POLITECNICO DI TORINO

Master's Degree in MECHATRONIC ENGINEERING

MASTER'S THESIS

A Deep Learning approach to Instance Segmentation of indoor environment

Supervisors
Prof. Marcello CHIABERGE
Dott. Andrea EIRALE

Candidate
Riccardo TESSE

ACADEMIC YEAR: 2021-2022

To my Family.

Abstract

Nowadays, mobile robots are frequently used in both indoor and outdoor situations, including agriculture, transportation in industries, surveillance, and cleaning buildings. These are being developed for several applications where long-term capabilities would be advantageous. The primary goal of mobile robotics is to build fully autonomous machines, meaning that they must be able to carry out their jobs without assistance from humans. Their industrial and technical use is continuously becoming more significant, especially when reliability (the uninterrupted and dependable completion of tasks like surveillance), accessibility (the inspection of locations that are inaccessible to humans, such as confined spaces, hazardous environments, or remote sites), or cost are considered. Computer vision is playing a vital part in making these projects more efficient due to the enormous strides that Machine Learning and Deep Learning have achieved in the sector. These innovations significantly altered how tracking and detecting issues are tackled, making real-world applications considerably more practical and successful.

The goal of this thesis is to investigate a system that can segment floor plans into individual rooms. Several robotics activities depend on this, including topological mapping, semantic mapping, place categorization, human-robot interaction, and automated commercial cleaning. Different map partitioning strategies can be used to complete this task. The Mask R-CNN model has been used to fulfil this target successfully. This network, an extension of Faster R-CNN, enables the prediction of an object mask in conjunction with the branch already in place for bounding box recognition. Since there is not a reasonably large, publicly accessible dataset of floor plans, this thesis's research involved creating one with the corresponding annotations.

The entire dataset containing 4224 images is then used to train the Mask R-CNN model, allowing us to obtain a neural network capable of performing an instance segmentation task on them. Once the model has been trained and validated, a new floor plan map is produced using measurements from a LIDAR sensor.

Then, the map is processed using computer vision software to create a crisper and cleaner map of the surrounding area and to prepare it for the segmentation method.

This work can be used as a foundation for creating more sophisticated systems capable of automatically classifying rooms (for instance, by including various room typologies in the dataset), or by integrating the algorithm onto a mobile robot to perform segmentation after mapping an entire area.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Technology advancements in recent years have increased the number of environment-mapping algorithms that are available, producing ever-improving accuracy results.
Modern methods based on neural networks have enabled the resolution of issues that were intractable in the past. This is achievable as a result of the expansion of publicly accessible data in recent years, which has made it possible to produce a lot more material for analysis. The economist wrote an article titled “The world’s most valuable resource is no longer oil, but data” and published it on May 6, 2017 [1]. Humanity has entered a data generation and collection bonanza as a consequence of an increase in internet users and the realization of the importance of data. Every day more than 2.5 quintillion bytes of data are generated, and as IoT (Internet of Things) becomes more and more popular, that number is likely to rise even further [2].
Due to the huge advancements that Machine Learning and Deep Learning have made in the field, computer vision is playing a crucial role in improving the efficiency of these projects that space from health care to space research. Real-world applications have become significantly more useful and effective on account of these breakthroughs.
There has been a rapid advancement in computer vision. Fundamentally, it is a technology that enables machines to process their environment similarly to humans. It has taken decades of research and experimentation to replicate the multitasking and fast decision-making abilities of the human brain in machines. Today, we are able to create computer vision models that can identify things, determine their shapes, forecast object motions, and perform essential actions depending on the information. The development of computer vision models has led to the development of self-driving automobiles, aerial

mapping, surveillance applications, and numerous other extended reality technologies like AR and VR.

This thesis investigates a method for separating floor plans into their individual rooms. This is essential for many robotics tasks, such as automated commercial cleaning and topological and semantic mapping, as well as place categorization, and human-robot interaction. For this job, many map-splitting techniques can be applied. Different works analyse the literature on room segmentation, in particular [3] offers four publicly accessible implementations of well-known techniques that are adjusted to provide segmentations into whole rooms and target the semantic mapping domain. The chapters that follow the most popular Machine Learning classification methods are briefly discussed, with a deeper focus on Deep Learning techniques and an explanation of what a Neural Network is, how it functions, and how it is used for image classification and segmentation. Ultimately, the principal strategy employed to accomplish the project's aim is highlighted, together with any challenges encountered and the remedies chosen. A brief report on upcoming updates and integrations follows, outlining how this project might be applied to more implementations.

1.2 Thesis structure and purpose

In recent years, various research and algorithms designed to divide up entire environments into rooms have been proposed. As shown in [3], various ways have been investigated. The major goal of this thesis project is to separate floor maps into several rooms using a deep learning approach. This has been accomplished by using a large collection of 2D floor plan maps without furniture. This is done to make it easier to scan an environment using a 360-degree Lidar sensor, as it might be challenging to get a clear view of the surroundings using only this kind of sensor. A few techniques can be used to clean up images and create a high-quality floor map image (using OpenCV). To perform the task of this project a Mask R-CNN is employed. This deep network is mainly used for image processing, in particular for *object detection* and *instance segmentation*. An instance segmentation problem for an image consists in differentiating every single

object, appropriately identified (object detection) and classified, to determine its exact position and differentiate it from any other instance (also belonging to the same class).
This thesis project consists of six chapters, the *first chapte*r serves as an introduction to the project, outlining its main motivations and objectives as well as briefly describing the subjects covered in the following sections. The *second chapter* provides a theoretical overview of Machine Learning, Deep Learning neural networks, how they operate, and why they are crucial for image classification and object detection. The *third chapter* is an insight into the deep neural network used for our task (Mask R-CNN), in particular how it has been developed starting from the Region-Based neural networks. Furthermore, the evaluation metrics for instance segmentation problems have been explained. The *fourth chapter* presents the dataset used for the model's training, and how it has been built and split. The training process and the parameter tuning procedure are highlighted. The *fifth chapter* shows the results obtained by the trained model and how it performs on new data obtained from gazebo simulation and floor plan maps used in other works [3]. The *final chapter* wraps up the paper by demonstrating how the results of this project might be improved, and how the main work can be extended to more difficult cases. It is also shown several real-world uses for which, this research can be helpful.

Chapter 2

Background

Modern approaches based on neural networks have allowed the creation of new models able to solve classification and regression problems. The fundamental ideas of Machine Learning as well as the most current advancements in Deep Learning will be covered in this chapter. Convolutional neural networks and their use in computer vision applications will be described. Additionally, a further theoretical explanation is provided for object detection, semantic segmentation, and instance segmentation. The last paragraph is devoted to providing a quick overview of the software infrastructure utilized in the project's development.

2.1 Machine Learning

The history of Machine Learning resides over 60 years ago when Arthur Lee Samuel coined for the first time this term. Considering his groundbreaking machine learning research, which he started in 1949, he is regarded as the father of Artificial Intelligence [4]. According to Arthur Samuel, Machine Learning is a "*Field of study that gives computers the ability to learn without being explicitly programmed"* however a more accurate definition is provided by Tom Mitchell "*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*"[5].
Arthur Samuel startled the entire world in 1962. He created a computer that was capable of competing against Robert Nealy, the current checkers' champion. The machine triumphed, but the news wasn't just about the victory. The victory's supporting software was what would alter the course of history. Instead of entering all the possible outcomes from a checkerboard into his computer, he told it to respond based on previous games. The computer "learned" to master the board after playing game after game while

analysing dozens of variables, estimating risk, and formulating the following, most effective moves. The same “machine learning” techniques that Samuel utilized in his early experiments are used now in practically every area.

Machine Learning allows us to learn from experience. It is mastering something when the performance of the program gets better after conducting a task or completing an action. The strength of an ML approach is the ability to produce results without writing rules but enable the machine to derive the rules from data; for this reason, it is an essential technology in fields where creating rules is very challenging, such as speech recognition, image segmentation, and object classification.

These algorithms can be used in a variety of fields, including medicine, speech recognition, computer vision, and generally whenever a direct approach cannot complete the task to a sufficient level of performance. Classification, regression, grouping, anomaly detection, dimensional reduction, or predictive analysis are the performed tasks.

The spam filter is one of the early uses of machine learning, which is currently very widespread. This is a program that has developed the ability to distinguish between legitimate and spam emails based on messages that users have received and flagged.

By identifying unusually frequent patterns of words in the spam samples compared to the conventional examples, the spam filter can learn which words and phrases are effective predictors of spam. The subsequent figure illustrates how machine learning’s strategy works.

