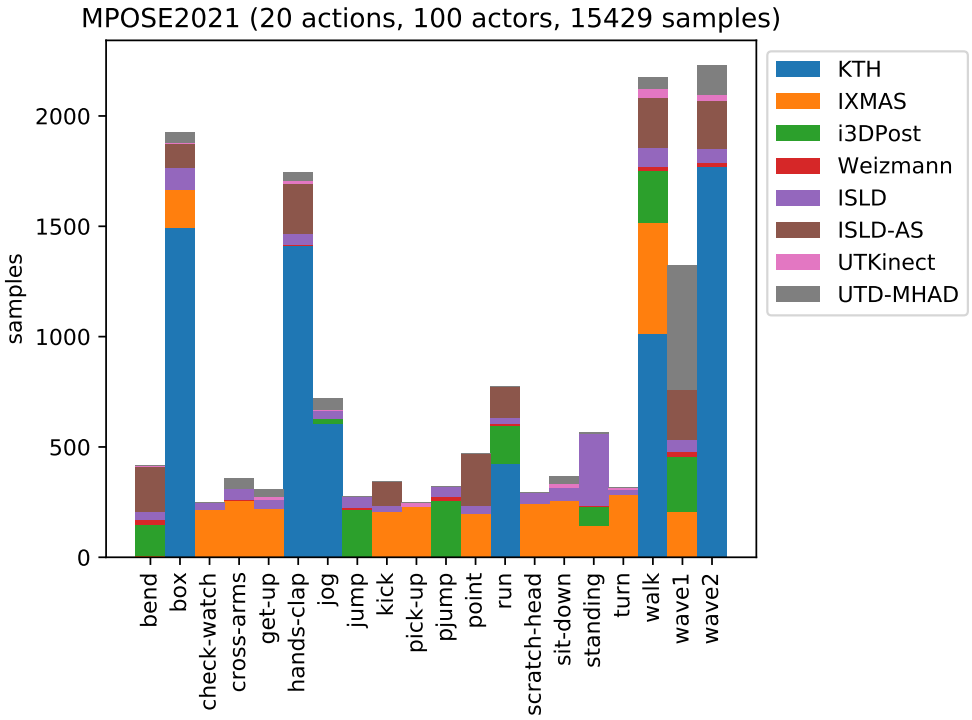


Figure 8.3: The number of samples of MPOSE2021 divided by action. The colors show the distribution of precursor datasets among classes, highlighting the unbalanced nature of the data. The final dataset contains 15429 samples where each sample represents one of 100 actors performing one of 20 actions.

stimulates the development of methodologies that require low latency to perform a prediction. That would largely benefit many real-world applications requiring real-time perception of the actions performed by humans nearby.

Subsequently, clips not containing a single action are discarded. Moreover, ambiguous clips are relabelled whenever possible or discarded otherwise. This process leads to 15429 samples, where each sample represents 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. 8.3, which also shows the distribution of precursor datasets.

OpenPose [30] and PoseNet [190] are used to extract landmarks from MPOSE2021 video samples. 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 rest of them for training. In this process, we ensure that all the splits cover all the labels. Roughly, 23% of samples are in the test set and 77% in the training set, for all the three splits. This division makes the proposed dataset a challenging

benchmark to effectively assess and compare the accuracy and robustness of different methodologies. Moreover, the suggested evaluation procedure requires training a target model on each split using 10 stratified validation folds got from the training set and averaging the obtained results on the test set. That makes possible to produce statistics and reduces the possibility of overfitting the split testing set with an accurate choice of hyperparameters.

With MPOSE2021, we aim to provide an end-to-end and easy-to-use benchmark to robustly compare state-of-the-art methodologies for the short-time human action recognition task. We thus release a code repository[1] to access the different levels of the dataset (video, RGB frames, 2D poses). Moreover, we open source a practical Python package to access, visualize and preprocess the poses with standard functions. The Python package can be easily installed with the command `pip install mpose`.

8.2 Methodology

Given a video input sequence with T frames $\boldsymbol{X}_{\text{rgb}}$, we extract the human poses using a multi-person 2D pose estimation network that extracts tensors of shape $N \times T \times P$, where N is the number of human subjects present in the frame and P is the number of keypoints predicted by the network:

$$\boldsymbol{X}_{\text{2Dpose}} = \varphi_{\text{2Dpose}}(\boldsymbol{X}_{\text{rgb}}) \tag{8.1}$$

where φ_{2Dpose} represents the pose estimation model. AcT receives as input a 1D sequence of token embeddings, so each of the N pose tensors $\boldsymbol{X}_{\text{2Dpose}}$, with shape $T \times P$, is separately processed by the AcT network. On the other hand, at inference time, all detected poses in the video frame can be batch-processed by the AcT model, simultaneously producing a prediction for all N subjects.

The T poses are mapped to a higher dimension D_{model} using a linear projection map $\boldsymbol{W}_{\text{proj}} \in \mathbb{R}^{P \times D_{\text{model}}}$, obtaining the sequence $\boldsymbol{X}_{\text{proj}} \in \mathbb{R}^{T \times D_{\text{model}}}$.

Similarly to the ViT model [59] (see Section 2.3.2), an additional trainable embedding $\boldsymbol{x}^0_{\text{cls}} \in \mathbb{R}^{1 \times D_{\text{model}}}$ is added to the input sequence. This class token [CLS] forces 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 $\boldsymbol{X}_{\text{pos}} \in \mathbb{R}^{(T+1) \times D_{\text{model}}}$ added to all tokens.

The linearly projected tokens and [CLS] are fed to a standard Transformer encoder [250] $\varphi_{\text{enc}}(\cdot)$ of L layers with a post-norm Layer Norm [17]:

$$\boldsymbol{X}^L = \varphi_{\text{enc}}(\boldsymbol{X}^0) = \varphi_{\text{enc}}([\boldsymbol{x}^0_{\text{cls}}; \boldsymbol{X}_{\text{proj}}] + \boldsymbol{X}_{\text{pos}}) \tag{8.2}$$

[1] https://github.com/PIC4SeRCentre/MPOSE2021

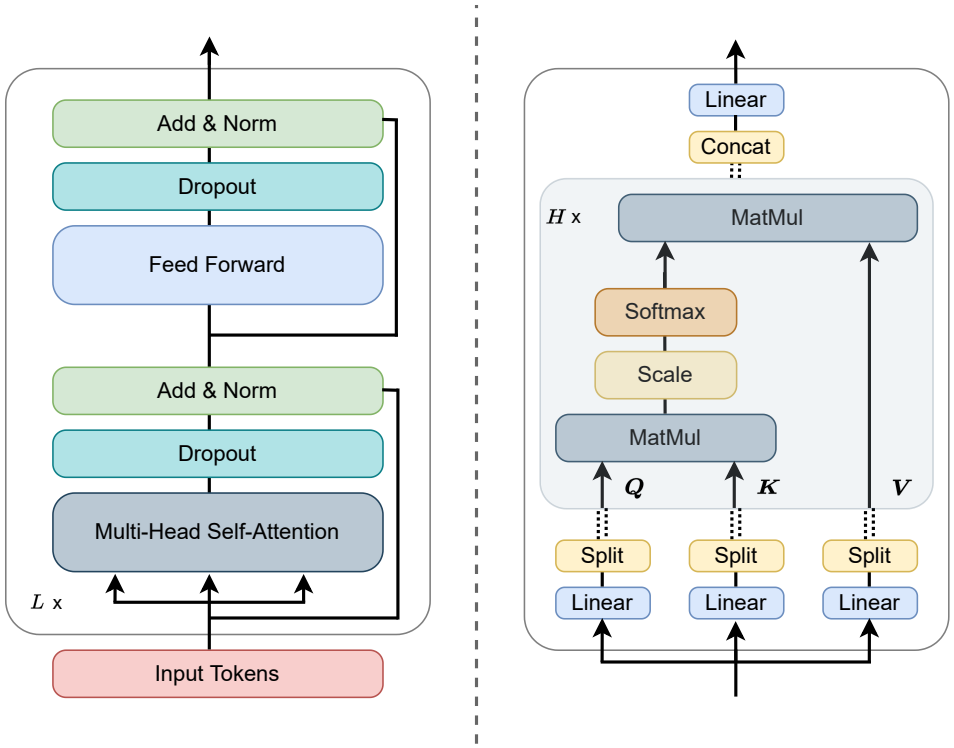


Figure 8.4: Transformer encoder layer architecture (left) and schematic overview of a multi-head self-attention block (right). Input tokens go through L encoder layers and H self-attention heads. For details see Section 2.3.2.

where $\boldsymbol{X}^l \in \mathbb{R}^{(T+1)\times D_{model}}$ is the overall representation produced by the model at its l layer, with $l = 0$ meaning the input and $l = L$ the output. The encoder network $\varphi_{\text{enc}}(\cdot)$ follows the classic Transformer model presented in Section 2.3.2 and depicted in Fig. 8.4.

Finally, only the output [CLS] token $\boldsymbol{x}^L_{\text{cls}}$ is fed into a linear classification head MLP, with a single hidden layer with dimensionality D_{mlp}, that performs the final class prediction:

$$\hat{\boldsymbol{z}} = \text{MLP}(\boldsymbol{x}^L_{\text{cls}}) \tag{8.3}$$

where $\hat{\boldsymbol{z}} \in \mathbb{R}^C$ is the output logit vector of the model, and C is the number of classes. At training time, the supervision signal comes only from the [CLS] token, while all remaining T tokens are the only input of the model. It is important to notice how the nature of the network makes possible to accept a reduced number of frames as input even if trained with a fixed T. That gives an additional degree of freedom at inference time, making AcT more adaptive than other existing models.

The resulting network, shown in Fig. 8.1, is a lightweight solution capable of predicting actions for multiple people in a video stream with high accuracy. The

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

advantage of building on 2D pose estimations enables effective real-time performance with low latency and energy consumption.

In order to reduce the number of hyperparameters and linearly scale the dimension of AcT, we fix $D_{model}/H = 64$, varying H (number of self-attention heads), D_{mlp} (dimensionality of the MLP head), and L (number of encoder blocks) to obtain different versions of the network. A simple grid search using train and validation sets is performed to determine lower and upper bounds for the four parameters. In Tab. 8.1, we summarize the four AcT versions with their respective number of parameters. The four models (micro, small, medium, and large) differ in their increasing number of heads and layers, substantially impacting the number of trainable parameters.

8.3 Experiments

This section describes the main experiments conducted to study the advantages of using a fully self-attentional model for 2D pose-based HAR. First, the four variants of AcT are compared to existing state-of-the-art methodologies and baselines on the MPOSE2021 dataset. Only for a specific comparison with ST-TR [192] and MS-G3D [148], we use additional ensemble versions of our model named AcT-μ (xE), E being the number of ensembled instances. Then, we further analyze the behavior of the network in order to get a visual insight of the attention mechanism and study the performance under a reduction of temporal information. Finally, we study model latency for all the designed architectures with two different CPU types, proving that AcT can easily be used for real-time applications.

8.3.1 Experimental settings

In the following experiments, we employ both the OpenPose and PoseNet versions of the MPOSE2021 dataset. Either set of data has $T = 30$ temporal frames and $P = 52$ or 68 pose features, respectively. In particular, for OpenPose we follow the same preprocessing of [13] obtaining 13 keypoints with four parameters each

Training		Regularization	
Training epochs	350	Weight decay	1e-4
Batch size	512	Label smoothing	0.1
Optimizer	AdamW	Dropout rate	0.3
Warmup epochs	40%	Random flip	50%
Step epochs	80%	Random noise σ	0.03

Table 8.2: Hyperparameters used in AcT experiments.

(position x, y and velocities v_x, v_y), while PoseNet data samples contain 17 keypoints with the same information. To find the most promising hyperparameters, we perform a grid search analysis on the Split1 dataset using the AcT-S model. We subdivide the 12562 training samples in 9421 samples for the grid search training and 3141 samples for validating the results and select the hyperparameters. We do not use any of the testing samples during this process.

Tab. 8.2 all the hyperparameters related to the training procedure. The AdamW optimization algorithm [153] is employed for all training with the same scheduling proposed in [250], but with a step drop of the learning rate η to 1×10^{-4} at a fixed percentage (80%) of the total number of epochs. We employ the TensorFlow 2 [1] framework to train the proposed network on a PC with 32-GB RAM, an Intel i7-9700K CPU, and an Nvidia 2080 Super GP-GPU. Following the previously defined benchmark strategy, the total training procedure for the four versions takes approximately 32 hours over the three different splits. We exploit publicly available code for what concerns other SOTA models and use the same hyperparameters and optimizer settings described by the authors in almost all the cases. The only exception is made for learning rate, epochs number, and batch size to adapt the methodologies to our dataset and obtain better learning curves. All results, training, and testing code for the AcT model are open source and publicly available[2].

8.3.2 Action recognition on MPOSE2021

We extensively experiment on MPOSE2021 considering some baselines, common HAR architectures, and our proposed AcT models. As presented in Section 8.1, we adopt a stratified cross-validation and report the testing metrics (mean and standard deviation) for 10 models trained using different validation splits to obtain statistically relevant results. The validation splits are constant for all the model variants and correspond to 10% of the train set, maintaining the same class distribution. The benchmark is executed for both OpenPose and PoseNet data and

[2] https://github.com/PIC4SeRCentre/AcT

repeated for all the three train/test splits provided by MPOSE2021. The baselines chosen for the benchmark are an MLP, a fully convolutional model (Conv1D), and REMNet, which is a more sophisticated 1D convolutional network with attention and residual blocks for time series feature extraction, see Section 7.2. In particular, the MLP is designed as a stack of three FC layers with 512 neurons each, followed by a Dropout layer and a final FC layer with as many output nodes as classes. Instead, the Conv1D model is built by concatenating five 1D convolutional layers with 512 filters alternated with Batch Norm stages and followed by a GAP operator, a Dropout layer, and an FC output stage as in the MLP. Finally, the configuration used for REMNet consists of two Residual Reduction Modules (RRM) with a filter number of 512, followed by Dropout and the same FC output layer as in the other baselines.

Regarding SOTA comparisons, four popular models used for multivariate time series classification, and in particular HAR, are reproduced and tested. Among those, MLSTM-FCN [117] combines convolutions, spatial attention, and an LSTM block, and its improved version ActionXPose [13] uses additional preprocessing, leading the model to exploit more correlations in data and, hence, be more robust against noisy or missing pose detections. On the other hand, MS-G3D [148] uses spatial-temporal graph convolutions to make the model aware of spatial relations between skeleton keypoints, while ST-TR [192] joins graph convolutions with Transformer-based self-attention independently applied to space and time. As the last two solutions also propose a model ensemble, these results are further compared to AcT ensembles AcT-μ (x E), made of 2, 5, and 10 single-shot models. We also report the achieved balanced accuracy for each model and use it as the primary evaluation metric to account for the uneven distribution of classes.

The results of the experimentation for OpenPose are reported in Tab. 8.3 and, in synthesis, in Fig. 8.5. The fully convolutional baseline strongly outperforms the MLP, while REMNet proves that introducing attention and residual blocks further increases accuracy. As regards MLSTM-FCN and ActionXPose, it is evident that explicitly modeling both spatial and temporal correlations, made possible by the two separate branches, slightly improves the understanding of actions with respect to models like REMNet. MS-G3D, in its joint-only (J) version, brings further accuracy improvement by exploiting graph convolutions and giving the network information on the spatial relationship between keypoints. On the other hand, T-TR (ST-TR model using only the temporal branch), shows performance comparable to all other single-shot models, despite, as MS-G3D, taking advantage of graph information.