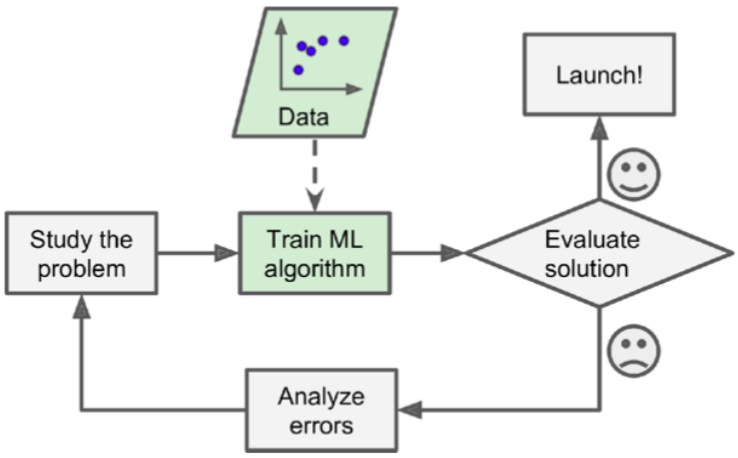

Figure 2.1: Machine Learning strategy [6].

There is a great amount of machine learning systems, and they can be split into different categories:

- **Supervised learning** is made up of a training dataset that also contains the expected outcomes, and these are the "labels." The classification of spam emails can provide a nice explanatory example. These algorithms are the most popular ones: Linear Regression, Logistic Regression, k-Nearest Neighbors, Support Vector Machines (SVMs), Decision Trees and Neural Networks.

- **Unsupervised Learning** approach uses an opposite attitude to the problem with respect to the first one, in fact, the machine is more independent since it must learn and identify complex schemes on its own. The output is not specified and so there are no labels. The algorithm has to recognize some data's properties and classify these data (clustering), detect some anomalies and so on. The most significant unsupervised learning techniques are Clustering, Anomaly detection and novelty detection, Association rule learning, visualization, and dimensionality reduction.

- **Semi-supervised learning** algorithm deals with data that are partially labelled. This is a combination of supervised and unsupervised algorithms and allows a great improvement in accuracy. A common application is the text document classifier. An illustrative example could be photo-service systems that cluster a certain number of people in the images submitted by the user. The model can identify the people in every other picture once the names of the persons are inserted into one.

- **Reinforcement learning**, this approach exploits a completely different method. The learning system "agent" take some action and register a score that could be a "reward" or a "penalty". It learns by itself what is the best procedure to follow, and this is called "policy". Fields including game theory, control theory, operation research, and information theory use these kinds of models. For instance, some applications include

autonomous driving tasks like trajectory optimization, motion planning or autonomous parking.

2.1.1 Machine Learning features

Particular attention must be paid to the training of our Machine Learning model. When some data are submitted to train the models, different difficulties could arise. For instance, the *quantity* of training data must be adequate to produce the best results. This occurs for extremely difficult problems like speech or image recognition since it takes a lot of data to get high accuracy. Microsoft researchers Michele Banko and Eric Brill demonstrated in a well-known study written in 2001 that various diverse ML algorithms, including simple ones, performed nearly identically well on a challenging natural language task [6]. Another important aspect to consider is the *quality* of the data. It is essential that your training data be reflective of the new cases you want to generalize. Naturally, it will be more difficult for the system to recognize the underlying patterns if your training data is full of mistakes, anomalies, and noise (for example, as a result of poor-quality measurements), thus your system is less likely to work successfully. Some operations could be done to make your data ready for the training, including the extraction and exclusion of some features.

2.1.2 Underfitting and Overfitting

There is a fundamental aspect of supervised machine learning to be considered and it is the concept of *overfitting* and *underfitting* the training data. Generally, the model training is evaluated through the computation of errors. These are extremely important to discover if the training is going smoothly. The concept of *overfitting* and *underfitting* are connected with a model's capacity to successfully generalize or precisely map inputs to outputs, and they are all strongly tied to the bias-variance trade-off. The algorithm gains the ability to generalize broad ideas or underlying trends from particular data points during the learning process. When exposed to real data with many of the same qualities, the algorithm ought to be able to translate inputs into outputs. Without bias or excessive variation, underfitting

or overfitting, the ideal model would generalize effectively. But in practice, this is a challenging process.

Overfitting is a phenomenon that is highly likely to occur when a model's capacity to apprehend from the training dataset is extremely high. In fact, it will also learn subtle patterns of the training set that also comprehend the noise. As a result, is unable to generalize to other instances, i.e., it is inefficient in estimating any other instance that does not belong to the training set. When the training error is relatively small and the error on the test set is significantly bigger, this event is practically recognizable.

Underfitting, as one might expect, is the antithesis of overfitting: the model is too simple, it is not able to fit the data and fails to generalize. The error on the training set and test set is incredibly high. In this case, the model can adopt some solutions to prevent underfitting. One approach is the augmentation of the training duration.

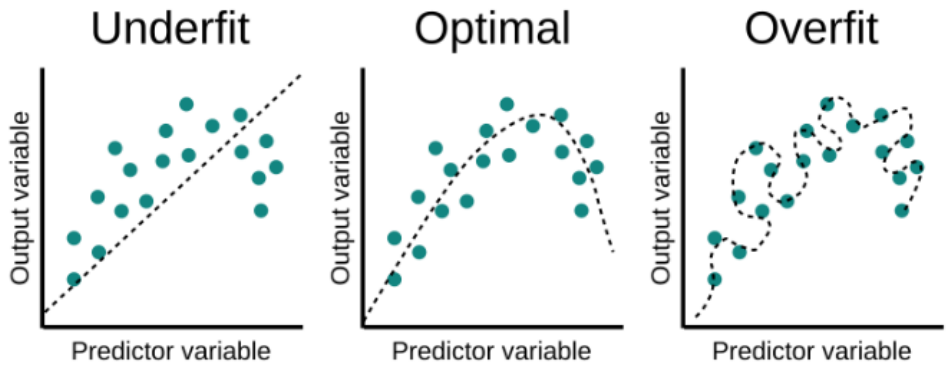

Figure 2.2: Overfitting and Underfitting representation.

Hyperparameters can be used to influence the model's behaviour. These values are initially standard, and they are defined by users rather than the algorithm. There are third-party algorithms that can determine which hyperparameters are best for the main model. Figure 2.2 depicts a polynomial regression example with a single hyperparameter: the function degree. The polynomial function has a high degree on the right, in fact, it overfits

the data. On the other hand, when the model is trained for a small number of epochs, it underfits the data and so it does not represent accurately the trend. By choosing accurately the parameters of the training, the model learns how to fit the data properly.

2.1.3 Validation and Test set

As seen before, the model must generalize properly on the new data. The best solution to achieve the generalization is to divide the entire dataset into two parts: the *training set* and the *test set*. After the learning process has been finished, the test set made of instances from the same distribution as the training set can be used to estimate a learner's generalization error. It is fundamental that the test cases are not utilized to make decisions regarding the model, particularly its hyperparameters.

Therefore, we build a *validation set* from the training data. The training data are divided into two distinct subsets. The parameters are learned using one of these selections. The other subset is our validation set, which is used to assess the generalization error during or after training and allows the hyperparameters to be adjusted as needed. Even though this may be misinterpreted with the larger pool of data used for the entire training process, the subset of data used to learn the parameters is still commonly referred to as the training set. The *validation set* is the subset of data used to inform the selection of hyperparameters. Generally, around 80% is used for the training data while the other 20% is used for validation.

2.2 Deep Learning

2.2.1 Evolution of Deep Learning

Deep Learning has progressed over time, generating tremendous upheaval in sectors and business domains. It is a subset of Machine Learning that uses algorithms to process data, simulate the thinking process, and even create abstractions. Deep Learning processes data using layers of algorithms, interpret human speech and recognizes items visually.

In the literature, it is classified as part of the Artificial Neural Network (ANN) family, however, the two terms are sometimes used interchangeably. ANNs are inspired by biological human brain architecture and how neurons interact with one another, although they are not intended to be realistic brain models. The first neural network was introduced in 1943 by Warren McCulloch and Walter Pitts, who created a computer model based on human brain neural networks. To simulate the mental process, Warren McCulloch and Walter Pitts employed a combination of mathematics and algorithms known as threshold logic. A McCulloch-Pitts neuron receives inputs, computes a weighted sum, and returns '0' if the result is less than a certain threshold and '1' otherwise. Frank Rosenblatt developed the *perceptron* in 1957 by combining Donald Hebb's idea of brain cell interaction with Arthur Samuel's Machine Learning work. The *perceptron* was originally intended to be a machine rather than a program. The software, which was initially intended for the IBM 704, was put in a custom-built image recognition machine named the Mark 1 perceptron.