The proposed AcT model demonstrates the potential of pure Transformer-based architectures, as all four versions outperform other methodologies while also showing smaller standard deviations. Moreover, even the smallest AcT-μ (227k parameters) is able to extract general and robust features from temporal correlations in sequences. Increasing the number of parameters, a constant improvement in

MPOSE2021 OpenPose		Split 1		Split 2		Split 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	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 [13]	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 [117]	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
T-TR [192]	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) [148]	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 [192]	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) [148]	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 8.3: Benchmark of different models for short-time HAR on MPOSE2021 splits using OpenPose 2D skeletal representations.

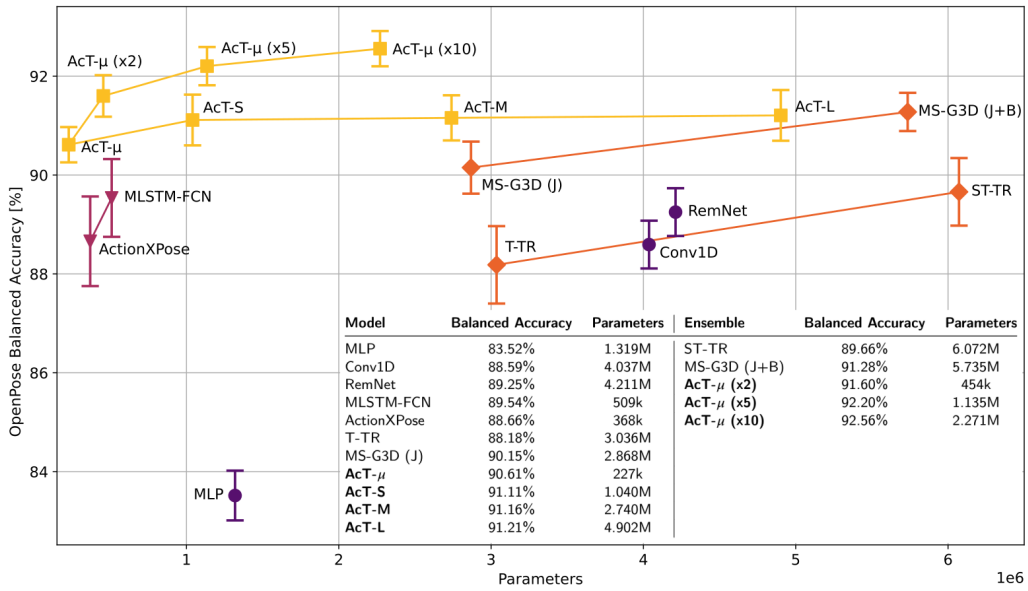


Figure 8.5: Visual representation of the benchmark of different models for short-time HAR on MPOSE2021 splits using OpenPose 2D skeletal representations. For brevity and clearness, the average balanced accuracy on the three splits of MPOSE2021 is reported. The lines connect models that use the same methodology.

MPOSE2021 PoseNet		Split 1		Split 2		Split 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	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 [13]	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 [117]	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 8.4: Benchmark of different models for short-time HAR on MPOSE2021 splits using PoseNet 2D skeletal representations.

balanced accuracy can be observed for Split3, while Split1 and Split2 present oscillations. The difference between splits reflects how much information is conveyed by training sets and how much the model can learn from that. So, it is evident that AcT scales best on Split3 because it presents complex correlations that a bigger model learns more easily. On the contrary, it seems AcT-μ is able to extract almost all the information relevant for generalization from Split2, as the accuracy only slightly increases going towards more complex models.

ST-TR exploits an ensemble of two networks modeling spatial and temporal sequence correlations, respectively. Moreover, MS-G3D leverages further information such as skeleton graph connections and the position of bones in one of the ensembled networks. Ensembles are very effective in reducing model variance and enhancing performance by exploiting independent representations learned by each network, so it is unfair to compare them with single architectures. For this reason, we create three ensemble versions of AcT-μ to have an even confront, with 2, 5, and 10 instances, respectively. To compute ensemble predictions, we average the output logits of the network instances before passing through a softmax function. The results reported at the bottom of Tab. 8.3 show that AcT-μ (x2) outperforms MS-G3D (J+B) in all the benchmarks except for balanced accuracy in Split3, despite having less than one-tenth of its parameters. Finally, the ensembles AcT-μ (x5) and AcT-μ (x10), made of 5 and 10 instances respectively, achieve even higher accuracy on all the splits with only around 1 to 2 million parameters. That proves how the balancing effect of the ensemble enhances model predictions even without feeding the network additional information.

As PoseNet data is mainly dedicated to real-time and Edge AI applications, only the models designed for this purpose have been considered in the benchmark, excluding MS-G3D, ST-TR, and AcT-L. In general, the results give similar insights. The tested models are the same as the previous case, with all the necessary modifications given by the different input formats. In the MLP case, however, performance seriously degrades as networks strongly tend to overfit input data after a small number of epochs, so the results are not included in Tab. 8.4. That is caused by the fact that PoseNet is a lighter methodology developed for Edge AI, and hence

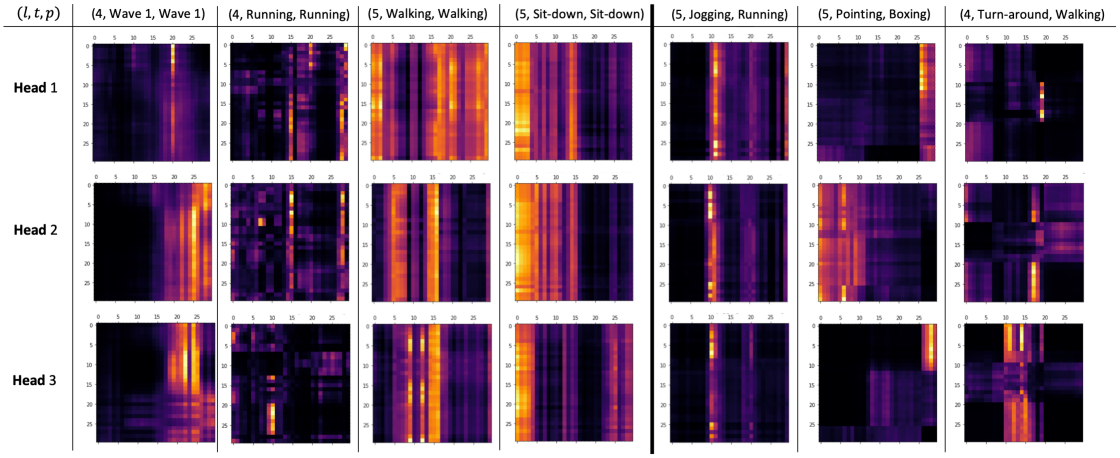


Figure 8.6: Self-attention weights of MPOSE2021 test samples. (l, t, p) represents the AcT-M l-th layer, the true label and the prediction respectively. The three rightmost columns show three attention maps of a failed prediction and the other columns are from correct classifications. It is clear from all examples how the model focuses on certain particular frames of the series in order to extract a global representation of the scene.

noisy and even missing keypoint detections are more frequent. That results in less informative data and emphasizes the difference between sequences belonging to different sub-datasets, confusing the model and inducing it to learn very specific but unusable features. The MLP is too simple and particularly prone to this kind of problem. Naturally, all the models are affected by the same problem, and the balanced accuracy on PoseNet is generally lower. The same considerations made for OpenPose apply in this case, where AcT outperforms all the other architectures and demonstrates its ability to give an accurate and robust representation of temporal correlations. Also, it is interesting to notice that Conv1D performs better than REMNet, proving to be less prone to overfitting and that standard deviations are more significant than in the OpenPose case.

8.3.3 Model introspection

In order to have an insight into the frames of the sequence the AcT model attends to, we extract the self-attention weights, i.e. the values after applying the softmax in Eq. 2.40, at different stages of the network. In Fig. 8.6, MPOSE2021 test samples are propagated through the AcT-M model, and attention weights of the three distinct heads are presented. It can be seen that the model pays attention to specific frames of the sequence when a specific gesture defines the action. On the other hand, attention is much more spread across the different frames for more distributed actions such as walking and running. Moreover, it is

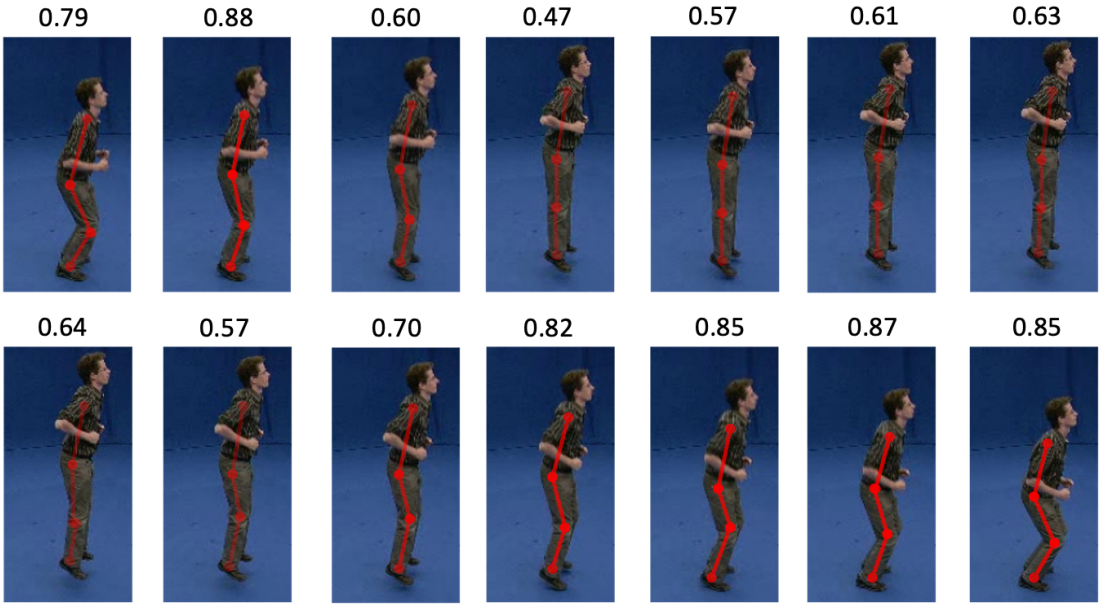


Figure 8.7: Self-attention weights of the [CLS] token with selected sequence tokens in the last encoder layer. The score is computed as the normalized sum of the different heads. Scores give a direct insight into the frames exploited by AcT to produce the classification output. The example clearly shows how bending positions are more meaningful for the network to predict the *jumping in place* action. In the image, the attention score is used as the skeleton alpha (transparency) channel.

clear how the three heads mostly focus on diverse frames of the sequence. Finally, the rightmost columns show three attention maps of failed predictions. In these last cases, attention weights are less coherent, and the model cannot extract a valid global representation of the scene.

On the other hand, Fig. 8.7 shows the last-layer self-attention scores of the [CLS] token with a set of sequence tokens. The RGB and skeleton representations of the considered sequence tokens are also shown. The scores are computed as the normalized sum of the three attention heads and give a direct insight into the frames exploited by AcT to produce the classification output. It can be seen that bent poses are much more informative for the model to predict the *jumping in place* action.

Moreover, we analyze the behavior of the network under a progressive reduction of temporal information. That can be easily done without retraining due to the intrinsic nature of AcT. In Fig. 8.8, we present how the test set balanced accuracy is affected by frame dropping. The two curves show a reduction starting from the beginning and the end of the temporal sequence, respectively. It is interesting to notice how the performance of AcT degrades with an almost linear trend. That

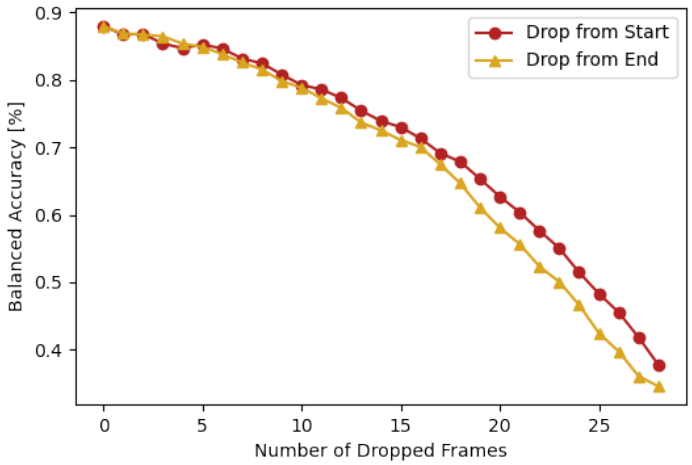


Figure 8.8: AcT-M balanced accuracy with an incremental reduction of temporal information. Due to the intrinsic nature of the network, it is possible to reduce the number of temporal steps without retraining or any kind of explicit adaptation.

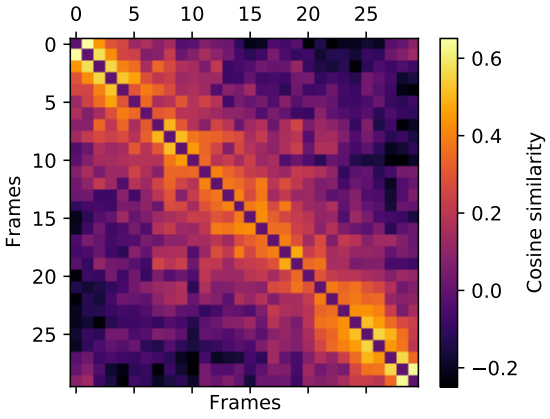


Figure 8.9: Cosine similarities of the learned T positional embeddings $\boldsymbol{X}_{\text{pos}}$ of AcT-M model.

highlights the robustness of the proposed methodology and demonstrates the possibility of adapting the model to applications with different temporal constraints.

Finally, we also study the positional embeddings $\boldsymbol{X}_{\text{pos}}$ of the AcT-M model by analyzing their cosine similarity, as shown in Fig. 8.9. Very nearby position embeddings demonstrate a high level of similarity, and distant ones are orthogonal or in the opposite direction. This pattern is constant for all T frames of the sequence,

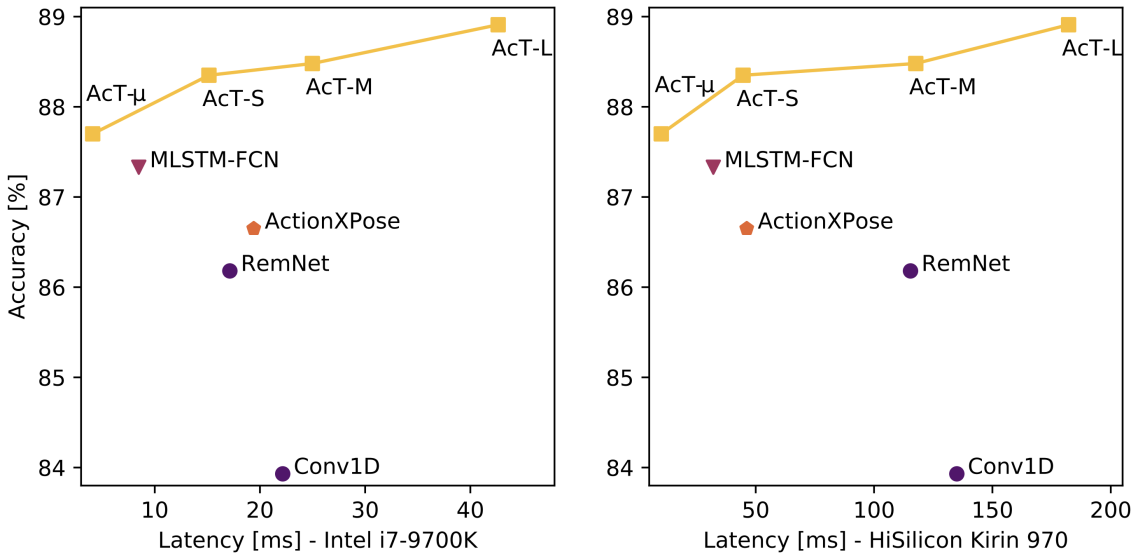


Figure 8.10: Study of the latency of different tested models on a high-performance Intel CPU and on a mobile phone equipped with an ARM-based CPU.

highlighting how actions are not particularly localized and that relative positions are essential for all the frames.

8.3.4 Real-time performance

We test the performance of all the considered models for real-time applications. To do so, we use the TFLite Benchmark[3] tool, which allows running TensorFlow Lite models on different computing systems and collecting statistical results on latency and memory usage. In our case, two CPUs are employed to measure model speed both on a PC and a mobile phone: an Intel i7-9700K for the former and the ARM-based HiSilicon Kirin 970 for the latter. In both experiments, the benchmark executes 10 warm-up runs followed by 100 consecutive forward passes, using 8 threads.

The results of both tests are reported in Fig. 8.10, where only the MLP has been ignored because, despite being the fastest-running model, its accuracy results are much lower than its competitors. The graph shows the great computational efficiency of Transformer-based architectures, whereas convolutional and recurrent networks result in heavier CPU usage. Indeed, in the Intel i7 case, REMNet achieves almost the same speed as AcT-S, but its accuracy is 2% lower. Moreover, AcT-μ is able to outperform REMNet, running at over four times its speed. MLSTM-FCN and ActionXPose, being smaller models, achieve lower latencies than the baselines:

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

the former stays between AcT-μ and AcT-S, while the latter performs similarly to AcT-S. Those results are remarkable but still outperformed by AcT-μ both on accuracy and speed.

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

Chapter 9

Generative Adversarial Super-Resolution at the Edge with Knowledge Distillation

In the last decade, DL techniques have pervaded robotic systems and applications drastically boosting automation in perception [195, 287], navigation and control [207, 267] tasks. The development of ML-driven algorithms is paving the way for advanced levels of autonomy for mobile robots, widely increasing the reliability of both UAV and UGV. In this context, the successful transmission of images acquired by the robot to the ground station often assumes a significant relevance to the task at hand, allowing the human operators to get real-time information, monitor the state of the mission, take critical planning decisions, and analyze the scenario. Moreover, unknown outdoor environments may present unexpected extreme characteristics which still hinder the release of unmanned mobile robots in the complete absence of human supervision. Indeed, complete or partial remote teleoperation remains the most reliable control strategy in uncertain scenarios, since irregular terrain and lighting conditions, as well as the loss of localization signal, can lead navigation algorithms to failure. As a direct consequence of navigation errors, the robotic platform can get stuck in critical states where human intervention is required or preferred.

However, visual data transmission for robot teleoperation, monitoring, or online data processing requires a stable continuous stream of images, 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. Besides this, UAVs and high-speed platforms require the pilot to receive the image stream at a high

framerate to follow the motion of the vehicle in non-line-of-sight situations. A straightforward but effective solution to mitigate poor bandwidth conditions and meet high-frequency transmission requirements is the reduction of 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, we propose EdgeSRGAN, a novel DL model for Single-Image Super-Resolution (SISR) at the edge to handle the problem of efficient image transmission. Single-Image Super-Resolution, also referred to as super-sampling or image restoration, aims at reconstructing a high-resolution (HR) image starting from a single low-resolution (LR) input image, trying to preserve details and the information conceived by the image. Therefore SISR, together with image denoising, is an ill-posed underdetermined inverse problem, since a multiplicity of possible solutions exist 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 SR. Such approaches rely on learning common patterns from multiple LR-HR pairs in a supervised fashion. SRCNN [55] was the first example of a CNN applied to SISR in literature, and it has been followed by multiple methods applying standard DL methodologies, such as residual learning [120, 139], dense connections [284], residual feature distillation [143], attention [283, 48, 183], self-attention, and transformers [29, 35, 138]. 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 proposed Generative Adversarial Networks (GAN) [76] for SISR to aim at reconstructing visually pleasing images. In this case, the focus is not on pixel values, but on perceptual indexes that try to reflect how humans perceive image quality. This is usually implemented using perceptual losses together with adversarial training and is referred to as visual-based SR. SRGAN [136] first proposed adversarial training and was later followed by other works [139, 68, 255]. Having in mind robotic image transmission as a target application, in this work we particularly focus on visual-based SR, with the aim of reconstructing visually pleasing images to be used by human operators for real-time teleoperation and monitoring.

Our intuition relies on the fact that a lightweight neural network allows to send low resolution images at a high transmission rate also with scarce bandwidth, and then reconstruct the high resolution image on the mobile device of the pilot. We propose an Edge-AI computationally efficient SR neural network to provide fast inference on CPUs and Edge TPU devices. To this aim, we adopt several optimization steps to boost the performance of our model while minimizing the quality drop. We refine the architecture of the original SRGAN [136] in order to speed up inference and perform model quantization. Nonetheless, we experiment with a teacher-student knowledge distillation technique for SISR to further enhance the

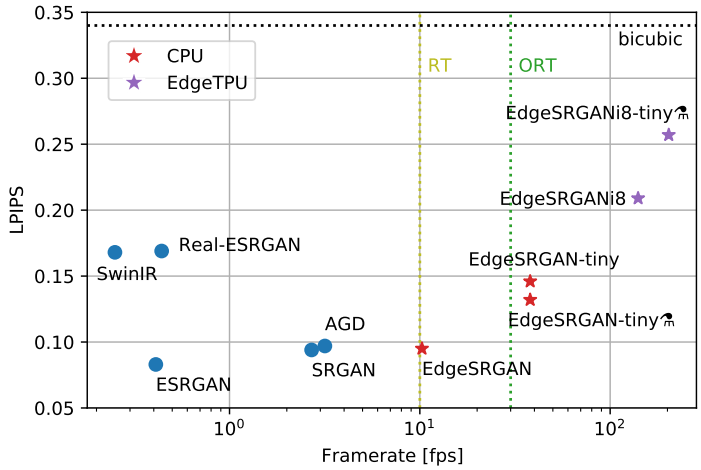


Figure 9.1: LPIPS [281] results (lower is better) on Set5 [25] vs framerate (80×60 input) of different visual-oriented SISR methods for $\times 4$ upsampling. Real-time (RT) and over-real-time (ORT) framerates are marked as references. Our models, marked with $\star$, reach real-time performance with a competitive perceptual similarity index on CPU. Edge TPU models can further increase inference speed far beyond real-time, still outperforming the bicubic baseline.

reconstructed image of our tiny model. We take inspiration from the work of [89] and obtain an improvement on all the considered metrics.

We perform experiments to validate the proposed methodology under multiple perspectives: numerical and qualitative analysis of the predicted images and inference efficiency on both CPU and Edge TPU devices. As an example, as shown in Fig. 9.1, EdgeSRGAN achieves real-time performance with a competitive perceptual similarity index LPIPS [281] when compared with other visual-oriented SISR methods. Moreover, a test of the performance of the system from a robotic perspective is performed. In particular, we focus on image transmission for teleoperation in case of bandwidth degradation, also performing tests with the popular robotic middleware ROS2. All the code related to this work is open source and publicly available [1].

9.1 Methodology

We choose to use an adversarial approach to obtain an optimal balance between pixel-wise fidelity and perceptual quality. For this reason, we take inspiration from

[1] https://github.com/PIC4SeR/EdgeSRGAN

three of the most popular GAN-based solutions for SISR: SRGAN [136], ESRGAN [257], and AGD [68]. The aim of the proposed method is to obtain a real-time SISR model (EdgeSRGAN) with minimal performance drop compared to state-of-the-art (SOTA) solutions. For this reason, we mix successful literature practices with computationally-efficient elements to obtain a lightweight architecture. Then, we design the network training procedure to leverage a combination of pixel-wise loss, perceptual loss, and adversarial loss. To further optimize the inference time, we apply knowledge distillation to transfer the performance of EdgeSRGAN to an even smaller model (EdgeSRGAN-tiny). Furthermore, we study the effect of quantization on network latency and accuracy. Finally, we propose an additional inference-time network interpolation feature to allow for real-time balancing between pixel-wise precision and photo-realistic textures.

9.1.1 Network architecture

As in [257], we modify the original design of SRGAN in both the architecture and the training procedure. However, in our case, the modifications seek efficiency as well as performance. To obtain a lighter architecture we reduce the depth of the model by using only $N = 8$ Residual Blocks instead of the original 16. In particular, we use simple residuals instead of the Residual-in-Residual Dense Blocks proposed by [257] as they are less computationally demanding. For the same reason, we change PReLU activation functions into basic ReLU. We also remove Batch Norm to allow the model for better convergence without generating artifacts [257]. Finally, we use Transpose Convolution for the upsampling head instead of Sub-pixel Convolution [223]. Despite its popularity and effectiveness, Sub-pixel Convolution is computationally demanding due to the Pixel Shuffling operation, which rearranges feature channels spatially. We choose instead to trade some performance for efficiency and apply Transpose Convolutions taking precautions to avoid problems such as checkerboard artifacts [185]. The complete EdgeSRGAN architecture is shown in Fig. 9.2. The adopted discriminator model is the same used in [136, 257], as it serves only training purposes and is not needed at inference time. Its architecture is shown in Fig. 9.3.

9.1.2 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 content loss. In this way, we help the generator to avoid local minima and generate visually pleasing results in the subsequent adversarial training. We use the mean absolute error (MAE) loss for the optimization as it has been proven to bring better convergence than mean squared error (MSE) [285, 139, 283, 257]:

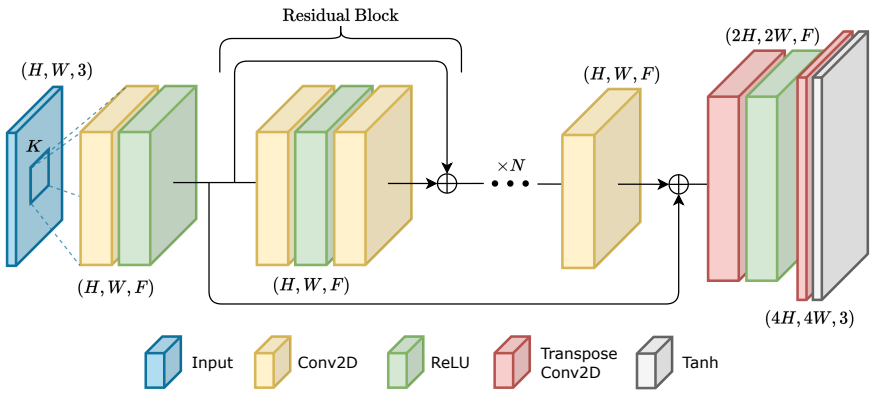


Figure 9.2: EdgeSRGAN Generator Architecture. We adopt N Residual Blocks without Batch Norm as basic blocks of the network. All the convolutional layers has kernel dimension K, single stride and zero padding to preserve the spatial dimensionality. Upsampling is performed with two $\times 2$ transpose convolutions.

$$\mathcal{L}_{\text{cont}} = \|\boldsymbol{y}_{\text{HR}} - \boldsymbol{y}_{\text{SR}}\|_1 \tag{9.1}$$

where $\boldsymbol{y}_{\text{HR}}$ is the ground-truth high resolution (HR) image and $\boldsymbol{y}_{\text{SR}}$ is the super-resolved (SR) image. We use the Peak Signal To Noise Ratio (PSNR) metric to validate the model in the first phase.

In the second phase, the resulting model is fine-tuned in an adversarial fashion, optimizing a loss that takes into account an adversarial loss and a perceptual loss. As presented in [136], the generator loss can be formulated as

$$\mathcal{L}_{\text{gen}} = \mathcal{L}_{\text{perc}} + \xi\, \mathcal{L}_{\text{adv}} + \eta\, \mathcal{L}_{\text{cont}}. \tag{9.2}$$

$\mathcal{L}_{\text{perc}}$ is the perceptual loss proposed by [136] and defined as the MSE between the features representations $\varphi(\cdot)$ of $\boldsymbol{y}_{\text{HR}}$ and $\boldsymbol{y}_{\text{SR}}$. The features are extracted using VGG19 [228] pre-trained on ImageNet:

$$\mathcal{L}_{\text{perc}} = \|\varphi(\boldsymbol{y}_{\text{HR}}) - \varphi(\boldsymbol{y}_{\text{SR}})\|_2 \tag{9.3}$$

On the other hand, $\mathcal{L}_{\text{adv}}$ is the generator adversarial loss, defined as:

$$\mathcal{L}_{\text{adv}} = -\log(D(\boldsymbol{y}_{\text{SR}})) \tag{9.4}$$

where D is the discriminator. Using this loss, the generator tries to fool the discriminator by generating images that are indistinguishable from the real HR ones. ξ and η are used to balance the weight of different loss components.

The weights of the discriminator D are optimized using a symmetrical adversarial loss, which tends to correctly discriminate HR and SR images:

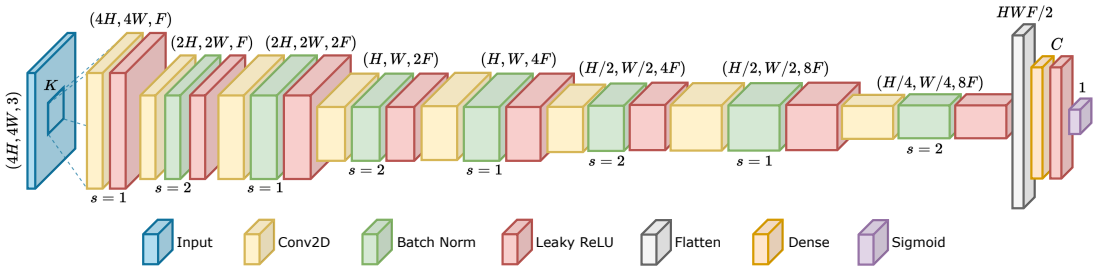


Figure 9.3: EdgeSRGAN Discriminator Architecture. Spatial dimensionality is reduced with the double-stride convolutional blocks. At the same time, the feature size is doubled in each single-stride convolutional block.

$$\mathcal{L}_{\text{discr}} = \log(D(\boldsymbol{y}_{\text{SR}})) - \log(D(\boldsymbol{y}_{\text{HR}})) \tag{9.5}$$

We optimize both models at the same time, without alternating weight updates like done in most seminal works on GANs.

9.1.3 Knowledge distillation

As mentioned in Section 2.4, knowledge distillation (KD) has gained increasing interest in DL for its ability to efficiently transfer knowledge from bigger models to simpler ones. 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 [89, 282]. However, to the best of our knowledge, KD has never been applied to GAN SISR models. For this reason, we adapt an existing technique developed for SISR called Feature Affinity-based Knowledge Distillation (FAKD) [89], 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 to tackle the fact that regression problems generate unbounded solution spaces. Indeed, most of the KD methods so far have only tackled classification tasks.