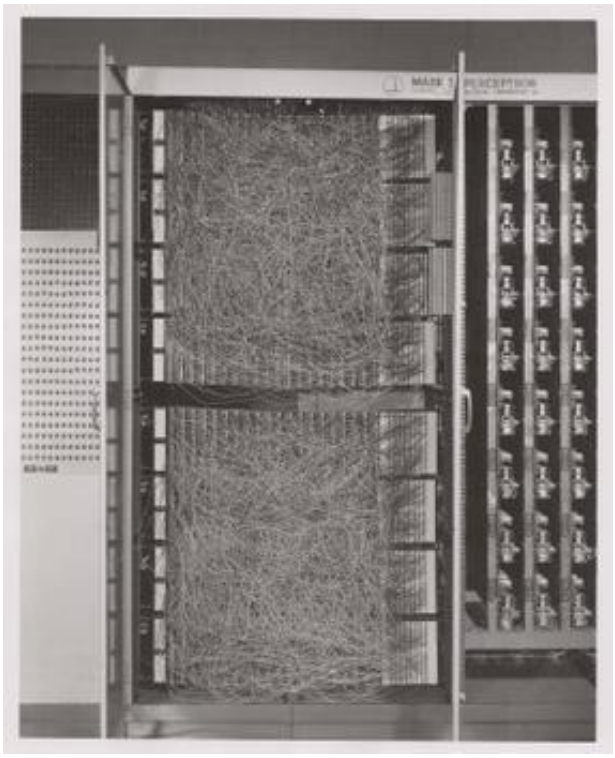

Figure 2.3: Mark I Perceptron machine [7].

The discovery and application of multilayers in neural network research paved the way in the 1960s. It was observed that giving and employing two or more layers in the perceptron provided much more processing power than a single layer perceptron. In 1960, Henry J. Kelley is credited with establishing the fundamentals of a continuous Back Propagation Model. Stuart Dreyfus produced a simpler version based solely on the chain rule in 1962. While the concept of backpropagation (the backward transmission of errors for training purposes) existed in the early 1960s, it was cumbersome and inefficient, and would not be practical until 1985. The first AI winter resulted from several factors coming together.

The main obstacle to AI and neural networks in that era was the computational requirements, which made any structure with more than two layers too complex to be computed with the technologies of that time. Moreover, other ML algorithms were discovered and as a result, neural networks were abandoned again for a period. Now there is currently a new surge in interest in ANN. This is achievable for numerous reasons, including the amount of data available to train the neural network, the development in computing power that allows for the training of huge neural networks efficiently, the advancement of algorithms, and so on.

2.2.2 Artificial Neuron

The artificial neuron, the fundamental element of Deep Learning, takes its name from the biological one because of their resemblance. As anticipated before, McCulloch and Pitts are the pioneers of a simple model similar to the biological neuron. Frank Rosenblatt in 1960 created the *perceptron*, based on the study of the other two scientists. The perceptron or also called threshold logic unit (TLU), is the simplest ANN. It takes different inputs and after computing a weighted sum of them, it produces a single output according to an activation function.

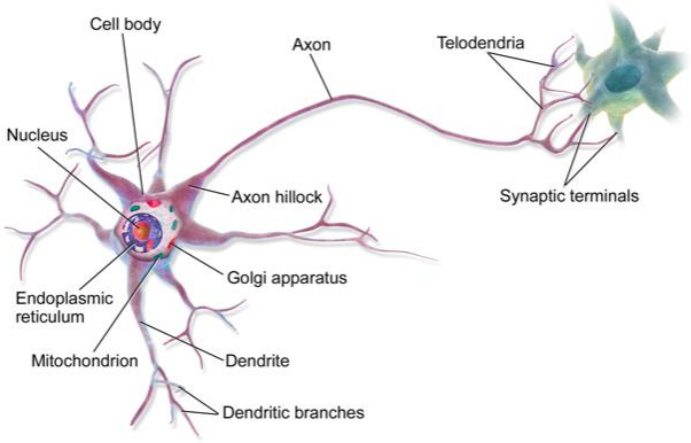

Figure 2.4: Biological neuron representation [8].

$$y = \begin{cases} 0, & if \sum_k w_k x_k \leq threshold \\ 1, & if \sum_k w_k x_k > threshold \end{cases} \quad (2.1)$$

The perceptron typically consists of two layers: the input layer, which contains the various elements provided to the network, and the output layer, which contains the element that the network calculates.

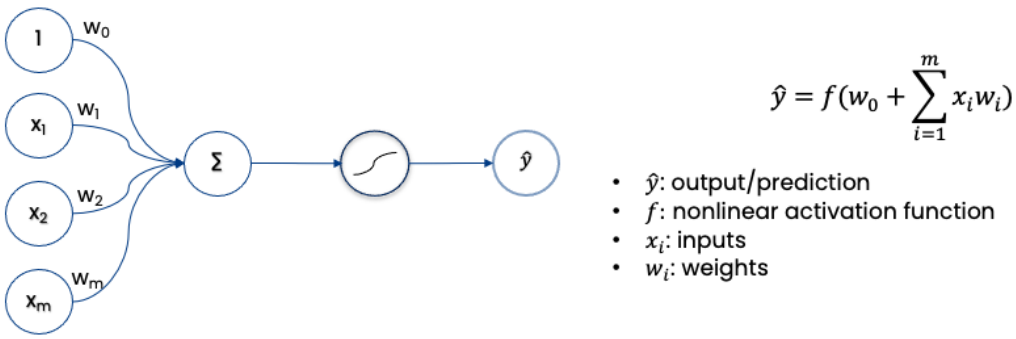

Figure 2.5: Threshold Logic Unit representation.

The model has some limitations, small changes in the input, weights or threshold can change in drastic way the output since it can assume only values 0 and 1. This problem

can be solved by modifying the activation function, in particular when the step function is substituted by the sigmoid function. In the following pages, there will be described the most common activation functions. Another limitation pointed out by Marvin Minsky and Seymour Papert in 1969 is the incapacity to solve futile problems as the *Exclusive OR* (*XOR*) classification problem. This barrier can be removed by combining various perceptrons, giving birth to *Multi Layer Perceptron* (*MLP*). It consists of three or more fully connected layers, meaning that each node in one layer connects with a certain weight w_{ik} to every node in the following layer.

2.2.3 Neural Networks architecture

The *Multi Layer Perceptron* gives an idea of the general architecture of Artificial Neural Network (ANN). Three different types of layers make up this system: the ***Input layer***, which receives data and transfers it to the other levels; the ***Hidden layer***, which contains one or more layers; and finally, the ***Output layer***, which presents the results. *Hidden layers* are what give neural networks their exceptional performance and intricacy. Decision-making is typically more complex and produces better results when there are more hidden levels. Typically, a *deep neural network* (DNN) is referred to as an ANN that has a deep stack of hidden layers.

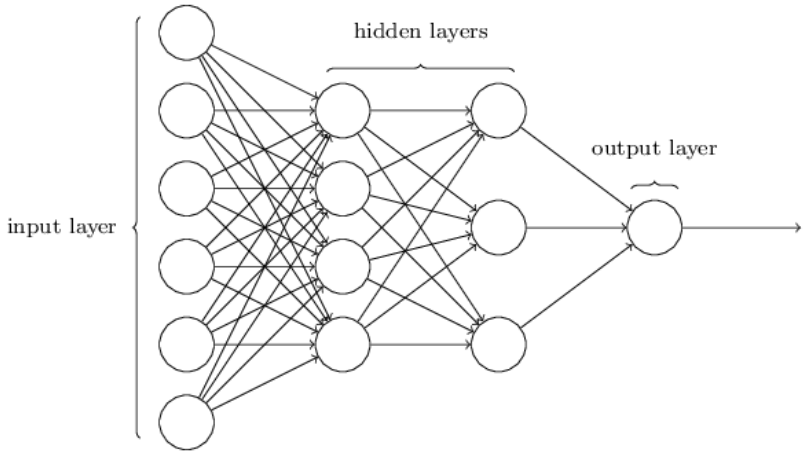

Figure 2.6: Artificial Neural Network Architecture [8].

Systems that have the ability to learn are extremely adaptive. These adjust information they have previously gathered, change their internal organization, and employ input from the outside world in their attempt to learn. That's how ANN acts, they adjust and adapt their architecture to learn. More specifically, the ANNs modify the weights of connections in accordance with input and targeted output.

In supervised learning, we essentially get a training set that includes a vector of desired output values and a vector of input values. The *cost function* computes the error vector once the network determines the output for one of the inputs. This error shows how closely our approximation matches the ground truth value. The mean squared error function is one of the most used cost functions:

$$J(w,b) = \frac{1}{2n}\sum_{x} \|y(x) - \hat{y}\| \tag{2.2}$$

The output generated by the neural network is represented by $y(x), x$ is the training input vector and $\hat{y}$ is the desired output. The elements w and b represent respectively the weights and biases. The process used to learn and modify accordingly the weights is called *backpropagation*. It is an advanced and complex algorithm that allows ANNs to modify the weights quickly.

The ***backpropagation*** algorithm follows this procedure:

- Firstly, it makes a prediction (*forward pass*) and computes the error.
- It performs a reverse run over each layer to calculate the contribution of each connection's error (*reverse pass*).
- Lastly, it adjusts the connection weights to minimize the error (*Gradient Descent step*).

The connection weights of all the hidden layers must be initialized randomly for training to succeed. For instance, if all weights and biases are initialized to zero, every neuron in a particular layer will be completely identical and will continue to be equal as long as

backpropagation continues to affect them in the same manner. In other words, even though your model has hundreds of neurons per layer, it will behave as if it has just one neuron [6].

An important role in neural networks is played by ***activation functions***. It determines if a neuron should be activated or not, and so it evaluates whether or not the neuron's input to the network is significant throughout the prediction process. The output computed by a neural network is strongly conditioned by the activation function used, which has a fundamental role in the speed of convergence and accuracy of the ANN. Moreover, in modern deep neural networks, the choice of the activation function to be used depends on the type of layer it is associated with. The main purpose of activation functions is to add non-linearity to neural networks [11]. The most important ones are described in the following paragraph.

Binary Step function

The first function is the binary step function. A defined threshold determines whether or not a neuron should be triggered. The function is represented as follows.

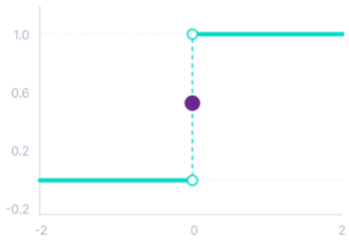

Figure 2.7: Binary Step function representation [11].

The mathematical representation is the following:

$$f(x) = \begin{cases} 0, & for\ x < 0 \\ 1, & for\ x \geq 0 \end{cases} \tag{2.3}$$

The output can assume only the values 0 and 1, so it cannot be used for multi-class classification problems. Furthermore, the gradient is zero and can cause problems in the backpropagation process. For these reasons, it is rarely employed.

Sigmoid function

The sigmoid function is one of the most known activation functions, it takes as input any real value and displays as output a value between 0 and 1.

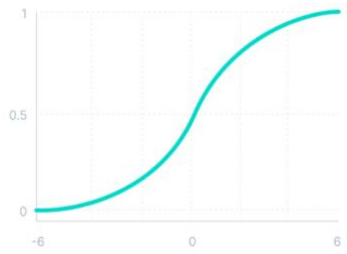

Figure 2.8: Sigmoid function representation [11].

The mathematical formulation is:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

It is frequently employed when the output has to be a probability prediction. It also provides a smooth gradient even if the values are significant between -3 and 3.

Hyperbolic tangent function

It has the S-shape as the sigmoid function, but the output range goes from -1 to 1.
It is represented by the following function:

$$f(x) = tanh(x) = \frac{sinh\,(x)}{cosh\,(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

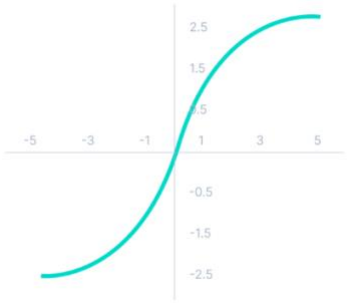

Figure 2.9: Hyperbolic tangent function representation [11].

The main characteristic is that it can produce a negative output. By getting the mean near 0, it helps in centring the data. This greatly simplifies learning for the subsequent layer.

ReLU function

It is the most used function in a neural network's hidden layers. ReLU stands for Rectified linear unit.

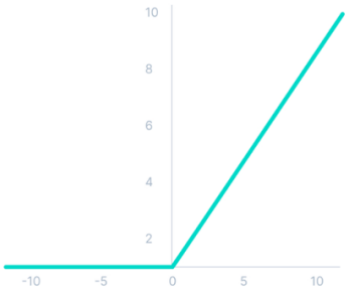

Figure 2.10: ReLU function representation [11].

The function has the following form:

$$f(x) = max(0, x) \tag{2.6}$$

Compared to *tanh* and *sigmoid*, it requires fewer computations since its mathematical operations are simpler. It speeds up the convergence towards the global minimum. The ReLU function has a drawback: the gradient is zero when computed on the negative side and so during backpropagation, weights and biases of some neurons are not updated.

The Leaky ReLU is employed to solve the latter problem. It presents a small positive slope on the negative side, and it is mathematically represented by:

$$f(x) = max(0.1x, x) \tag{2.7}$$

It takes longer to learn the model parameters because the gradient for negative values is small [11].

Softmax function

For classification problems, the sigmoid activation function is well suited in case you have only two classes. If the number of classes is higher, the softmax activation function shall be used, and it is defined as:

$$softmax(z_j) = \frac{e^{z_j}}{\sum_k e^{z_k}} \tag{2.8}$$

The Softmax function is described as a combination of multiple sigmoids.

2.2.4 Gradient Descent

Gradient Descent is an optimization technique that can locate the best solution to a variety of problems. It is a method for minimizing a given function to its local minimum by iteratively adjusting the parameters of a function that generally, for regression problems, is the *MSE*.

But how it calculates the minimum of this function? Starting from a random point on the cost function, it computes the local gradient of the error function with respect to n-dimensional vector *v* and it goes in the direction of descending gradient. Let us consider a variation of the vector *v*. This modification will imply a change of the cost function *J* that can be expressed as follows:

$$\Delta J \approx \frac{\partial J}{\partial v_1}\Delta v_1 + \frac{\partial J}{\partial v_2}\Delta v_2 + \cdots + \frac{\partial J}{\partial v_i}\Delta v_i + \cdots + \frac{\partial J}{\partial v_n}\Delta v_n \qquad (2.9)$$

The objective is to find a combination of v_i that make ΔJ negative and so to determine the global minimum of the cost function $J(w, b)$. We can also exploit the gradient definition and write J as the vector of the partial derivatives.

$$\nabla J = \left(\frac{\partial J}{\partial v_1}, \ldots, \frac{\partial J}{\partial v_n}\right)^T \qquad (2.10)$$

The expression (2.9) can be rewritten as:

$$\Delta J \approx \nabla J \cdot \Delta v \qquad (2.11)$$

As stated before, it is necessary to determine Δv that minimises the cost function (making ΔJ negative). In particular, we select:

$$\Delta v = -\eta \cdot \nabla J \qquad (2.12)$$

The parameter η represents the *learning rate*. It assumes always a small and positive value. Substituting the expression 2.12 into 2.11 we obtain:

$$\Delta J \approx -\eta \nabla J \cdot \nabla J = -\eta \|\nabla J\|^2 \qquad (2.13)$$

The second term $\|\nabla J\|^2 \geq 0$, implying that $\Delta J \leq 0$. The new vector of input is:

$$v' = v - \eta \cdot \nabla J \qquad (2.14)$$

In conclusion, the gradient descent algorithm computes iteratively the gradient ∇J before moving in the opposite direction and "dropping down" the valley's slope. Finding the

global minimum is not always possible, different factors can lead to a local minimum, but the algorithm is very efficient and most of the time allows the ANN to learn [8].

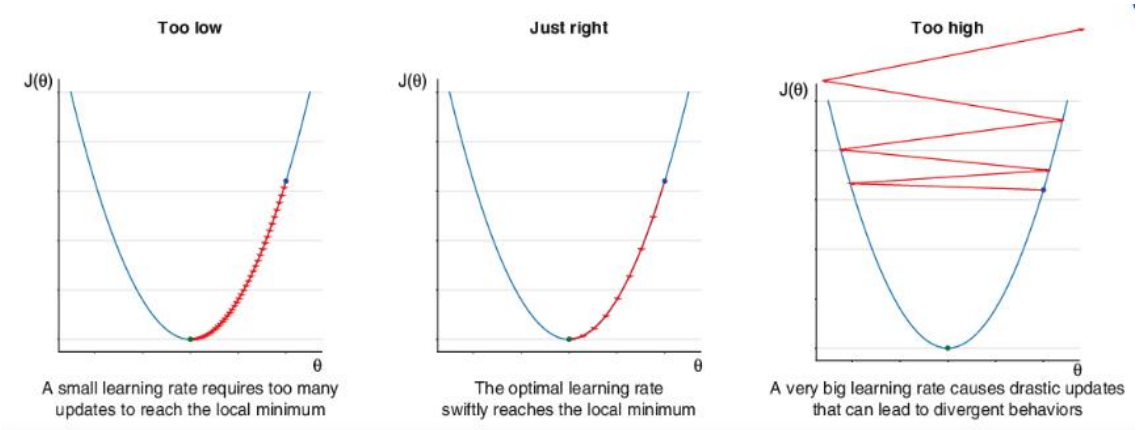

Figure 2.11: Convergence to the global minima with different learning rates [9].

The *learning rate* has a big role in the determination of the minimum of the cost function. The bigger the learning rate the faster the convergence to the minimum. However, this can cause divergent behaviour since there are 'jumps' caused by drastic updates.

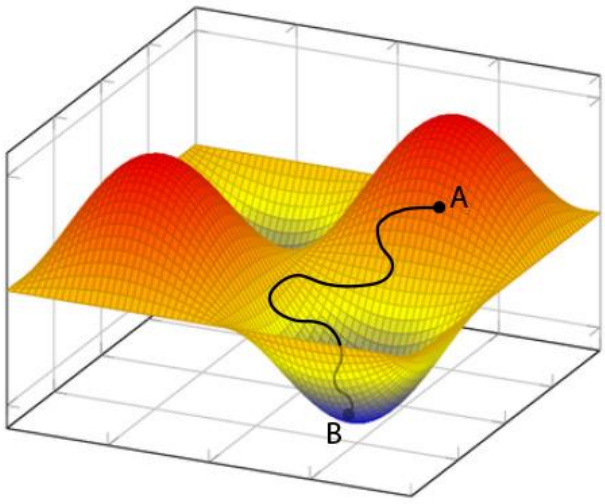

Figure 2.12: Gradient Descent 3D visualization.