Given a layer l of the network, a generic feature map extracted from that layer (after the activation function) has shape $C \times W \times H$, where C is the number of channels, W and H are the width and the height of the tensor. We first flatten the tensor along the last two components obtaining the three-dimensional feature map $\boldsymbol{F}_l$ with shape $C \times (W \cdot H)$, which now holds all the spatial information along a single axis. We define the affinity matrix $\boldsymbol{A}_l$ as the product:

$$\boldsymbol{A}_l = \tilde{\boldsymbol{F}}_l^{\,T} \tilde{\boldsymbol{F}}_l \tag{9.6}$$

where $\tilde{\boldsymbol{F}}_l$ is the normalized feature map obtained as:

$$\tilde{\boldsymbol{F}}_l = \frac{\boldsymbol{F}_l}{\|\boldsymbol{F}_l\|_2} \tag{9.7}$$

Differently from [89] the norm is calculated for the whole tensor and not only along the channel axis. Moreover, we find better convergence using the Euclidean norm instead of its square. In this way, the affinity matrix $\boldsymbol{A}_l$ has shape $(W \cdot H) \times (W \cdot H)$ and the total distillation loss is computed as:

$$\mathcal{L}_{\text{dist}} = \lambda \|\boldsymbol{y}_{\text{SR}}^{\text{teach}} - \boldsymbol{y}_{\text{SR}}^{\text{stud}}\|_1 + \frac{1}{L}\sum_{l=1}^{L} \|\boldsymbol{A}_l^{\text{teach}} - \boldsymbol{A}_l^{\text{stud}}\|_1 \tag{9.8}$$

where L is the number of distilled layers. Differently from [89], we sum the loss along all the tensor dimensions and average the result obtained for different layers. These modifications experimentally lead to better training convergence. We also add another loss component, weighted by λ, which optimizes the student model to generate outputs close to teacher ones. In our experimentation, the distillation loss is simply added to the overall generator loss weighted by the parameter γ.

9.1.4 Model quantization

To make EdgeSRGAN achieve even lower inference latency, we optimize the model to reduce the computational effort at the cost of a loss in performance. In particular, we apply full-integer quantization (see Section 2.4) using the TFLite library[2]. This strategy drastically reduces memory and computational demands due to the high efficiency of integer computations on microcontrollers. For our experimentation, we deploy the quantized model on a Coral USB Accelerator, targeting its Edge TPU coprocessor (see Tab. 2.1 for details on the hardware device).

9.1.5 Model interpolation

Following the procedure proposed in [257], we adopt a flexible and effective strategy to obtain a tunable trade-off between a content-oriented model and a GAN-trained one. This feature can be very useful for real-time applications, as it allows the SISR network to promptly adapt to the needs of the user. Indeed, some real scenarios may need better perceptual quality, for example when the remote control of a robot has to be performed by a human pilot. 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, we linearly interpolate the generator parameters layer-by-layer, according to the following formula:

[2] https://tensorflow.org/lite

$$\theta_{\text{gen}} = \alpha\,\theta_{\text{gen}}^{\text{content}} + (1-\alpha)\theta_{\text{gen}}^{\text{GAN}} \tag{9.9}$$

where $\theta_{\text{gen}}^{\text{content}}$, and $\theta_{\text{gen}}^{\text{GAN}}$ are the parameters of the content-based model and the GAN fine-tuned model, respectively, and $\alpha \in [0,1]$ is the interpolation weight. We avoid the alternative technique consisting of directly interpolating network outputs: applying this method in real-time would require running two models at the same time. Moreover, Wang et al. [257] report that this approach does not guarantee an optimal trade-off between noise and blur.

9.2 Experiments

9.2.1 Experimental setting

In this section, we define the implementation details of our method as well as the procedure we followed to train and validate the model. As previously done by most SISR works, we train the network on the high-quality DIV2K dataset [7] with a scaling factor of 4. The dataset contains 1000 images with 2K resolution (each image has 2040 pixels for either the horizontal or the vertical dimension), subdivided in 800 training samples, 100 validation samples and 100 testing samples. The dataset provides HR images at native resolution (for training and validation sets only), as well as the corresponding downgraded images with different scaling factors (x2, x3, x4, x8). Since the testing samples do not have the target HR samples, we adopt the validation set for testing. We train our model with input patches of size 24x24 pixels (96x96 in the HR space), selected randomly from the training set images. We apply data augmentation by randomly flipping or rotating the images by multiples of 90°. We adopt a batch size of 16.

For the standard EdgeSRGAN implementation, we choose $N = 8$ Residual Blocks, $F = 64$ filters in the convolutional layer, and discriminator head channels $C = 1024$, obtaining a generator with around 660k parameters and a discriminator of over 23M (due to the fully-connected head). All the convolutional layer has kernel size $K = 3$ and the discriminator Leaky ReLU parameter is set to 0.2. We first train EdgeSRGAN with the content loss for 5×10^5 steps with Adam optimizer and a constant learning rate of 1×10^{-4}. Then, the model is fine-tuned in the adversarial setting described in Section 9.1 for 1×10^5 steps. Adam optimizer is used for both the generator and the discriminator with a learning rate of 1×10^{-5} which is further divided by 10 after 5×10^4 steps. For the loss function, we set $\xi = 1 \times 10^{-3}$ and $\eta = 0$.

To obtain an even smaller model for our distillation experiments, we build EdgeSRGAN-tiny by setting $N = 4$, $F = 32$, and $D = 256$. We further shrink the size of the discriminator by removing all the single-stride blocks (see Fig. 9.3). In this configuration, we also remove the Batch Norm layer from the first double-stride block to be coherent with the larger version. The obtained generator has

Method	Scale	Params	Framerate (80 × 60) [fps]		Framerate (160 × 120) [fps]	
			CPU	EdgeTPU	CPU	EdgeTPU
SwinIR [138]	×4	11.9M	0.25 ± 0.01	-	0.06 ± 0.01	-
ESRGAN [257]		16.7M	0.40 ± 0.01	-	0.10 ± 0.01	-
Real-ESRGAN [255]		16.7M	0.44 ± 0.01	-	0.11 ± 0.01	-
SRGAN [136]		1.5M	2.70 ± 0.08	-	0.95 ± 0.02	-
AGD [68]		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		0.10M	37.99 ± 1.42	203.16 ± 3.03	11.76 ± 0.20	20.57 ± 0.05
SwinIR [138]	×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 9.1: Framerate comparison of different methods for ×4 and ×8 upsampling, with two different input resolutions (80×60 and 160×120). The results are provided as mean and standard deviation of 10 independent experiments of 100 predictions each. Current content-oriented SISR state-of-art method SwinIR [138] is reported as a reference. Real-time and over-real-time framerates are in blue and red colors, respectively. The proposed solution is the only one compatible with EdgeTPU devices and allows reaching real-time performance in both conditions.

around 96K parameters and the discriminator around 2.75M. The pre-training procedure is the same as 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. 9.8. EdgeSRGAN is used as the teacher model, distilling its layers 2, 5, and 8 into EdgeSRGAN-tiny 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, we set $\xi = 1 \times 10^{-3}$ and $\eta = 0$.

Finally, we create a third version of our model to upscale images with a factor of 8, using the corresponding x8 DIV2K samples, with input patches of 12x12 pixels. To do so, we change the first transpose convolution layer of EdgeSRGAN and EdgeSRGAN-tiny to have a stride of 4 instead of 2 and leave the rest of the architecture unchanged. The resulting generators have 710K and 108K parameters, respectively. The discriminator remains unchanged as per the x4 upscaling case. The training procedure for these models is analogous to the ones used for the x4 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 found by running a random search and choosing the best-performing models on DIV2K validation. We use PSNR to validate the models during content-based loss optimization and LPIPS [281] (with AlexNet backbone [125]) during GAN training. To perform all the training experiments we employ the TensorFlow 2 framework [1] and a workstation with 64 GB of RAM and an Nvidia 3090 RTX GPU.

9.2.2 Real-time performance

Since the main focus of the proposed methodology is to train an optimized SISR model to be efficiently run at the edge in real-time, we first report an inference speed comparison between the proposed method and other literature methodologies. All the results are shown in Tab. 9.1 as mean and standard deviation of 10 independent experiments of 100 predictions each. We compare the proposed methodology with other GAN-based methods [136, 257, 255, 68] and with the current state-of-the-art in content-oriented SISR SwinIR [138]. Since the original implementations of the GAN-based solutions consider $\times 4$ upsampling only, for the $\times 8$ comparison we just report SwinIR. We 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 by the fact that (640×480) is a standard resolution provided as a native video stream for most commercial cameras. We also report the number of parameters for all the models.

For all the considered methods we measure the CPU timings with the model format of the original implementation (PyTorch or TensorFlow) on a MacBook Pro with an Intel i5-8257U CPU. The concept of real-time performance strongly depends on the downstream task. For robotic monitoring and teleoperation, we consider 10 fps as the minimum real-time framerate, considering as over-real-time everything above 30 fps, which is the standard framerate for most commercial cameras. The proposed methodology clearly outperforms all the other methods in terms of inference speed and achieves real-time performance on CPU in almost all the testing conditions. It is worth noting that AGD is specifically designed to reduce latency for GAN-based SR and has fewer parameters than EdgeSRGAN, but still fails at achieving real-time without a GPU.

In addition, we report the framerate 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 underlined how the $\times 8$ models with (160×120) input resolution cannot target the EdgeTPU device due to memory limitations.

	Set5 [25]			Set14 [280]			BSD100 [166]			Manga109 [169]			Urban100 [102]		
Method	**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 [138]	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 [136]	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 [257]†	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 [68]	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 9.2: Quantitative comparison of different methods for content-oriented ×4 upsampling. Current SISR state-of-art method SwinIR [138] and bicubic baseline are reported as reference.
↑: higher is better, ↓: lower is better, †: trained on DIV2K [7] + Flickr2K [239] + OST [256]

	Set5 [25]			Set14 [280]			BSD100 [166]			Manga109 [169]			Urban100 [102]		
Model	**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 [138]	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 [136]	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[257]†	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 [255]†	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 [68]	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 9.3: Quantitative comparison of different methods for visual-oriented ×4 upsampling. Current SISR SOTA method SwinIR [138] and bicubic baseline are reported as references.
↑: higher is better, ↓: lower is better, †: trained on DIV2K [7] + Flickr2K [239] + OST [256]

		Set5 [25]			Set14 [280]			BSD100 [166]			Manga109 [169]			Urban100 [102]		
Model		**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 [138]		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	content	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	content	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	visual	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	visual	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 9.4: Quantitative performance of the proposed method for ×8 upsampling. Current SISR SOTA method SwinIR [138] and bicubic are reported as references. ↑: higher is better, ↓: lower is better

		Set5 [25]			Set14 [280]			BSD100 [166]			Manga109 [169]			Urban100 [102]		
Model	**Scale**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**	**PSNR ↑**	**SSIM ↑**	**LPIPS ↓**
EdgeSRGANi8	×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	×4	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	×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	×8	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 9.5: Quantitative performance of the full-integer quantized models for ×4 and ×8 visual-based SR. ↑: higher is better, ↓: lower is better

9.2.3 Super-Resolution results

To present quantitative results on image Super-Resolution we refer to content-oriented SR for models trained with content-based loss only and to visual-oriented SR for models trained with adversarial and perceptual losses. Content-based loss (mean absolute error or mean squared error) aims at maximizing Peak Signal To Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM) [259], while adversarial and perceptual losses aim at maximizing visual quality. We test EdgeSRGAN models on five benchmark datasets (Set5 [25], Set14 [280], BSD100 [166], Manga109 [169], and Urban100 [102]) measuring PSNR, SSIM, and LPIPS. We follow the standard procedure for SISR adopted in [138], where the metrics are computed on the luminance channel Y of the YCbCr converted images. Also, S pixels are cropped from each border of the image, where S is the model scale factor.

Tab. 9.2 and Tab. 9.3 show the comparison with other methods for content-oriented and visual-oriented $\times 4$ SR, respectively. We report results of other GAN-based methodologies [136, 257, 255, 68] as well as the current content-oriented SOTA SwinIR [138] and bicubic baseline, as reference. Differently from what is usually found in the literature, we refer to the OpenCV bicubic resize implementation instead of the MATLAB one. For visual-oriented SR, we also report the results of the distilled tiny model EdgeSRGAN-tiny⚗. The proposed method reaches competitive results in all the metrics, even with some degradation for tiny models due to the considerable parameter reduction. The distillation method clearly helps EdgeSRGAN-tiny training by transferring knowledge from the standard model and decreasing the degradation due to the reduced number of parameters. Note that ESRGAN and RealESRGAN are trained on Flickr2K [239] and OST [256] datasets in addition to DIV2K. Tab. 9.4 reports results of the $\times 8$ models, together with SwinIR and bicubic. Also in this case, the proposed models reach competitive results, and KD helps in reducing performance degradation in the tiny model. As a final qualitative evaluation, Fig. 9.4 compares the super-resolved images obtained by EdgeSRGAN with the considered state-of-the-art solutions. Our model shows comparable results, highlighting more texture and details than networks trained with pixel loss while remaining true to the ground truth image.

Network interpolation

We report the results of network interpolation on the benchmark datasets in Fig. 9.5. We consider α values between 0 and 1 with a step of 0.1, with 0 meaning a full visual-oriented model and 1 a full content-oriented model. All results refer to the standard EdgeSRGAN model for $\times 4$ upsampling. This procedure effectively shows how it is possible to choose the desired trade-off between content-oriented and visual-oriented SR simply by changing 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 considered 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 in any way the inference speed. For an additional visual evaluation, Fig. 9.6 reports the outputs obtained for increasing values of α on a Set5 dataset sample.

Network optimization

To target Edge TPU devices and reach over-real-time inference results, we follow the quantization scheme of Eq. 2.42 for both weights and activations to obtain a full-integer model. Since quantized models must have a fixed input shape, we generate a full-integer network for each input shape of the testing samples. We use the 100 images from the DIV2K validation set as a representative dataset to calibrate the quantization algorithm. We refer to the int8-quantized standard model as EdgeSRGANi8. As for the tiny model, we optimize the distilled network EdgeSRGANi8-tiny⚗. Results for the visual-oriented optimized models are shown in Tab. 9.5. Due to the reduced activation and weight space of the full-integer models, we experience a great increase in inference speed up to over-real-time, at the cost of degradation in SR performance. All the proposed quantized models still outperform bicubic baseline on the perceptual index LPIPS and therefore represent a good option for applications in which really fast inference is needed. A comparison of the LPIPS performance vs framerate of different models for visual-oriented $\times 4$ upsampling is shown in Fig. 9.1.

9.2.4 Application to image transmission in mobile robotics

Our real-time SISR methodology can provide competitive advantages in a wide variety of practical engineering applications. In this section, we target a specific use case of mobile robotics, proposing our EdgeSRGAN model as an efficient DL-based solution for real-time image transmission. Indeed, robot remote control in unknown terrains needs a reliable transmission of visual data at a satisfying framerate, 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. In the last years, the robotics community has focused on the development of globally shared solutions for robot software and architectures, also handling data communications between multiple platforms and devices.