On the other hand, a relatively small learning rate can slow down the convergence and so more *epochs* are needed to obtain good results.
It is possible to apply the gradient descent technique computing the gradient on the entire dataset or on a group of elements. This is called *batch* and corresponds to the number of elements used to compute the gradient at every iteration. This can influence the choice about the weights and biases and so the convergence of the loss function. There are different variants of Gradient Descent:

- ***Full Batch Gradient Descent*** computes the loss function considering all the elements in the dataset at each step. There are some disadvantages: the operation is quite long when it is present a large dataset. Another problem is its lack of dynamics: to improve the model with new data it is necessary to repeat the training process on the entire dataset.

- ***Stochastic Gradient Descent*** addresses the major issue of the *full batch gradient descent*. It uses a different approach, which selects a "random" instance of training data at each step before computing the gradient, making it significantly faster than Batch GD because there are fewer data to handle at once. It can be used for large datasets and reaches convergence much faster. On the other hand, when it gets close to the minimum point, it doesn't settle down but instead bounces around. This allows us to obtain a good result but not an optimal one.

- ***Mini Batch Gradient Descent*** is a compromise between the two techniques seen before. At each epoch, a subset of the dataset is selected, and then the loss function is calculated as the average of the loss function computed within the subset. There is no certainty that an error will converge more effectively but compared to SGD, the batching updates offer a method that is more computationally efficient.

Summing up, using the *mini batch gradient descent* and the *stochastic gradient descent* we obtain trajectories that do not go directly to the minimum of the function. This happens because in the first case we compute the gradient by taking a subset of the entire dataset, while in the second case the gradient is calculated by picking a single instance of the dataset. On the other hand, the *full batch gradient descent* measures the gradient over the complete dataset.

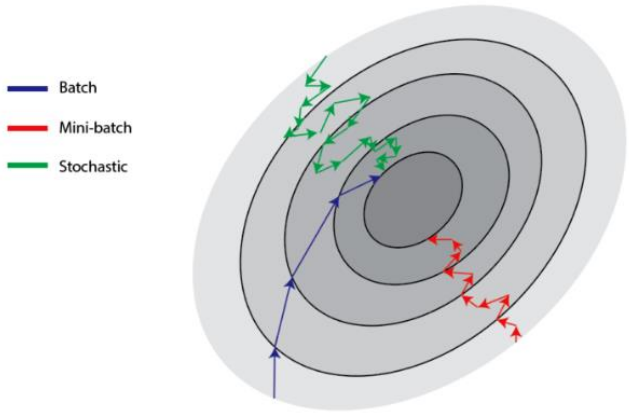

Figure 2.13: Full Batch, Mini Batch and Stochastic gradient descent trajectories [10].

2.3 Convolutional Neural Networks

In the previous chapter, we have seen how Artificial Neural Networks are built, in particular how a *fully connected neural network* is constructed. Every neuron of each layer is connected to all the neurons of the adjacent one. However, this configuration does not perform well on classification problems with images.

Let us look at an explicative example. One of the most famous is about handwritten digit classification. The dataset MNIST contains 70.000 images that are split as follows: 60.000 for training and 10.000 for testing. The images are 28 by 28 pixels, greyscale and represent digits that go from 0 to 9. When a particular image is fed as input to the neural network, it is interpreted as an array of pixels. Based on the intensity of the colour of a pixel, it is associated with a value belonging to the interval [0,255]. Therefore, a single

image of the MNIST dataset is seen from the ANN as an array of 28×28 = 784 elements. These elements are the so-called *features* of the image.

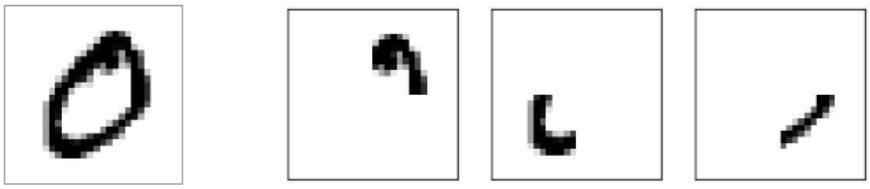

Figure 2.14: Digit present on MNIST dataset (on the left), possible features representation (on the right) [8].

The *fully connected network* encounters a problem when dealing with the images since the computational effort is very high. The extraction of features of an image is difficult, in fact, to represent an image in RGB format (3 channels) of 10×10 pixels size it is necessary to employ 3 matrices 10×10. This can be represented as a single matrix with a size of 10×10×3, also known as a tensor, with 300 nodes in the first layer. Generally, the dimension of pictures is larger than the one defined. Let us take a picture in RGB format with 256×256 dimension. It will have the first layer of 256×256×3 = 196.608 features. These neurons have to be connected to the ones of the first *hidden layer* and so there will be needed 196.608·x weights, where x stands for the nodes present in that layer. As can be observed, the use of such a neural network leads to having a node for each pixel of the image. Most of the time, pixels near one another are correlated and this spatial relationship is not maintained, as for the entire structure of the image. Pixels that are close to each other, are treated as the ones that are far away.

Since the 1980s, Convolutional Neural Networks (CNNs) have been utilised in image identification. CNNs were developed through research on the visual cortex of the brain. They have recently been able to perform unnaturally well on several challenging visual tasks as a result of the rise in computer power and the quantity of training data. They work in a good way also for tasks such as voice recognition, natural language processing and so on [6]. CNNs have been developed inspired by a peculiarity of the neurons of the visual cortex: the *local receptive field*. Neurons respond only to visual stimuli present in

a certain area of the visual field. Receptive fields may overlap, and when combined, they tile the entire visual field. CNNs are based on a new extraction method of the features. Each image is divided into different regions and the characteristic of that particular region is taken using *filters*. The presence of the term *convolutional* is because the network uses a mathematical operation called convolution.

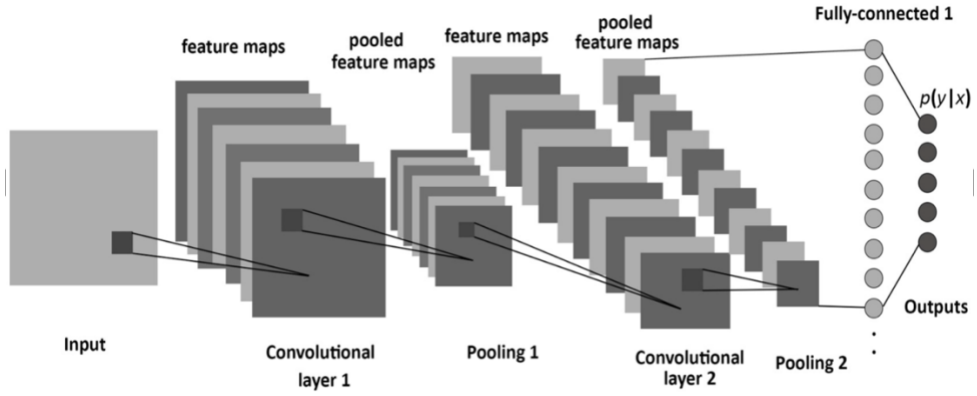

Figure 2.15: Convolutional Neural Network architecture [12].

To ensure invariance concerning translation and distortions, CNNs are based on three important aspects: *local receptive fields*, *shared weights*, and *pooling*. Before introducing the convolutional layer let us define what is a *local receptive field*.

In *fully connected* neural networks, presented in the previous paragraph, the inputs are seen as an array of values, where each value corresponds to the pixel colour intensity. In CNNs, the image can be seen as a matrix of dimension $m \times n$ that corresponds to the size of the image itself. In this case, the single entry of the matrix represents the intensity of the pixel at (*i,j*). This configuration can be clearer by looking at figure 2.16. Differently from fully connected networks, each node of the *input layer* is not connected to all the nodes of the following layer. Each neuron of the *hidden layer* will be linked to a small region of the input. The region of the input layer associated with the neuron of the hidden one is called ***local receptive field***.

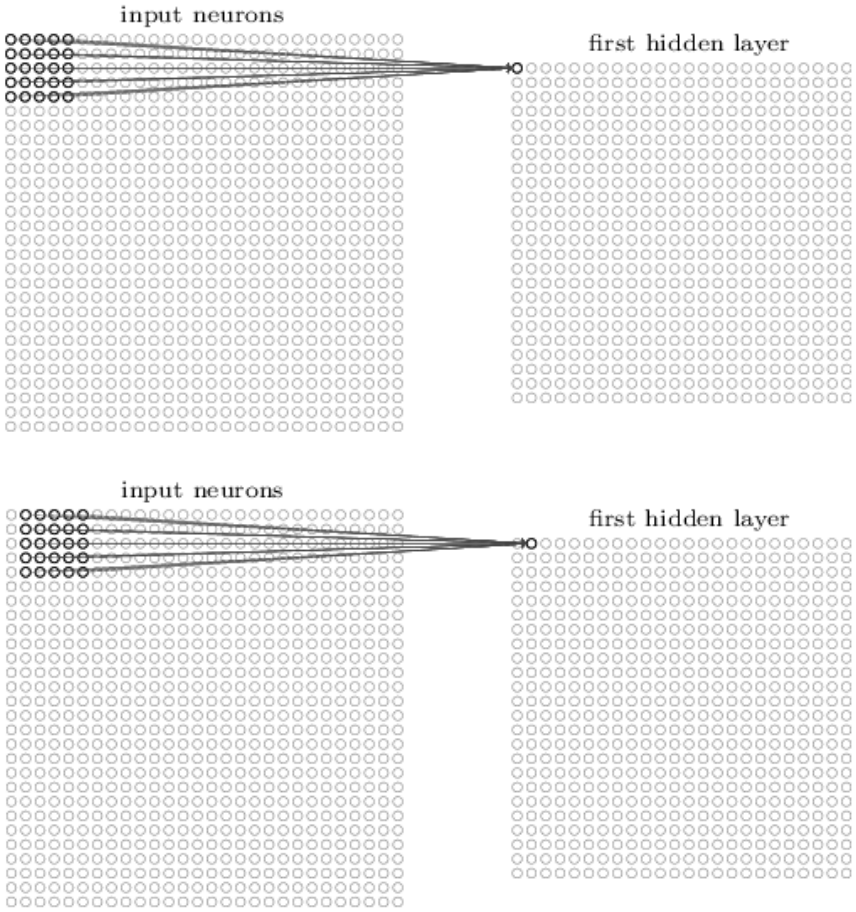

Figure 2.16: Local receptive field representation [8].

To determine the receptive field associated with the next neuron of the hidden layer, we need to let the window slide by a certain quantity *k*. This shift is called *stride*. In figure 2.16, it is possible to notice that the local receptive field of dimension 5×5 is shifted by 1 pixel with respect to the first region.

Other fundamental concepts of Convolutional Neural Networks are the ***shared weight and biases***. In the example seen above, an input image of dimension 28×28 is fed to the CNN. Then, connecting the nodes of the first hidden layer to the local receptive field, a layer of dimension 24×24 is obtained. Since all these neurons in the first hidden layer will have identical weights and biases, they will all detect the same *feature* in the input

image at various locations. The feature is the kind of pattern that triggers a neuron i.e., an edge of the image or some form. The output computed by the node at position (*j,k*) of the hidden layer is:

$$Output_{j,k} = \sigma\left(b + \sum_{l}\sum_{m} w_{l,m} a_{l+j,m+k}\right) \tag{2.15}$$

Where $a_{j,k}$ represents the input activation at position (*j,k*), $w_{l,m}$ and b are respectively the shared weights and bias and σ is the activation function. This put in evidence a fundamental characteristic of the convolutional neural networks; in fact, if a pattern is localized at a particular position of the image, it will be revealed also in another position of that picture. This property is called *translation invariance*. The map generated starting from the input layer to the hidden layer is said *feature map*. The shared weights and bias determine the *kernel* or also called *filter*. In our case, the kernel is the 5×5 region defined above. Typically, multiple *feature maps* make up a convolutional layer, hence different kernels are used. Each one of these will learn a particular characteristic of the image [8]. Moreover, the feature map dimension is decided considering three parameters: *filters*, *stride* and *zero padding.*

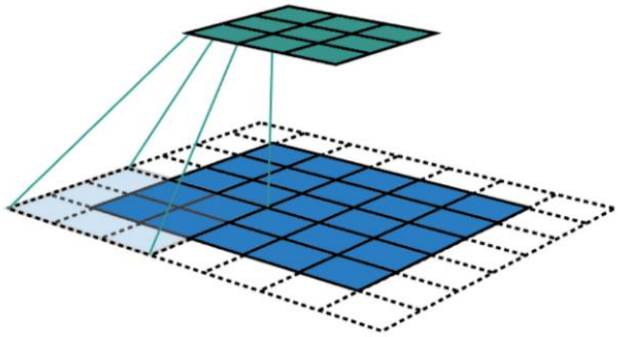

Figure 2.17: Zero padding representation [13].

The *zero padding* technique consists in adding zeros around the input layers. This allows us to obtain a layer with the same height and width as the previous one. It is a crucial strategy employed to control the output size of the layer and to preserve certain aspects of the image that should not be lost.

The last aspect that needs to be pointed out is the ***pooling layer***. This is usually inserted after a convolutional layer, and it has the objective of decreasing the image dimension. The matrix of pixels is divided into different subgroups and each of them is substituted with a value determined by a statistical function. For instance, a common procedure is *max pooling*. However, it must be taken into account that even if the pooling operation leads to a reduction in the length and width of the matrix, the depth will remain unaltered. Two different pooling techniques are commonly used: *max pooling* and *average pooling*. Let's see how each operates:

Max pooling

This is the most common *pooling* technique used. The max pooling operates using a filter of size 2×2 and a stride of 2×2, for each subset of pixels, will be extracted only the pixel having the maximum value. Looking at the first block, composed of the following elements [29,15,0,100], the max value extracted is 100.

Average pooling

In this case, the reduction of the image's dimension is obtained by performing the mean value of the elements present in one block. Considering the first block in the picture, the *average pooling* will return 36.

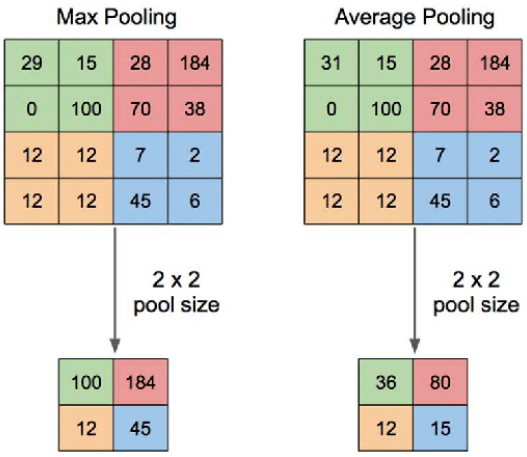

Figure 2.18: Max and Average Pooling application [14].

At the end of convolutional layers, there are *fully connected layers*. This is possible due to the *flattening* operation; in fact, a matrix of dimension W×H×D is converted into a vector W·H·D. The structure of the fully connected layers is the same described in paragraph '2.2.3'. The *output layer* of the CNN will have neurons equal to the total number of classes defined for the classification problem. The *output vector* corresponds to the probability that a specific object belongs to a certain class.

Different variants of CNN architecture have been created over time, resulting in incredible advancements in the industry. In 1998, Yann LeCun developed the most famous one, the LeNet-5 architecture. Generally, this was frequently employed for handwritten digit recognition. In 2012, the AlexNet CNN won ImageNet ILSVRC challenge. It is much deeper and larger with respect to LeNet-5, and it is made by stacking together convolutional layers instead of putting pooling layers in between.
In 2014, Christian Szegedy et al. developed GoogLeNet architecture and won the ILSVRC 2014 challenge. The main innovation is the *inception module*. Compared to AlexNet, GoogLeNet uses parameters significantly more effectively, in fact, it has ten times fewer parameters than the previous network. Other CNNs designed in the next years

are VGGNet, ResNet, Xception and SENet [6]. In the next section, there will be carried out an in-depth analysis of ResNet architecture.

2.3.1 ResNet Architecture

In 2015, Kaiming He, Shaoqing Ren, Jian Sun and Xiangyu Zhang won the ILSVRC challenge with Residual Network (ResNet). It is an extension of CNNs, made by 152 layers, 8 times deeper than VGG and with lower computational complexity. The process of training these large neural networks takes a lot of time and is very difficult. One of the problems with deep neural networks is that as the depth of the network increases, the accuracy of the model decreases. This problem is caused by the increment training error. This issue is solved by adding *skip connections* to the neural network. How does it work? The input signal entering a certain layer is also submitted to a higher layer of the stack, so it is a direct connection that jumps some layers in between.

Given an input x, the network maps x to output y employing the target function $h(x)$. If a skip connection is used, and the input is inserted at the output of the network, then, the target function will be $f(x)=h(x)-x$. This procedure is *residual learning*. Typically, the learning is significantly faster when skip connections are added and they also allow to prevent a bottleneck. Even if certain layers have not yet begun to learn, the network can nevertheless advance. The signal can easily travel the entire network because of skip connections [6]. The performance of neural networks with more layers has substantially improved because of the use of ResNet. ResNet structures provide a much lower error percentage compared to the model with the same layers. Some common ResNet architecture are: ResNet34, ResNet50, ResNet101, ResNet152, ResNet101 V2, ResNet152 V2. The number corresponds to the layers present in the model.