ROS2 [156] is the standard operative system for robotic platforms. It is a middleware based on a Data Distribution System (DDS) protocol where application nodes communicate with each other through a topic with a publisher/subscriber mechanism. However, despite the most recent attempts to improve the reliability and efficiency of message and data packet communications between different

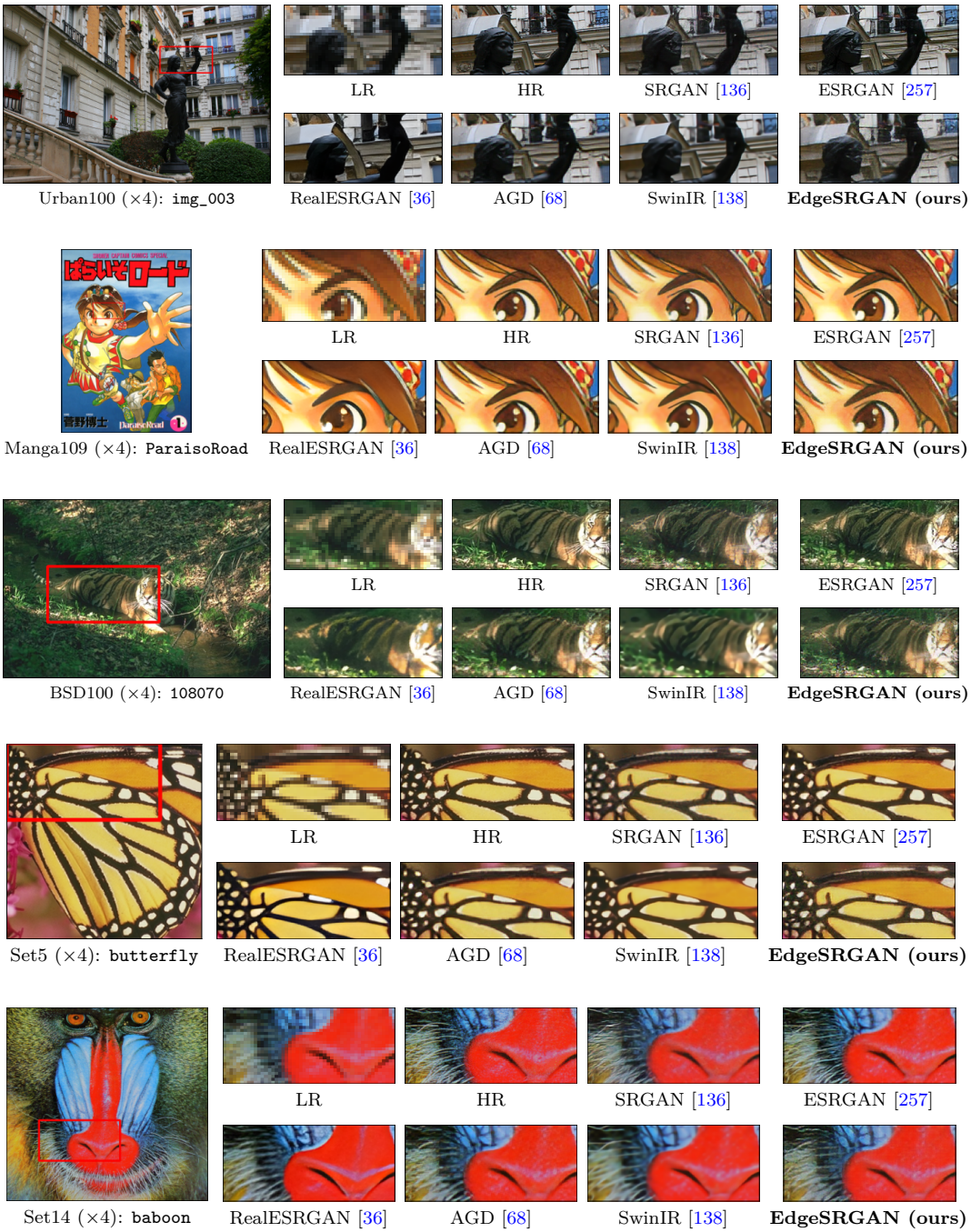


Figure 9.4: Visual comparison of bicubic image SR (×4) methods on random samples from the considered datasets. EdgeSRGAN achieves results that are comparable to state-of-the-art solutions with $\sim 10\%$ of the weights.

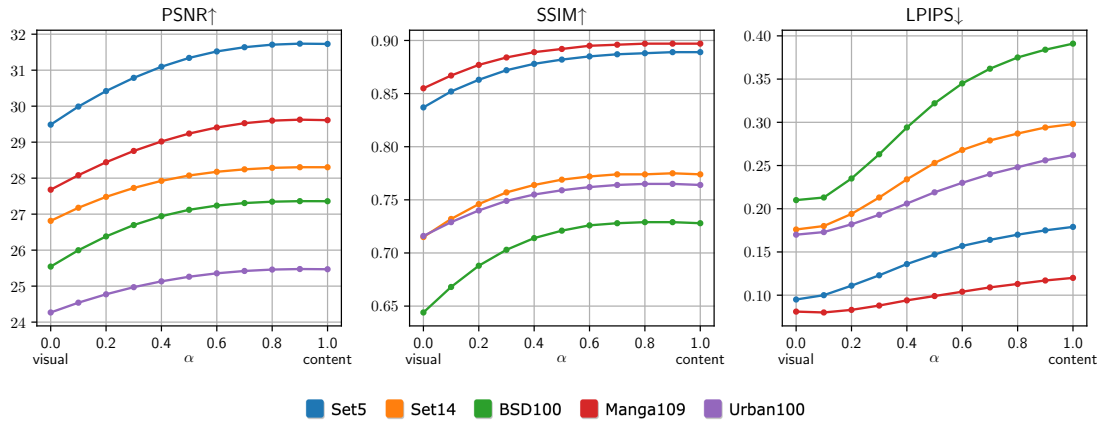


Figure 9.5: EdgeSRGAN network interpolation results on the benchmark datasets for ×4 upsampling. Changing the network interpolation weight α, it is possible to select the desired trade-off between content-oriented and visual-oriented SR.
↑: higher is better, ↓: lower is better

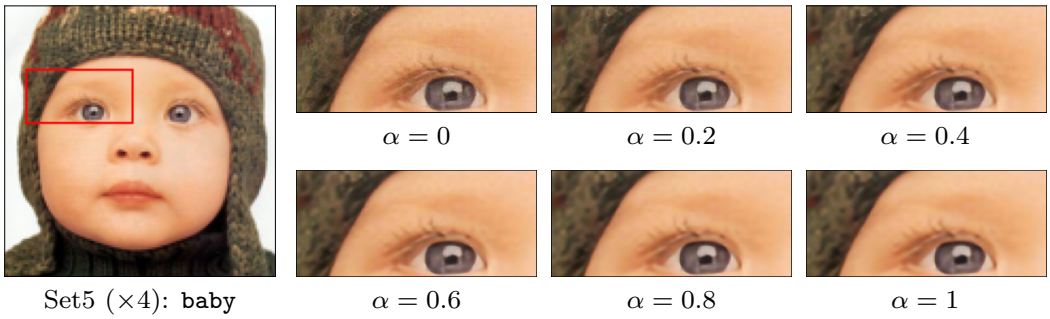


Figure 9.6: Visual comparison of interpolated EdgeSRGAN for different values of α. Values closer to $\alpha = 1$ generate outputs focused on content fidelity, while small values go towards visually pleasing results.

nodes and platforms, heavier data transmission such as image streaming is not yet optimized and reliable.

The typical practical setting used for robot teleoperation and exploration in unknown environments is composed of a ground station and a rover connected to the same wireless network. As shown in Fig. 9.7, we 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 the experiment, we use both an Intel RealSense D435i camera and a Logitech C920 webcam mounted on a Clearpath Jackal robot, together with a Microhard BulletPlus router for image transmission. The available image resolutions

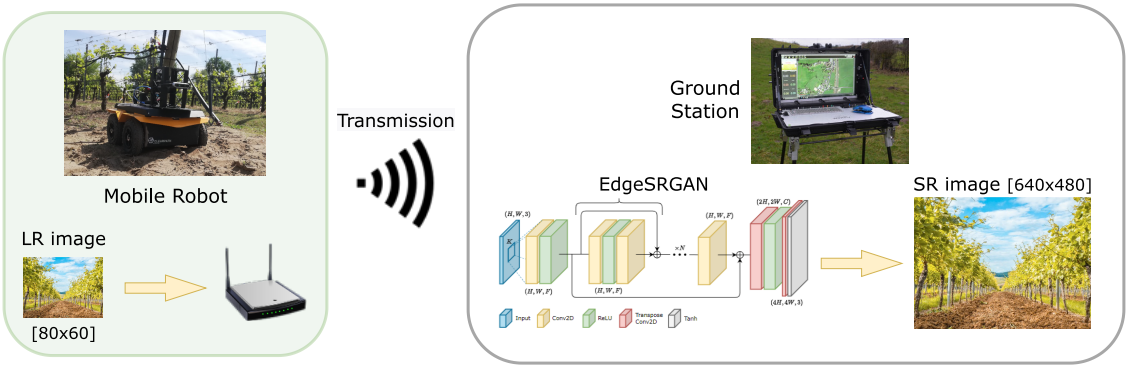


Figure 9.7: Efficient image transmission system with EdgeSRGAN for mobile robotic applications in outdoor environments.

with RealSense cameras, which are the standard RGB-D 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 strong bandwidth limitations, transmission delays, or partial loss of packets, the maximum resolution and framerate allowed by ROS2 communication are extremely low: we 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 framerate and resolution hinders the high-speed motion of a robotic platform in 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.

The adoption of our real-time SR system ensures the timely arrival of both RGB and depth images via ROS2. Thanks to the fast-inference performance of EdgeSRGAN, we can stream low resolution images (80×60) at a high framerate (30 fps) and receive a high resolution output: (320×240) with a x4 image upsampling and (640×480) with a x8 upsampling, showing a clear improvement on standard performance. Our 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: high resolution images can be directly exploited by the human operator for remote control. Moreover, they can be used to feed computationally hungry algorithms like sensorimotor agents, visual-odometry, or visual-SLAM which we may prefer to run on the ground station to save the constrained power resources of the robot and significantly boost the autonomy level of the mission.

We also test video transmission performance in a more general framework to reproduce all the potential bandwidth conditions. We use the well-known video

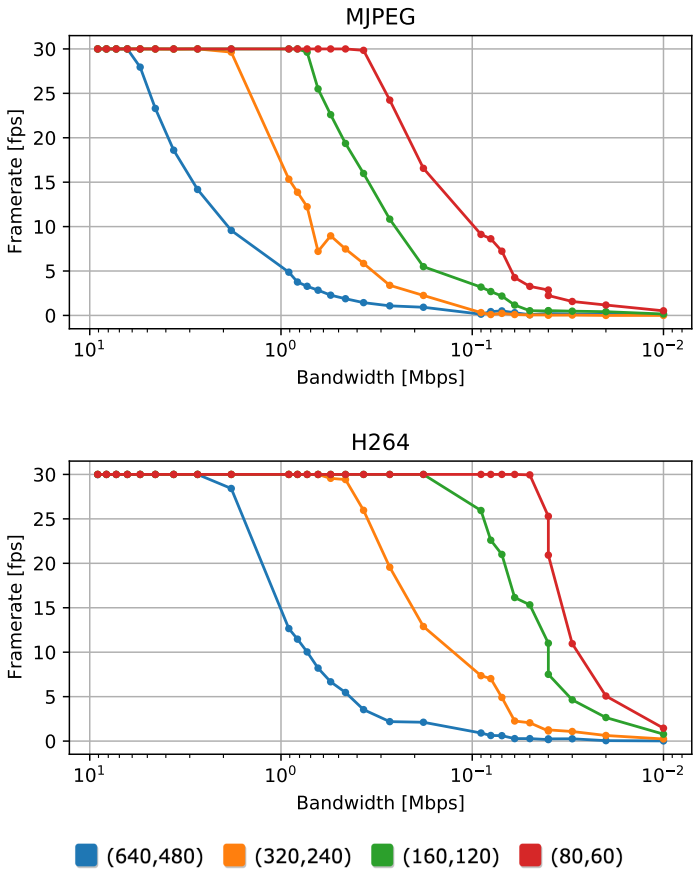


Figure 9.8: Framerate results vs bandwidth for video transmission at different input resolutions with MJPEG and H264 compression. Bandwidth is in log scale.

streaming library GStreamer[3] to transmit video samples changing the available bandwidth. We progressively reduce the bandwidth from 10 Mbps to 10 kbps using the Wondershaper library[4] and measure the framerate at the receiver side. We use 10 seconds of the standard video sample *smtpe* natively provided by GStreamer *videotestsrc* video source at 30 fps, and we encode it for transmission using MJPEG and H264 video compression standards. The encoding is performed offline in order to be sure that all the available resources are reserved for transmission only. Indeed, most cameras provide hardware-encoded video sources, without the need for software compression. To be consistent with the other experiments, we keep using (640×480) and (320×240) as high resolutions and (160×120) and (80×60) as low resolutions. Each experiment is performed 10 times to check the consistency

[3] https://gstreamer.freedesktop.org/

[4] https://github.com/magnific0/wondershaper

in results. Fig. 9.8 presents the average framerate achieved with different bandwidths. Streaming the video source directly without any middleware such as ROS2 ensures a higher transmission performance. However, as expected, streaming high resolution images is not possible in the case of low bandwidth, and the framerate quickly drops to very low values, resulting unsuitable for real-time applications. On the other hand, lower resolutions can be streamed with minimal frame drop even with lower available bandwidths. H264 compression shows the same behavior as MJPEG, but shifted to lower bandwidths. Indeed, H264 is more sophisticated and efficient, as it uses temporal frame correlation in addition to spatial compression. In a practical application with a certain bandwidth constraint, a proper combination of a low resolution video source and an SR model can be selected to meet the desired framerate requirements on the available platform (CPU or Edge TPU). This kind of mechanism can also be dynamically and automatically activated and deactivated depending on the current connectivity, in order to avoid framerate drops and ensure a smooth image transmission.

Chapter 10

Conclusions

In recent years, Artificial Intelligence has been the focus of technology research interest for both academias and industries. The latest developments of data-driven algorithms, in particular in the field of Deep Learning, have reached milestones in terms of accuracy and efficiency. Every month, tenth or even hundreds of new methodologies are proposed in the literature, and more and more DL algorithms are released and available for usage by people in a high number of products. For example, the recent release of the ChatGPT chatbot has come to the broad public interest and raised a lot of debate on both ethical and technological perspectives of AI [53]. In parallel, with the development of novel paradigms such as Industry and Agriculture 4.0, also robotics and automation are acquiring more and more key roles in the solution of a lot of industrial and technological problems. In particular, service robotics products can provide assistance to humans in order to ease a number of laborious and repetitive tasks. Fields such as smart agriculture and indoor assistance can greatly benefit from the development of autonomous agents able to sense the environment, collect data, identify interest points, navigate, and solve specific tasks. To achieve this goal, tight integration between AI and robotics must be reached, with an interest in both algorithmic accuracy and efficiency. Research in this field of application should always carefully consider the algorithm deployment, and, in particular for onboard solutions, the right tradeoff between accuracy and inference time should be reached. In this context, optimization techniques such as graph pruning or quantization can drastically increase real-time performance with limited loss in accuracy.

In this dissertation, several techniques have been presented that apply DL-based methodologies to solve service robotics-related tasks. Part II focused on the development of an algorithmic pipeline for robotic navigation in an agricultural context, featuring DL methods in different steps, such as visual pre-processing, global path planning, and local navigation. This approach represents a good example of how data-driven methods can interact to tackle several tasks at different levels of the

pipeline stack. Moreover, it shows an application of both offline/cloud and online/onboard execution of such algorithms, showing how the inference efficiency relevance greatly depends on the final deployment context. On the other hand, Part III focused more on various methodologies related to robotic perception. All these methods fall in the field of processing sensing data coming from the different sensors mounted onboard the robotic platform. The ability to interpret and extract knowledge from visual and ranging signals is fundamental to increase the environment understanding and enable autonomous agents to take better decisions and actions. In particular, in this dissertation, we analyze tasks ranging from object detection (Chapter 6) and scene understanding (Chapter 8), that are more on the high application level, to UWB signals processing (Chapter 7) and efficient visual data stream for remote teleoperation (Chapter 9), that instead are more on the low signal level. In all these contexts, great care has been devoted to real-time efficiency and onboard execution.