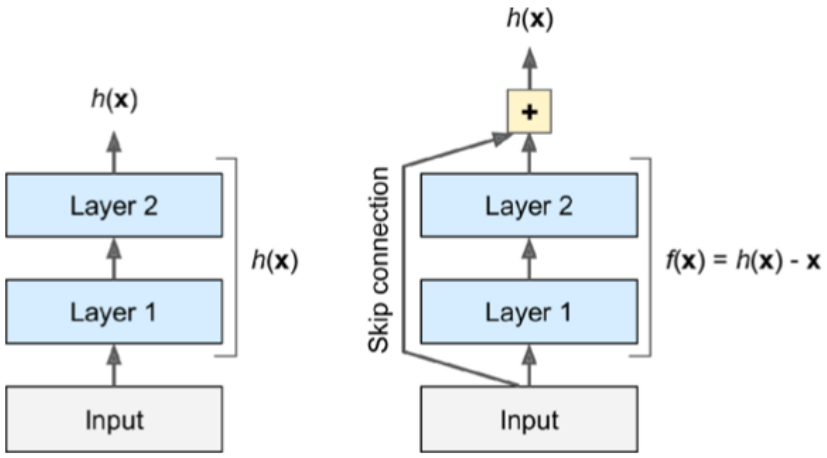

Figure 2.19: Residual learning [6].

The ResNet architecture is made by a stack of *residual units*. Each unit is composed as follows:

- 2 convolutional layers (without pooling layer).
- Batch Normalization and ReLU activation function.
- Kernel of dimension 3×3, preservation of spatial dimension that consists of stride 1 and 'same' padding.

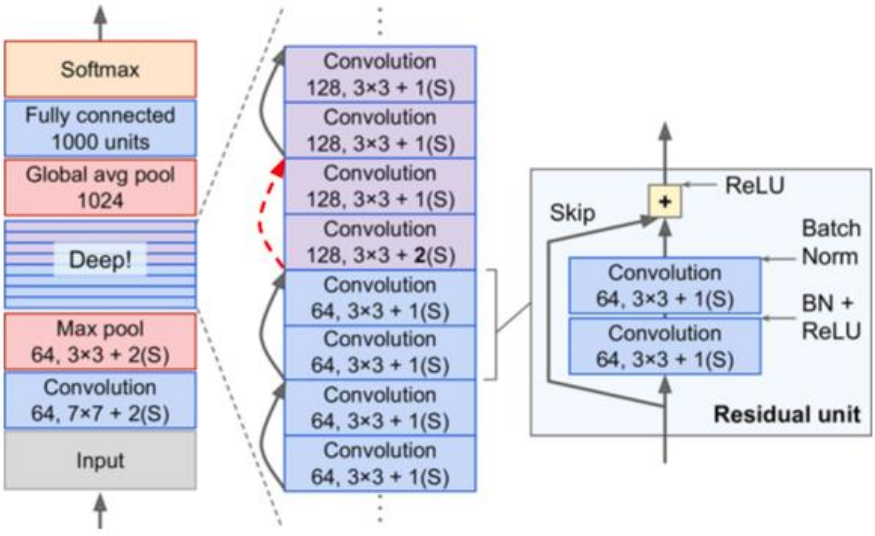

Figure 2.20: ResNet structure [6].

The risk of vanishing and exploding gradients issues during training is considerably reduced by the *Batch Normalization* technique. The method entails inserting an operation into the model just before or after each hidden layer's activation function. Using two new parameter vectors per layer—one for scaling and the other for shifting—this operation simply zero-centres, normalizes, then scales and shifts each input with the two parameter vectors. In other words, the operation enables the model to discover the ideal scale and mean for each input of the layer. In many circumstances, standardizing your training set is unnecessary if you include a BN layer as the initial layer of your neural network. The method must determine the mean and standard deviation of each input before zero-centring and normalizing them. It accomplishes this by calculating the input's mean and standard deviation for the most recent mini-batch. During deep neural network training, batch normalization can have a negative effect on training if batches are small, so this layer is often frozen and functions as a linear layer.

Chapter 3

Mask R-CNN

Convolutional neural networks are mostly utilised for image processing, and among the topics addressed by them, it is possible to mention *object detection*, *semantic segmentation*, and *instance segmentation*. The tasks introduced by classification: object detection, and semantic segmentation, are included in the definition of instance segmentation.

An *instance segmentation* problem for an image is the process of separating each individual object from other instances in the picture after it has been correctly detected (object detection) and categorised (also for objects belonging to the same class). This final feature enables the distinction between a *semantic segmentation* problem and an *instance segmentation* problem. A semantic segmentation problem assigns the same value to all predetermined instances belonging to the same class.

There are innumerable applications of *instance segmentation* that space from segmentation of objects to detection of medical diseases.

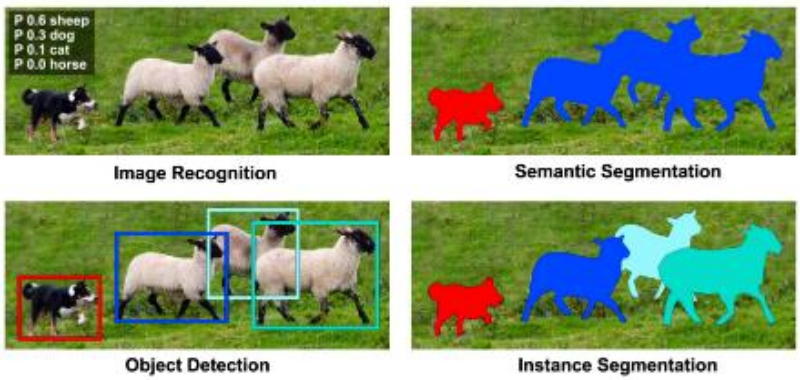

Figure 3.1: Representation of image classification (top left), object detection (bottom left), semantic segmentation (top right), and instance segmentation (bottom right) [15].

Segmentation is extremely crucial in various applications and during these years various algorithms have been developed. In 2014, Ross Girshick et al. developed a *Region-Based Convolutional Neural Network* (R-CNN) that set a real milestone in object detection problems. This approach improved the *mean average precision* score (*mAP*) by more than 30% over the latest best result. Over the years, R-CNN was greatly improved by introducing Fast R-CNN and Faster R-CNN, which outperformed the previous neural network in terms of performance and accuracy. The Mask R-CNN model improves the Faster R-CNN by adding a feature that allows the neural network to segment single instances of the image. This network will be used to carry out the instance segmentation of floor plans.

3.1 Region-based CNN

Before explaining the architecture used for the project (*Mask R-CNN*), it is worth highlighting the major aspects of *region-based* networks. It has been developed for object detection problems. This is separated into two different tasks: localization and classification. The first gives information about the position and the second about the class. The process of object detection for R-CNN is divided into three parts: *region proposal*, *feature extraction*, and *classification*.

- ***Region proposal*** algorithm (*selective search*) clusters regions depending on pixel intensity. Therefore, it organises pixels into groups based on the hierarchical grouping of related pixels. The authors extracted 2000 *regions of interest* (*RoI*) in the original paper.

- ***Feature extraction*** is the process actuated by a large convolutional neural network. It takes out a feature vector of size 4096 from every single region proposal. In order to be compatible with the CNN architecture, the image data must be modified, in fact, it is necessary to have a 227×227 pixel dimension.

- ***Region/Object classifier*** needs to categorize positive and negative instances and introduce an IoU overlap threshold. The region that partially overlaps the object is classified as a negative example if the IoU is below the defined threshold. In the original architecture, it has been selected as 0.3 after a grid search over an interval on the validation set. Finally, an SVM is applied for classification and linear regression to shrink the bounding boxes of the object [16].

The primary issue with R-CNN is the high cost of computation: each proposed region of an image must be classified by a convolutional network afterwards, which is a very expensive procedure because it must be repeated for each individual region extracted.

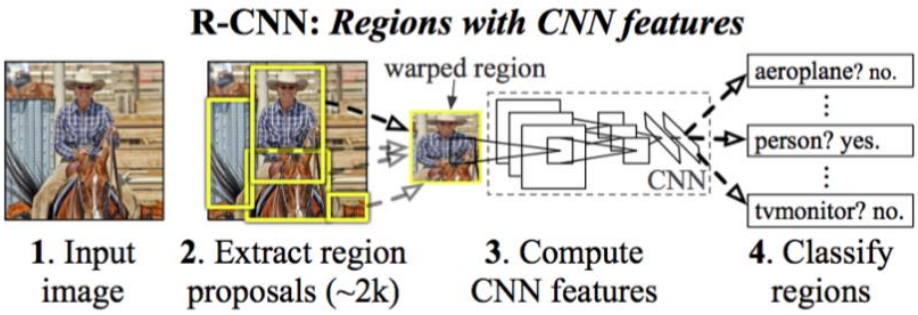

Figure 3.2: R-CNN functionality [16].

Furthermore, no information will be shared during the individual training of each bounding box regressor. R-CNN employs several SVMs, each of which must be trained separately during the training phase because the number of SVMs used by R-CNN is equal to the number of classes.