10.1 Future works

The methodologies presented in this dissertation are only a first attempt to solve some fundamental tasks in the DL for service robotics landscape and, as so, they can be used as the basis for future works. The algorithmic pipeline for robotic autonomous navigation presented in Part II can be further enriched by developing a tight integration between a satellite-based and a UAV-based mapping method to obtain a higher precision occupancy grid for the global path planning process. Moreover, further real-world experiments can be performed in order to test the differences between the Reinforcement Learning local navigation, and the visual-based one. A specific RL-based methodology can also be used to specialize the agent for autonomously performing the inter-row segments navigation, currently tackled with classic GPS-based localization. This kind of optimization could allow to completely avoid the usage of the GNSS system for local navigation, using it for redundancy trajectory checking only. That could help to improve the navigation quality and reduce the costs since satellite signals can be insufficiently reliable in agricultural environments without adopting expensive RTK-enabled GNSS receivers. Part III methodologies can be further developed by integrating them into more complex applications. As an example, the fruit detection system presented in Chapter 6 can be used as the basis for a counting system that can perform automatic yield estimation if integrated on a robotic platform able to move in orchards autonomously, using the navigation pipeline of Part II. Moreover, the Human Action Recognition methodology presented in Chapter 8 can be integrated into a human-aware robotic navigation system that plans and acts according to the recognized people's behavior. Another important future work will be to study and apply more runtime optimization methodologies to further reduce the inference footprint on the robotic platform, while keeping algorithms accuracy high. An increase in efficiency can also

be beneficial for parallelizing tasks and allowing more algorithms to run simultaneously on the same platform. A fundamental field to be investigated is the trade-off between the algorithm accuracy, its weight expressed in number of parameters, and its inference efficiency. Also, further investigations in methods such as Knowledge Distillation and Domain Generalization can improve knowledge transfer between different models and environments. All these aspects represent promising topics for future research starting from the methodologies presented in this work. Therefore, this dissertation constitutes a step toward the full integration between robotics and AI and paves the way for future research in this direction.

Acronyms

AcT Action Transformer.

Adam Adaptive moment estimation.

AI Artificial Intelligence.

ANN Artificial Neural Network.

AP Average Precision.

BN BatchNorm, Batch Normalization.

CIR Channel impulse response.

CNN Convolutional Neural Network.

CV Computer Vision.

DBSCAN Density-based spatial clustering of applications with noise.

DL Deep Learning.

DNN Deep Neural Network.

DRL Deep Reinforcement Learning.

DWA Dynamic Window Approach.

ELU Exponential Linear Unit.

EM Expectation-maximization.

EMA Exponential moving average.

EOR End of the row.

ESA European Space Agency.

FAKD Feature affinity-based knowledge distillation.

FC Fully Connected.

FN False negative.

FNN Feed-forward Neural Network.

FP False positive.

FPN Feature Pyramid Network.

fps Frames per second.

G.O. Graph optimization.

GAN Generative Adversarial Network.

GAP Global Average Pooling.

GD Gradient descent.

GELU Gaussian Error Linear Unit.

GMP Global Max Pooling.

GNSS Global Navigation Satellite System.

GP-GPU General purpose graphic processing unit.

HAR Human Action Recognition.

HR High resolution.

HW Hardware.

IMU Inertial measurement unit.

IoU Intersection over union.

ISO International Organization for Standardization.

KD Knowledge Distillation.

LoS Line of sight.

LR Low resolution.

LSTM Long short-term memory.

MAE Mean absolute error.

MC Dropout Monte Carlo Dropout.

MISR Multi-Image Super-Resolution.

ML Machine Learning.

MLP Multi-layer Perceptron.

MSA Multi-head self-attention.

MSE Mean squared error.

NAG Nesterov accelerated gradient.

NIR Near infrared.

NLoS Non-line-of-sight.

NLP Natural Language Processing.

NMS Non-maximum suppression.

NN Neural Network.

ORT Over real time.

PCA Principal Component Analysis.

PReLU Parametric Rectified Linear Unit.

PSNR Peak Signal to Noise Ratio.

Q.A. Quantization-aware.

QoS Quality-of-Service.

RAMS Residual Attention Multi-Image Super-Resolution model.

ReLU Rectified Linear Unit.

REMNet Range Error Mitigation network.

RGB Red-green-blue.

RGB-D Red-green-blue-depth.

RMSE Root mean squared error.

RMSprop Root Mean Squared propagation.

RNN Recurrent Neural Network.

RRM Residual Reduction Module.

RT Real time.

RTK Real Time Kinematic.

SA Self-attention.

SAC Soft actor-critic algorithm.

SE Squeeze-and-Excitation.

SGD Stochastic gradient descent.

SISR Single-Image Super-Resolution.

SLAM Simultaneous localization and mapping.

SOTA State-of-the-art.

SR Super-Resolution.

SSIM Structural Similarity Index Measure.

TN True negative.

ToA Time-of-arrival.

TP True positive.

TPU Tensor Processing Unit.

TTW Trough-the-wall.

UAV Unmanned Aerial Vehicles.

UGV Unmanned Ground Vehicles.

UWB Ultra-wideband.

ViT Vision Transformer.

VPU Vision Processing Unit.

Bibliography

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A system for large-scale machine learning". In: *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 2016, pp. 265–283.

[2] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Nachaat AbdElatif Mohamed, and Humaira Arshad. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4.11 (2018), e00938.

[3] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1 (1985), pp. 147–169.

[4] Diego Aghi, Simone Cerrato, Vittorio Mazzia, and Marcello 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.

[5] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. "Autonomous navigation in vineyards with deep learning at the edge". In: *International Conference on Robotics in Alpe-Adria Danube Region*. Springer. 2020, pp. 479–486.

[6] Diego Aghi, Vittorio Mazzia, and Marcello Chiaberge. "Local motion planner for autonomous navigation in vineyards with a RGB-D camera-based algorithm and deep learning synergy". In: *Machines* 8.2 (2020), p. 27.

[7] Eirikur Agustsson and Radu 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.

[8] Zhao De-An, Lv Jidong, Ji Wei, Zhang Ying, and Chen Yu. "Design and control of an apple harvesting robot". In: *Biosystems engineering* 110.2 (2011), pp. 112–122.

[9] Simone Angarano, Mauro Martini, Francesco Salvetti, Vittorio Mazzia, and Marcello Chiaberge. "Back-to-bones: Rediscovering the role of backbones in domain generalization". In: *arXiv preprint arXiv:2209.01121* (2022).

[10] Simone Angarano, Vittorio Mazzia, Francesco Salvetti, Giovanni Fantin, and Marcello Chiaberge. "Robust ultra-wideband range error mitigation with deep learning at the edge". In: *Engineering Applications of Artificial Intelligence* 102 (2021), p. 104278.

[11] Simone Angarano, Francesco Salvetti, Mauro Martini, and Marcello Chiaberge. "Generative adversarial super-resolution at the edge with knowledge distillation". In: *Engineering Applications of Artificial Intelligence* 123 (2023), p. 106407.

[12] Simone Angarano, Francesco Salvetti, Vittorio Mazzia, Giovanni Fantin, Dario Gandini, and Marcello Chiaberge. "Ultra-Low-Power Range Error Mitigation for Ultra-Wideband Precise Localization". In: *Science and Information Conference*. Springer. 2022, pp. 814–824.

[13] Federico Angelini, Zeyu Fu, Yang Long, Ling Shao, and Syed Mohsen Naqvi. "2D Pose-Based Real-Time Human Action Recognition With Occlusion-Handling". In: *IEEE Transactions on Multimedia* 22.6 (2020), pp. 1433–1446.

[14] Federico Angelini and Syed Mohsen 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.

[15] Federico Angelini, Jiawei Yan, and Syed Mohsen 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.

[16] Pietro Astolfi, Alessandro Gabrielli, Luca Bascetta, and Matteo Matteucci. "Vineyard autonomous navigation in the echord++ grape experiment". In: *IFAC-PapersOnLine* 51.11 (2018), pp. 704–709.

[17] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *NIPS 2016 Deep Learning Symposium*. 2016.

[18] Oscar C Barawid Jr, Akira Mizushima, Kazunobu Ishii, and Noboru Noguchi. "Development of an autonomous navigation system using a two-dimensional laser scanner in an orchard application". In: *Biosystems Engineering* 96.2 (2007), pp. 139–149.

[19] Valentín Barral, Carlos J Escudero, and José 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.

[20] Daniel Belanche, Luis V Casalo, Carlos Flavian, and Jeroen Schepers. "Service robot implementation: a theoretical framework and research agenda". In: *The Service Industries Journal* 40.3-4 (2020), pp. 203–225.

[21] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. "Attention augmented convolutional networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 3286–3295.

[22] Hédi Ben-Younes, Éloi Zablocki, Patrick Pérez, and Matthieu Cord. "Driving behavior explanation with multi-level fusion". In: *Pattern Recognition* (2021), p. 108421.

[23] Pawel Benecki, Michal Kawulok, Daniel Kostrzewa, and Lukasz Skonieczny. "Evaluating super-resolution reconstruction of satellite images". In: *Acta Astronautica* 153 (2018), pp. 15–25.

[24] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz. "Keyword Transformer: A Self-Attention Model for Keyword Spotting". In: *Proc. Interspeech 2021*. 2021, pp. 4249–4253.

[25] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie-Line Alberi Morel. "Low-Complexity Single-Image Super-Resolution based on Nonnegative Neighbor Embedding". In: *British Machine Vision Conference (BMVC)*. 2012.

[26] Anna Boschi, Francesco Salvetti, Vittorio Mazzia, and Marcello Chiaberge. "A cost-effective person-following system for assistive unmanned vehicles with deep learning at the edge". In: *Machines* 8.3 (2020), p. 49.

[27] Klemen Bregar and Mihael Mohorčič. "Improving indoor localization using convolutional neural networks on computationally restricted devices". In: *IEEE Access* 6 (2018), pp. 17429–17441.

[28] Jose Caballero, Christian Ledig, Andrew Aitken, Alejandro Acosta, Johannes Totz, Zehan Wang, and Wenzhe Shi. "Real-time video super-resolution with spatio-temporal networks and motion compensation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4778–4787.

[29] Jiezhang Cao, Yawei Li, Kai Zhang, and Luc Van Gool. "Video super-resolution transformer". In: *arXiv preprint arXiv:2106.06847* (2021).

[30] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields". In: *IEEE transactions on pattern analysis and machine intelligence* 43.1 (2019), pp. 172–186.

[31] Joao Carreira and Andrew 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)*. July 2017, pp. 6299–6308.

[32] Simone Cerrato, Diego Aghi, Vittorio Mazzia, Francesco Salvetti, and Marcello 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.

[33] Simone Cerrato, Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. "A deep learning driven algorithmic pipeline for autonomous navigation in row-based crops". In: *arXiv preprint arXiv:2112.03816* (2021).

[34] Chen Chen, Roozbeh Jafari, and Nasser 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.

[35] Hanting Chen, Yunhe Wang, Tianyu Guo, Chang Xu, Yiping Deng, Zhenhua Liu, Siwei Ma, Chunjing Xu, Chao Xu, and Wen Gao. "Pre-trained image processing transformer". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12299–12310.

[36] Honggang Chen, Xiaohai He, Linbo Qing, Yuanyuan Wu, Chao Ren, Ray E Sheriff, and Ce Zhu. "Real-world single image super-resolution: A brief review". In: *Information Fusion* 79 (2022), pp. 124–145.

[37] Leiyu Chen, Shaobo Li, Qiang Bai, Jing Yang, Sanlong Jiang, and Yanming Miao. "Review of image classification algorithms based on convolutional neural networks". In: *Remote Sensing* 13.22 (2021), p. 4712.

[38] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 801–818.

[39] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[40] Yu-Yao Chen, Shih-Ping Huang, Ting-Wei Wu, Wei-Ting Tsai, Chong-Yi Liou, and Shau-Gang Mao. "UWB System for Indoor Positioning and Tracking with Arbitrary Target Orientation, Optimal Anchor Location, and Adaptive NLOS Mitigation". In: *IEEE Transactions on Vehicular Technology* (2020).

[41] Sangwoo Cho, Muhammad Maqbool, Fei Liu, and Hassan 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.

[42] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[43] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015).

[44] Oded Cohen, Raphael Linker, and Amos Naor. "Estimation of the number of apples in color images recorded in orchards". In: *International conference on computer and computing technologies in agriculture*. Springer. 2010, pp. 630–642.

[45] Lorenzo Comba, Alessandro Biglia, Davide Ricauda Aimonino, and Paolo Gay. "Unsupervised detection of vineyards by 3D point-cloud UAV photogrammetry for precision agriculture". In: *Computers and Electronics in Agriculture* 155 (2018), pp. 84–95.

[46] R Craig Coulter. *Implementation of the pure pursuit path tracking algorithm*. Tech. rep. Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[47] Stéphane d'Ascoli, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun. "Convit: Improving vision transformers with soft convolutional inductive biases". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2286–2296.

[48] Tao Dai, Jianrui Cai, Yongbing Zhang, Shu-Tao Xia, and Lei Zhang. "Second-order attention network for single image super-resolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11065–11074.

[49] Arthur P Dempster, Nan M Laird, and Donald B Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (1977), pp. 1–22.

[50] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[51] Michel Deudon, Alfredo Kalaitzis, Israel Goytom, Md Rifat Arefin, Zhichao Lin, Kris Sankaran, Vincent Michalski, Samira E Kahou, Julien Cornebise, and Yoshua Bengio. "HighRes-net: Recursive Fusion for Multi-Frame Super-Resolution of Satellite Imagery". In: *arXiv preprint arXiv:2002.06460* (2020).

[52] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[53] Eva AM van Dis, Johan Bollen, Willem Zuidema, Robert van Rooij, and Claudi L Bockting. "ChatGPT: five priorities for research". In: *Nature* 614.7947 (2023), pp. 224–226.

[54] Kris Doelling, Jeongsik Shin, and Dan O Popa. "Service robotics for the home: a state of the art review". In: *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*. 2014, pp. 1–8.

[55] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. "Image super-resolution using deep convolutional networks". In: *IEEE transactions on pattern analysis and machine intelligence* 38.2 (2015), pp. 295–307.

[56] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. "Learning a deep convolutional network for image super-resolution". In: *European conference on computer vision*. Springer. 2014, pp. 184–199.

[57] Chao Dong, Chen Change Loy, and Xiaoou Tang. "Accelerating the super-resolution convolutional neural network". In: *European conference on computer vision*. Springer. 2016, pp. 391–407.

[58] Francisco Dorr. "Satellite image multi-frame super resolution using 3D wide-activation neural networks". In: *Remote Sensing* 12.22 (2020), p. 3812.

[59] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

[60] Timothy Dozat. "Incorporating nesterov momentum into adam". In: *International Conference on Learning Representations Workshop*. 2016.

[61] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[62] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv:1603.07285* (2016).

[63] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[64] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[65] Absalom E Ezugwu, Abiodun M Ikotun, Olaide O Oyelade, Laith Abualigah, Jeffery O Agushaka, Christopher I Eke, and Andronicus A Akinyelu. "A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects". In: *Engineering Applications of Artificial Intelligence* 110 (2022), p. 104743.

[66] Sina Farsiu, M Dirk Robinson, Michael Elad, and Peyman Milanfar. "Fast and robust multiframe super resolution". In: *IEEE transactions on image processing* 13.10 (2004), pp. 1327–1344.

[67] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics Automation Magazine* 4.1 (1997), pp. 23–33. DOI: 10.1109/100.580977.

[68] Yonggan Fu, Wuyang Chen, Haotao Wang, Haoran Li, Yingyan Lin, and Zhangyang Wang. "AutoGAN-Distiller: searching to compress generative adversarial networks". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020, pp. 3292–3303.

[69] Kunihiko Fukushima and Sei Miyake. "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition". In: *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[70] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[71] Nikolaos Gkalelis, Hansung Kim, Adrian Hilton, Nikos Nikolaidis, and Ioannis Pitas. "The i3dpost multi-view and 3d human action/interaction database". In: *2009 Conference for Visual Media Production*. IEEE. 2009, pp. 159–168.

[72] Xavier Glorot, Antoine Bordes, and Yoshua 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.

[73] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. "Compressing deep convolutional networks using vector quantization". In: *arXiv preprint arXiv:1412.6115* (2014).

[74] Juan Angel Gonzalez-Aguirre, Ricardo Osorio-Oliveros, Karen L Rodriguez-Hernandez, Javier Lizarraga-Iturralde, Ruben Morales Menendez, Ricardo A Ramirez-Mendoza, Mauricio Adolfo Ramirez-Moreno, and Jorge de Jesus Lozoya-Santos. "Service robots: Trends and technology". In: *Applied Sciences* 11.22 (2021), p. 10702.

[75] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[76] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[77] Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. "Actions as space-time shapes". In: *IEEE transactions on pattern analysis and machine intelligence* 29.12 (2007), pp. 2247–2253.

[78] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. "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.

[79] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, et al. "Conformer: Convolution-augmented transformer for speech recognition". In: *arXiv preprint arXiv:2005.08100* (2020).

[80] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. "Deep learning with limited numerical precision". In: *International Conference on Machine Learning*. 2015, pp. 1737–1746.

[81] Karthikeyan Gururaj, Anojh Kumaran Rajendra, Yang Song, Choi Look Law, and Guofa 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.

[82] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey 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.

[83] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. "EIE: efficient inference engine on compressed deep neural network". In: *ACM SIGARCH Computer Architecture News* 44.3 (2016), pp. 243–254.

[84] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[85] Shijie Hao, Yuan Zhou, and Yanrong Guo. "A brief survey on semantic segmentation with deep learning". In: *Neurocomputing* 406 (2020), pp. 302–321.

[86] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[87] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian 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.

[89] Zibin He, Tao Dai, Jian Lu, Yong Jiang, and Shu-Tao 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.

[90] Dan Hendrycks and Kevin Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

[91] Geoffrey Hinton. "rmsprop: Divide the gradient by a running average of its recent magnitude". 2012.

[92] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. "Distilling the Knowledge in a Neural Network". In: *NeurIPS Deep Learning and Representation Learning Workshop*. 2014.

[93] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[94] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[95] Jane Holland, Liz Kingston, Conor McCarthy, Eddie Armstrong, Peter O'Dwyer, Fionn Merz, and Mark McConnell. "Service robots in the healthcare sector". In: *Robotics* 10.1 (2021), p. 47.

[96] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural networks* 2.5 (1989), pp. 359–366.

[97] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. "Searching for mobilenetv3". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1314–1324.

[98] Chaur-Heh Hsieh, Jen-Yang Chen, and Bo-Hong Nien. "Deep learning-based indoor localization using received signal strength and channel state information". In: *IEEE access* 7 (2019), pp. 33256–33267.

[99] Jie Hu, Li Shen, and Gang Sun. "Squeeze-and-excitation networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 7132–7141.

[100] Ping Hu, Federico Perazzi, Fabian Caba Heilbron, Oliver Wang, Zhe Lin, Kate Saenko, and Stan Sclaroff. "Real-time semantic segmentation with fast attention". In: *IEEE Robotics and Automation Letters* 6.1 (2020), pp. 263–270.

[101] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[102] Jia-Bin Huang, Abhishek Singh, and Narendra 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.

[103] Linjiang Huang, Yan Huang, Wanli Ouyang, and Liang Wang. "Part-aligned pose-guided recurrent network for action recognition". In: *Pattern Recognition* 92 (2019), pp. 165–176. ISSN: 0031-3203.

[104] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[105] Michal Irani and Shmuel Peleg. "Improving resolution by image registration". In: *CVGIP: Graphical models and image processing* 53.3 (1991), pp. 231–239.

[106] ISO Central Secretary. *Information technology — Vocabulary*. en. Standard ISO/IEC 2382:2015. Geneva, CH: International Organization for Standardization, May 2015. URL: https://www.iso.org/standard/63598.html.

[107] ISO Central Secretary. *Robotics — Vocabulary*. en. Standard ISO 8373:2021. Geneva, CH: International Organization for Standardization, Nov. 2021. URL: https://www.iso.org/standard/75539.html.

[108] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "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.

[109] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. "Data clustering: a review". In: *ACM computing surveys (CSUR)* 31.3 (1999), pp. 264–323.

[110] Wei Ji, Dean Zhao, Fengyi Cheng, Bo Xu, Ying Zhang, and Jinjing Wang. "Automatic recognition vision system guided for apple harvesting robot". In: *Computers & Electrical Engineering* 38.5 (2012), pp. 1186–1195.

[111] Changhui Jiang, Jichun Shen, Shuai Chen, Yuwei Chen, Di Liu, and Yuming Bo. "UWB NLOS/LOS Classification Using Deep Learning Method". In: *IEEE Communications Letters* 24.10 (2020), pp. 2226–2230.

[112] Younghyun Jo, Seoung Wug Oh, Jaeyeon Kang, and Seon Joo Kim. "Deep video super-resolution network using dynamic upsampling filters without explicit motion compensation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3224–3232.

[113] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual losses for real-time style transfer and super-resolution". In: *European conference on computer vision*. Springer. 2016, pp. 694–711.

[114] Md Shaha Nur Kabir, Ming-Zhang Song, Nam-Seok Sung, Sun-Ok Chung, Yong-Joo Kim, Noboru Noguchi, and Soon-Jung Hong. "Performance comparison of single and multi-GNSS receivers under agricultural fields in Korea". In: *Engineering in agriculture, environment and food* 9.1 (2016), pp. 27–35.

[115] Andreas Kamilaris and Francesc X Prenafeta-Boldú. "Deep learning in agriculture: A survey". In: *Computers and electronics in agriculture* 147 (2018), pp. 70–90.

[116] Armin Kappeler, Seunghwan Yoo, Qiqin Dai, and Aggelos K Katsaggelos. "Video super-resolution with convolutional neural networks". In: *IEEE Transactions on Computational Imaging* 2.2 (2016), pp. 109–122.

[117] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. "Multivariate LSTM-FCNs for time series classification". In: *Neural Networks* 116 (2019), pp. 237–245. ISSN: 0893-6080.

[118] Michal Kawulok, Pawel Benecki, Daniel Kostrzewa, and Lukasz Skonieczny. "Evolving imaging model for super-resolution reconstruction". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2018, pp. 284–285.

[119] Michal Kawulok, Pawel Benecki, Szymon Piechaczek, Krzysztof Hrynczenko, Daniel Kostrzewa, and Jakub Nalepa. "Deep learning for multiple-image super-resolution". In: *IEEE Geoscience and Remote Sensing Letters* (2019).

[120] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu 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.

[121] SP Kim, Nirmal K Bose, and Hector M Valenzuela. "Recursive reconstruction of high resolution image from noisy undersampled multiframes". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38.6 (1990), pp. 1013–1027.

[122] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[123] Sven Koenig and Maxim Likhachev. "Fast replanning for navigation in unknown terrain". In: *IEEE Transactions on Robotics* 21.3 (2005), pp. 354–363.

[124] Ron Kohavi, David H Wolpert, et al. "Bias plus variance decomposition for zero-one loss functions". In: *ICML*. Vol. 96. 1996, pp. 275–83.

[125] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[126] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[127] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, et al. "The open images dataset v4". In: *International Journal of Computer Vision* 128.7 (2020), pp. 1956–1981.

[128] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. "Deep laplacian pyramid networks for fast and accurate super-resolution". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 624–632.

[129] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. "Fast and accurate image super-resolution with deep laplacian pyramid networks". In: *IEEE transactions on pattern analysis and machine intelligence* 41.11 (2018), pp. 2599–2613.

[130] Benjamin Langmann, Klaus Hartmann, and Otmar Loffeld. "Depth Camera Technology Comparison and Performance Evaluation." In: *ICPRAM (2)*. 2012, pp. 438–444.

[131] Steven M LaValle et al. "Rapidly-exploring random trees: A new tool for path planning". In: (1998).

[132] Yann LeCun et al. "Generalization and network design strategies". In: *Connectionism in perspective* 19.143-155 (1989), p. 18.

[133] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[134] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[135] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[136] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, 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.

[137] Jun Li, Xianglong Liu, Mingyuan Zhang, and Deqing Wang. "Spatio-temporal deformable 3D ConvNets with attention for action recognition". In: *Pattern Recognition* 98 (2020), p. 107037. ISSN: 0031-3203.

[138] Jingyun Liang, Jiezhang Cao, Guolei Sun, Kai Zhang, Luc Van Gool, and Radu Timofte. "Swinir: Image restoration using swin transformer". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1833–1844.

[139] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung 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.

[140] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[141] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal loss for dense object detection". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[142] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[143] Jie Liu, Jie Tang, and Gangshan Wu. "Residual feature distillation network for lightweight image super-resolution". In: *European Conference on Computer Vision*. Springer. 2020, pp. 41–55.

[144] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C Kot. "Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding". In: *IEEE transactions on pattern analysis and machine intelligence* 42.10 (2019), pp. 2684–2701.

[145] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. "On the variance of the adaptive learning rate and beyond". In: *arXiv preprint arXiv:1908.03265* (2019).

[146] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. "Ssd: Single shot multibox detector". In: *European conference on computer vision*. Springer. 2016, pp. 21–37.

[147] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. "Swin transformer: Hierarchical vision transformer using shifted windows". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10012–10022.

[148] Ziyu Liu, Hongwen Zhang, Zhenghao Chen, Zhiyong Wang, and Wanli 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.

[149] Stuart Lloyd. "Least squares quantization in PCM". In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137.

[150] Iker Lluvia, Elena Lazkano, and Ander Ansuategi. "Active mapping and robot exploration: A survey". In: *Sensors* 21.7 (2021), p. 2445.

[151] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

[152] Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. "Deep transfer learning with joint adaptation networks". In: *International conference on machine learning*. PMLR. 2017, pp. 2208–2217.

[153] Ilya Loshchilov and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations Workshop*. 2019.

[154] Diogo Luvizon, David Picard, and Hedi Tabia. "Multi-task deep learning for real-time 3D human pose estimation and action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* (2020).

[155] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Citeseer. 2013, p. 3.

[156] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074.

[157] James MacQueen et al. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Oakland, CA, USA. 1967, pp. 281–297.

[158] Valerio Magnago, Pablo Corbalán, Gian Pietro Picco, Luigi Palopoli, and Daniele Fontanelli. "Robot Localization via Odometry-assisted Ultra-wideband Ranging with Stochastic Guarantees." In: *IROS*. 2019, pp. 1607–1613.

[159] Mishaim Malik, Muhammad Kamran Malik, Khawar Mehmood, and Imran Makhdoom. "Automatic speech recognition: a survey". In: *Multimedia Tools and Applications* 80.6 (2021), pp. 9411–9457.

[160] K Manikanda Kumaran and M Chinnadurai. "Cloud-based robotic system for crowd control in smart cities using hybrid intelligent generic algorithm". In: *Journal of Ambient Intelligence and Humanized Computing* 11.12 (2020), pp. 6293–6306.

[161] Chengzhi Mao, Kangbo Lin, Tiancheng Yu, and Yuan Shen. "A probabilistic learning approach to UWB ranging error mitigation". In: *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE. 2018, pp. 1–6.

[162] Stefano Marano, Wesley M Gifford, Henk Wymeersch, and Moe Z Win. "NLOS identification and mitigation for localization based on UWB experimental data". In: *IEEE Journal on selected areas in communications* 28.7 (2010), pp. 1026–1035.

[163] Luca Marchionna, Giulio Pugliese, Mauro Martini, Simone Angarano, Francesco Salvetti, and Marcello Chiaberge. "Deep Instance Segmentation and Visual Servoing to Play Jenga with a Cost-Effective Robotic System". In: *arXiv preprint arXiv:2211.07977* (2022).

[164] Sam Marden and Mark 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.

[165] Marcus Märtens, Dario Izzo, Andrej Krzic, and Daniël Cox. "Super-resolution of PROBA-V images using convolutional neural networks". In: *Astrodynamics* 3.4 (2019), pp. 387–402.

[166] David Martin, Charless Fowlkes, Doron Tal, and Jitendra 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.* Vol. 2. IEEE. 2001, pp. 416–423.

[167] Mauro Martini, Simone Cerrato, Francesco Salvetti, Simone Angarano, and Marcello 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.

[168] Jiri Matas, Charles Galambos, and Josef Kittler. "Robust detection of lines using the progressive probabilistic hough transform". In: *Computer vision and image understanding* 78.1 (2000), pp. 119–137.

[169] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. "Sketch-based manga retrieval using manga109 dataset". In: *Multimedia Tools and Applications* 76.20 (2017), pp. 21811–21838.

[170] Vittorio Mazzia. "Machine Learning Algorithms and their Embedded Implementation for Service Robotics Applications". PhD thesis. Politecnico di Torino, 2022.

[171] Vittorio Mazzia, Simone Angarano, Francesco Salvetti, Federico Angelini, and Marcello Chiaberge. "Action Transformer: A self-attention model for short-time pose-based human action recognition". In: *Pattern Recognition* 124 (2022), p. 108487.

[172] Vittorio Mazzia, Aleem Khaliq, Francesco Salvetti, and Marcello Chiaberge. "Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application". In: *IEEE Access* 8 (2020), pp. 9102–9114.

[173] Vittorio Mazzia, Francesco Salvetti, Diego Aghi, and Marcello Chiaberge. "Deepway: a deep learning waypoint estimator for global path generation". In: *Computers and Electronics in Agriculture* 184 (2021), p. 106091.

[174] Vittorio Mazzia, Francesco Salvetti, and Marcello Chiaberge. "Efficient-capsnet: Capsule network with self-attention routing". In: *Scientific reports* 11.1 (2021), pp. 1–13.

[175] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[176] Diganta Misra. "Mish: A self regularized non-monotonic neural activation function". In: *BMVC - Proceedings of the British Machine Vision Conference*. 2019.

[177] Andrea Bordone Molini, Diego Valsesia, Giulia Fracastoro, and Enrico Magli. "DeepSUM: Deep neural network for Super-resolution of Unregistered Multitemporal images". In: *IEEE Transactions on Geoscience and Remote Sensing* (2019).

[178] Andrea Bordone Molini, Diego Valsesia, Giulia Fracastoro, and Enrico Magli. "DeepSUM++: Non-local Deep Neural Network for Super-Resolution of Unregistered Multitemporal Images". In: *arXiv preprint arXiv:2001.06342* (2020).

[179] Ali H Muqaibel, Mohamed A Landolsi, and Mohammed 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.

[180] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Icml*. 2010.

[181] Yurii Nesterov. "A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)". In: *Doklady an ussr*. Vol. 269. 1983, pp. 543–547.

[182] Arne Niitsoo, Thorsten Edelhäuβer, and Christopher 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.

[183] Ben Niu, Weilei Wen, Wenqi Ren, Xiangde Zhang, Lianping Yang, Shuzhen Wang, Kaihao Zhang, Xiaochun Cao, and Haifeng Shen. "Single image super-resolution via a holistic attention network". In: *European conference on computer vision*. Springer. 2020, pp. 191–207.

[184] Nanne van Noord and Eric Postma. "A learned representation of artist-specific colourisation". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2017, pp. 2907–2915.

[185] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016).

[186] Luiz FP Oliveira, Antonio P Moreira, and Manuel F Silva. "Advances in agriculture robotics: A state-of-the-art review and challenges ahead". In: *Robotics* 10.2 (2021), p. 52.

[187] Timothy Otim, Luis E Díez, Alfonso Bahillo, Peio Lopez-Iturri, and Francisco Falcone. "Effects of the body wearable sensor position on the UWB localization accuracy". In: *Electronics* 8.11 (2019), p. 1351.

[188] Daniel W Otter, Julian R Medina, and Jugal K Kalita. "A survey of the usages of deep learning for natural language processing". In: *IEEE transactions on neural networks and learning systems* 32.2 (2020), pp. 604–624.

[189] Dirk Padfield. "Masked object registration in the Fourier domain". In: *IEEE Transactions on image processing* 21.5 (2011), pp. 2706–2718.

[190] George Papandreou, Tyler Zhu, Liang-Chieh Chen, Spyros Gidaris, Jonathan Tompson, and Kevin 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.

[191] Deepak Patil, Munsaf Ansari, Dilisha Tendulkar, Ritesh Bhatlekar, Vijaykumar Naik Pawar, and Shailendra Aswale. "A survey on autonomous military service robot". In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE. 2020, pp. 1–7.

[192] Chiara Plizzari, Marco Cannici, and Matteo Matteucci. "Skeleton-based action recognition via spatial and temporal transformer networks". In: *Computer Vision and Image Understanding* 208 (2021), p. 103219.

[193] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *Ussr computational mathematics and mathematical physics* 4.5 (1964), pp. 1–17.

[194] Alwin Poulose and Dong Seog Han. "UWB Indoor Localization Using Deep Learning LSTM Networks". In: *Applied Sciences* 10.18 (2020), p. 6290.

[195] Raul de Queiroz Mendes, Eduardo Godinho Ribeiro, Nicolas dos Santos Rosa, and Valdir Grassi Jr. "On deep learning techniques to boost monocular depth estimation for autonomous navigation". In: *Robotics and Autonomous Systems* 136 (2021), p. 103701.

[196] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL: https://openreview.net/forum?id=Hkuq2EkPf.

[197] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[198] Joseph Redmon and Ali Farhadi. "YOLO9000: better, faster, stronger". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 7263–7271.

[199] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv preprint arXiv:1804.02767* (2018).

[200] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks". In: *Advances in neural information processing systems* 28 (2015).

[201] Douglas A Reynolds. "Gaussian mixture models." In: *Encyclopedia of biometrics* 741.659-663 (2009).

[202] Giuseppe Riggio, Cesare Fantuzzi, and Cristian Secchi. "A low-cost navigation strategy for yield estimation in vineyards". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 2200–2205.

[203] Itsaso Rodriguez-Moreno, Jose Maria Martinez-Otzeta, Izaro Goienetxea, Igor Rodriguez-Rodriguez, and Basilio Sierra. "Shedding light on people action recognition in social robotics by means of common spatial patterns". In: *Sensors* 20.8 (2020), p. 2436.

[204] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. "FitNets: Hints for Thin Deep Nets". In: *3rd International Conference on Learning Representations, ICLR 2015*. 2015.

[205] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[206] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[207] Priya Roy and Chandreyee Chowdhury. "A survey of machine learning techniques for indoor localization and navigation systems". In: *Journal of Intelligent & Robotic Systems* 101.3 (2021), pp. 1–34.

[208] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[209] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[210] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[211] Bryan C Russell, Antonio Torralba, Kevin P Murphy, and William T Freeman. "LabelMe: a database and web-based tool for image annotation". In: *International journal of computer vision* 77.1 (2008), pp. 157–173.

[212] Kamel S Saidi, Thomas Bock, and Christos Georgoulas. "Robotics in construction". In: *Springer handbook of robotics*. Springer, 2016, pp. 1493–1520.

[213] Francesco Salvetti, Simone Angarano, Mauro Martini, Simone Cerrato, and Marcello 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.

[214] Francesco Salvetti, Vittorio Mazzia, Aleem Khaliq, and Marcello Chiaberge. "Multi-image super resolution of remotely sensed images using residual attention deep neural networks". In: *Remote Sensing* 12.14 (2020), p. 2207.

[215] Arthur L Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3 (July 1959), pp. 210–229.

[216] Simo Särkkä. *Bayesian filtering and smoothing*. Vol. 3. Cambridge University Press, 2013.

[217] Vladimir Savic, Erik G Larsson, Javier Ferrer-Coll, and Peter Stenumgaard. "Kernel methods for accurate UWB-based ranging with reduced complexity". In: *IEEE Transactions on Wireless Communications* 15.3 (2015), pp. 1783–1793.

[218] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. "A review of clustering techniques and developments". In: *Neurocomputing* 267 (2017), pp. 664–681.

[219] Lorenz Schmid, David Salido-Monzú, and Andreas 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.

[220] Jens Schroeder, Stefan Galler, Kyandoghere Kyamakya, and Klaus Jobmann. "NLOS detection algorithms for ultra-wideband localization". In: *2007 4th Workshop on Positioning, Navigation and Communication*. IEEE. 2007, pp. 159–166.

[221] Christian Schuldt, Ivan Laptev, and Barbara Caputo. "Recognizing human actions: a local SVM approach". In: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Vol. 3. IEEE. 2004, pp. 32–36.

[222] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang 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.

[223] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. "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)*. June 2016.

[224] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1874–1883.

[225] Cesar Andres Sierra Pardo, Marcello Chiaberge, Francesco Salvetti, and Simone Angarano. "Remote sensing-based vineyard image segmentation with deep computer vision for precision agriculture". MA thesis. Politecnico di Torino, 2022.

[226] Bruno Silva and Gerhard P Hancke. "IR-UWB-based non-line-of-sight identification in harsh environments: Principles and challenges". In: *IEEE Transactions on Industrial Informatics* 12.3 (2016), pp. 1188–1195.

[227] Bruno J Silva and Gerhard Petrus Hancke. "Ranging Error Mitigation for Through-the-Wall Non-Line-of-Sight Conditions". In: *IEEE Transactions on Industrial Informatics* (2020).

[228] Karen Simonyan and Andrew 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*. Ed. by Yoshua Bengio and Yann LeCun. 2015.

[229] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[230] Leslie N Smith. "Cyclical learning rates for training neural networks". In: *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE. 2017, pp. 464–472.

[231] Liangchen Song, Gang Yu, Junsong Yuan, and Zicheng Liu. "Human Pose Estimation and Its Application to Action Recognition: A Survey". In: *Journal of Visual Communication and Image Representation* (2021), p. 103055.

[232] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[233] Maximilian Stahlke, Sebastian Kram, Christopher Mutschler, and Thomas 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.

[234] Anthony Stentz. "Optimal and efficient path planning for partially known environments". In: *Intelligent unmanned ground vehicles*. Springer, 1997, pp. 203–220.

[235] Enrico Sutera., Vittorio Mazzia., Francesco Salvetti., Giovanni Fantin., and Marcello Chiaberge. "Indoor Point-to-Point Navigation with Deep Reinforcement Learning and Ultra-Wideband". In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART,* INSTICC. SciTePress, 2021, pp. 38–47.

[236] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[237] Ying Tai, Jian Yang, and Xiaoming Liu. "Image super-resolution via deep recursive residual network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 3147–3155.

[238] Ying Tai, Jian Yang, Xiaoming Liu, and Chunyan Xu. "Memnet: A persistent memory network for image restoration". In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 4539–4547.

[239] Radu Timofte, Eirikur Agustsson, Luc Van Gool, Ming-Hsuan Yang, and Lei 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.

[240] Radu Timofte, Rasmus Rothe, and Luc Van Gool. "Seven ways to improve example-based single image super resolution". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1865–1873.

[241] Praveen Tiwari and Praveen Kumar 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.

[242] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. "Training data-efficient image transformers & distillation through attention". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10347–10357.

[243] Michael Tschannen, Olivier Bachem, and Mario Lucic. "Recent advances in autoencoder-based representation learning". In: *arXiv preprint arXiv:1812.05069* (2018).

[244] Iis Tussyadiah. "A review of research into automation in tourism: Launching the Annals of Tourism Research Curated Collection on Artificial Intelligence and Robotics in Tourism". In: *Annals of Tourism Research* 81 (2020), p. 102883.

[245] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization". In: *arXiv preprint arXiv:1607.08022* (2016).

[246] Diego Valsesia and Petros T Boufounos. "Universal encoding of multispectral images". In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2016, pp. 4453–4457.

[247] Diego Valsesia and Enrico Magli. "A novel rate control algorithm for onboard predictive coding of multispectral and hyperspectral images". In: *IEEE Transactions on Geoscience and Remote Sensing* 52.10 (2014), pp. 6341–6355.

[248] Aaron Van den Oord, Yazhe Li, Oriol Vinyals, et al. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* 2.3 (2018), p. 4.

[249] Gül Varol, Ivan Laptev, and Cordelia Schmid. "Long-Term Temporal Convolutions for Action Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.6 (2018), pp. 1510–1517.

[250] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[251] Ivan Vidović and Rudolf Scitovski. "Center-based clustering for line detection and application to crop rows detection". In: *Computers and electronics in agriculture* 109 (2014), pp. 212–220.

[252] Quoc Duy Vo and Pradipta De. "A survey of fingerprint-based outdoor localization". In: *IEEE Communications Surveys & Tutorials* 18.1 (2015), pp. 491–506.

[253] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. "Deep learning for computer vision: A brief review". In: *Computational intelligence and neuroscience* 2018 (2018).

[254] Boyang Wan, Wenhui Jiang, Yu-Ming Fang, Minwei Zhu, Qin Li, and Yang Liu. "Revisiting image captioning via maximum discrepancy competition". In: *Pattern Recognition* 122 (2022), p. 108358.

[255] Xintao Wang, Liangbin Xie, Chao Dong, and Ying 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.

[256] Xintao Wang, Ke Yu, Chao Dong, and Chen Change 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.

[257] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. "Esrgan: Enhanced super-resolution generative adversarial networks". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

[258] Xuanming Wang and Gautam Srivastava. "The security of vulnerable senior citizens through dynamically sensed signal acquisition". In: *Transactions on Emerging Telecommunications Technologies* (2021), e4037.

[259] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[260] Daniel Weinland, Remi Ronfard, and Edmond Boyer. "Free viewpoint action recognition using motion history volumes". In: *Computer vision and image understanding* 104.2-3 (2006), pp. 249–257.

[261] Jinming Wen, Li He, and Fumin Zhu. "Swarm robotics control and communications: Imminent challenges for next generation smart logistics". In: *IEEE Communications Magazine* 56.7 (2018), pp. 102–107.

[262] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. "Cbam: Convolutional block attention module". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[263] Yuxin Wu and Kaiming He. "Group normalization". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 3–19.

[264] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. "Unsupervised feature learning via non-parametric instance discrimination". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 3733–3742.

[265] Henk Wymeersch, Stefano Maranò, Wesley M Gifford, and Moe Z Win. "A machine learning approach to ranging error mitigation for UWB localization". In: *IEEE transactions on communications* 60.6 (2012), pp. 1719–1728.

[266] Lu Xia, Chia-Chih Chen, and Jake 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.

[267] Xuesu Xiao, Bo Liu, Garrett Warnell, and Peter Stone. "Motion planning and control for mobile robot navigation using machine learning: a survey". In: *Autonomous Robots* (2022), pp. 1–29.

[268] Zhuoling Xiao, Hongkai Wen, Andrew Markham, Niki Trigoni, Phil Blunsom, and Jeff Frolik. "Non-line-of-sight identification and mitigation using received signal strength". In: *IEEE Transactions on Wireless Communications* 14.3 (2014), pp. 1689–1702.

[269] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853* (2015).

[270] Sijie Yan, Yuanjun Xiong, and Dahua 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.

[271] Ru Ying, Ting Jiang, and Zhihao Xing. "Classification of transmission environment in UWB communication using a support vector machine". In: *2012 IEEE Globecom Workshops*. IEEE. 2012, pp. 1389–1393.

[272] Changqian Yu, Changxin Gao, Jingbo Wang, Gang Yu, Chunhua Shen, and Nong Sang. "Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation". In: *International Journal of Computer Vision* 129.11 (2021), pp. 3051–3068.

[273] Jiahui Yu, Yuchen Fan, Jianchao Yang, Ning Xu, Zhaowen Wang, Xinchao Wang, and Thomas Huang. "Wide activation for efficient and accurate image super-resolution". In: *arXiv preprint arXiv:1808.08718* (2018).

[274] Yuhui Yuan, Xiaokang Chen, Xilin Chen, and Jingdong Wang. "Segmentation transformer: Object-contextual representations for semantic segmentation". In: *arXiv preprint arXiv:1909.11065* (2019).

[275] Linwei Yue, Huanfeng Shen, Jie Li, Qiangqiang Yuan, Hongyan Zhang, and Liangpei Zhang. "Image super-resolution: The techniques, applications, and future". In: *Signal Processing* 128 (2016), pp. 389–408.

[276] Faheem Zafari, Athanasios Gkelias, and Kin K Leung. "A survey of indoor localization systems and technologies". In: *IEEE Communications Surveys & Tutorials* 21.3 (2019), pp. 2568–2599.

[277] Sergey Zagoruyko and Nikos Komodakis. "Paying More Attention to Attention: Improving the Performance of Convolutional Neural Networks via Attention Transfer". In: *5th International Conference on Learning Representations, ICLR 2017*. 2017.

[278] Matthew D Zeiler. "Adadelta: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701* (2012).

[279] Zhuoqi Zeng, Rubing Bai, Lei Wang, and Steven 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.

[280] Roman Zeyde, Michael Elad, and Matan Protter. "On single image scale-up using sparse-representations". In: *International conference on curves and surfaces*. Springer. 2010, pp. 711–730.

[281] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver 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.

[282] Yiman Zhang, Hanting Chen, Xinghao Chen, Yiping Deng, Chunjing Xu, and Yunhe 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.

[283] Yulun Zhang, Kunpeng Li, Kai Li, Lichen Wang, Bineng Zhong, and Yun 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.

[284] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. "Residual dense network for image super-resolution". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 2472–2481.

[285] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. "Loss functions for image restoration with neural networks". In: *IEEE Transactions on computational imaging* 3.1 (2016), pp. 47–57.

[286] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. "Object detection with deep learning: A review". In: *IEEE transactions on neural networks and learning systems* 30.11 (2019), pp. 3212–3232.

[287] Fengda Zhu, Yi Zhu, Vincent Lee, Xiaodan Liang, and Xiaojun Chang. "Deep learning for embodied vision navigation: A survey". In: *arXiv preprint* (2021).

[288] Jurgen Zoto, Maria Angela Musci, Aleem Khaliq, Marcello Chiaberge, and Irene 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.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.