3.2 Fast R-CNN

The main problem related to R-CNN is the amount of time consumed during training, the storage and computation power. Moreover, a complicated multi-stage training process is present. In R-CNN the *selective search* produces 2000 region proposals for every image and each region is submitted to the network. This implies that there are 2000 forward

passes for every single image. Taking into consideration a dataset with a great number of pictures there would be a huge number of forward passes. In 2015, Fast R-CNN was developed by Ross Girshick, and the neural network successfully resolved these problems being approximately 9 times quicker during training and 146 times faster during the testing with respect to R-CNN. We've seen in R-CNN that the potential regions of interest are initially outlined and then classified by a CNN. In Fast R-CNN, the two operations are "inverted": establishing regions of interest is not the first action taken by the network, but rather follows the final feature map established by the network. Essentially, in Fast R-CNN, the initial convolutional layers help to draw attention to the image's significant details and facilitate the succeeding feature selection step, which improves the identification of the regions of interest. Additionally, classification is now carried out by a single softmax classifier rather than an SVM for each class. The performances obtained by Fast R-CNN are significantly improved with respect to R-CNN, in fact, accuracy and training time are superior.

On the other hand, the issue of selecting relevant regions of interest has not yet been fully resolved because the *Selective Search* algorithm is still slow and continues to display an excessive number of inappropriate *RoI*.

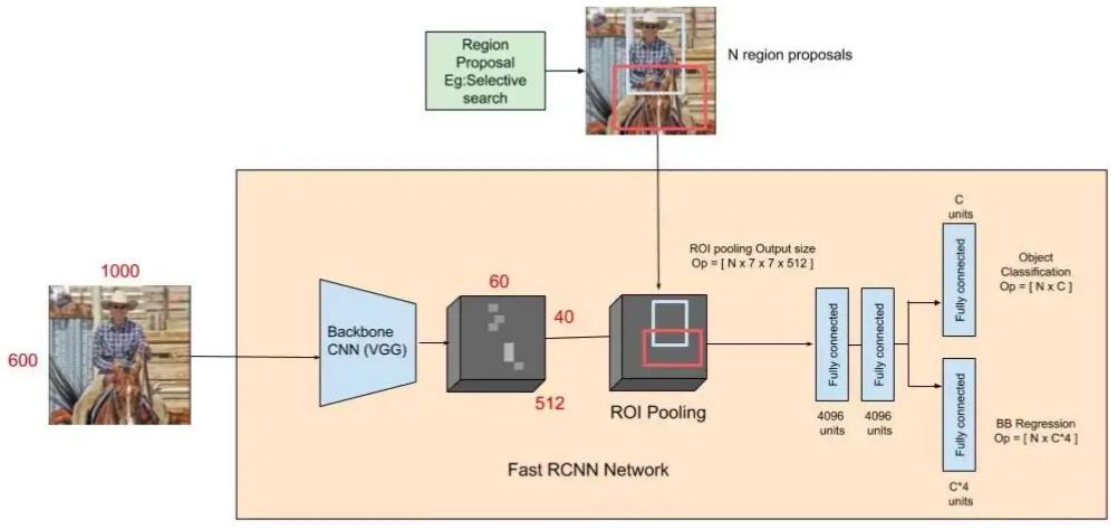

Figure 3.3: Fast R-CNN architecture [17].

Fast R-CNN has two distinct output layers: the first returns a discrete probability distribution $p = (p_0, \ldots, p_k)$ on $K + 1$ categories, where p is calculated using a softmax function on the $K + 1$ outputs of the fully-connected layer, and the second returns the bounding box coordinates $t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$ determined by the regressor. The loss function associated to each *RoI* is:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{bbox}(t^u, v) \tag{3.1}$$

The loss function related to the classifier is:

$$L_{cls}(p, u) = -logp_u \tag{3.2}$$

where the probability that an object belongs to class u is defined by p_u. The second term of equation 3.1 represents the loss function calculated over the predicted bounding box ($t^k = (t_x^k, t_y^k, t_w^k, t_h^k)$) with respect to the target one of the class u. The term $\lambda[u \geq 1]$ is 1 when $u \geq 1$ and 0 on the contrary. The background class is targeted with $u = 0$. The regressor loss function is:

$$L_{bbox}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^u - v_i) \tag{3.3}$$

where the $smooth_{L1}$ function is:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2, & if\ |x| < 1 \\ |x| - 0.5, & if\ |x| \geq 1 \end{cases} \tag{3.4}$$

The hyper-parameter λ in Eq. 3.1 regulates how the two task losses are balanced [18].

3.3 Faster R-CNN

The fundamental concept of Faster R-CNN is to improve Fast R-CNN performance by modifying the region-selection procedure. Since earlier versions used an external *selective search* algorithm, which is not a true component of the neural network, this method could not be trained. Fast R-CNN initial convolutional layers were created to highlight the map's properties and enable the development of regions of interest: Faster R-CNN replaces the Selective Search method entirely by adding extra layers that allow the division of *RoI* using the actual and legitimate convolutional network known as *Region Proposal Network* (*RPN*). To create a unified network that can be trained end-to-end, Faster R-CNN introduces Region Proposal Network, a deep learning technique that generates object proposals while also sharing the convolutional layers of the Fast R-CNN detector [19].

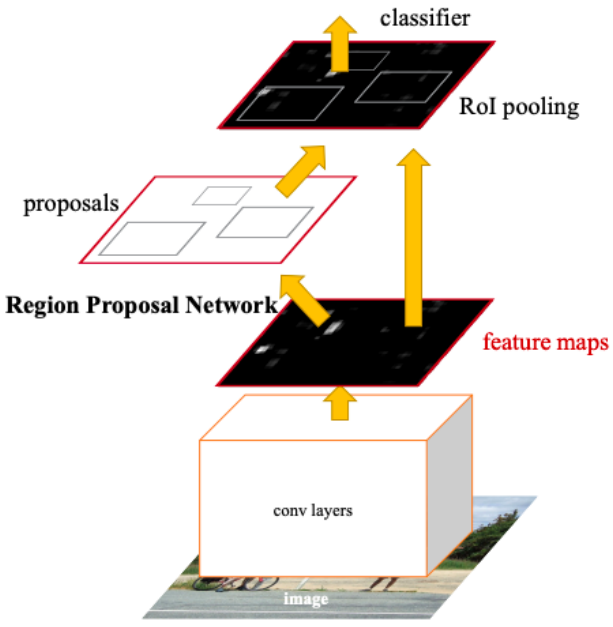

Figure 3.4: Region Proposal Network module [19].

Region Proposal Network. As it is explained in the original paper [19], the Region Proposal Network (RPN) network takes an image as input and produces a list of suggested regions, each with a probability of actually finding the object there. The term *region* refers to a rectangular-shaped area where an object can be located in the network. This network must determine for every point of the output *feature map* if an object is present in the input picture at its corresponding location and estimate its size. Once arrived at the last layer of the first CNN within Faster R-CNN, a *sliding window* of dimension 3×3 is employed to scan the *feature map* and so to determine the boundaries of the region proposal, also known as *anchors*. When the sliding window is moved along the feature map, more than k anchors are identified by varying the *scale* and *aspect ratio*. By default, there are three distinct scale values (128, 256, 512) and three different aspect ratios (1:1, 1:2, 2:1), adding up to a total of 9 anchors for each sliding window. Each of these regions will have a probability associated with it (a score and the specific anchor box coordinates). Therefore, there will be $W \cdot H \cdot k$ anchor boxes for a feature map of dimension $W \times H$.

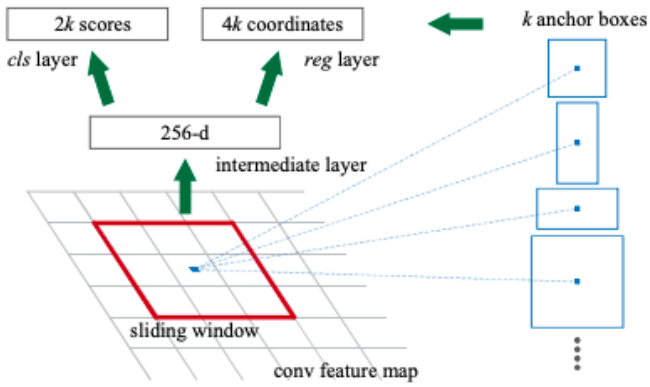

Figure 3.5: Sliding window and anchors in Region Proposal Network [19].

W and H can also be determined by using the formulas $W = {}^{w}/_{r}$ and $H = {}^{h}/_{r}$, where w and h are the width and height of the original input image, while r is the subsampling ratio used in the backbone. Once the region proposals have been created, the network will elaborate them, in fact, the next layers are identical to the ones present in Fast R-CNN:

Pooling layer, Softmax classifier, and bounding box regressor. During the training of RPNs, each anchor is assigned a binary class label to identify if there is an object or not. One of the two following conditions must be satisfied to classify an anchor as a "positive" sample:

I. The anchor has the highest Intersection over Union with a ground-truth box.

II. The anchor has an IoU value bigger than 0.7 with any ground-truth box. It happens that the same ground-truth box can cause multiple anchors to be assigned with positive labels.

The anchors that obtain an IoU with all the ground-truth boxes below the threshold of 0.3 are labelled as "negative". The remaining anchors (IoU belonging to interval [0.3, 0.7]) are not used for RPN training.

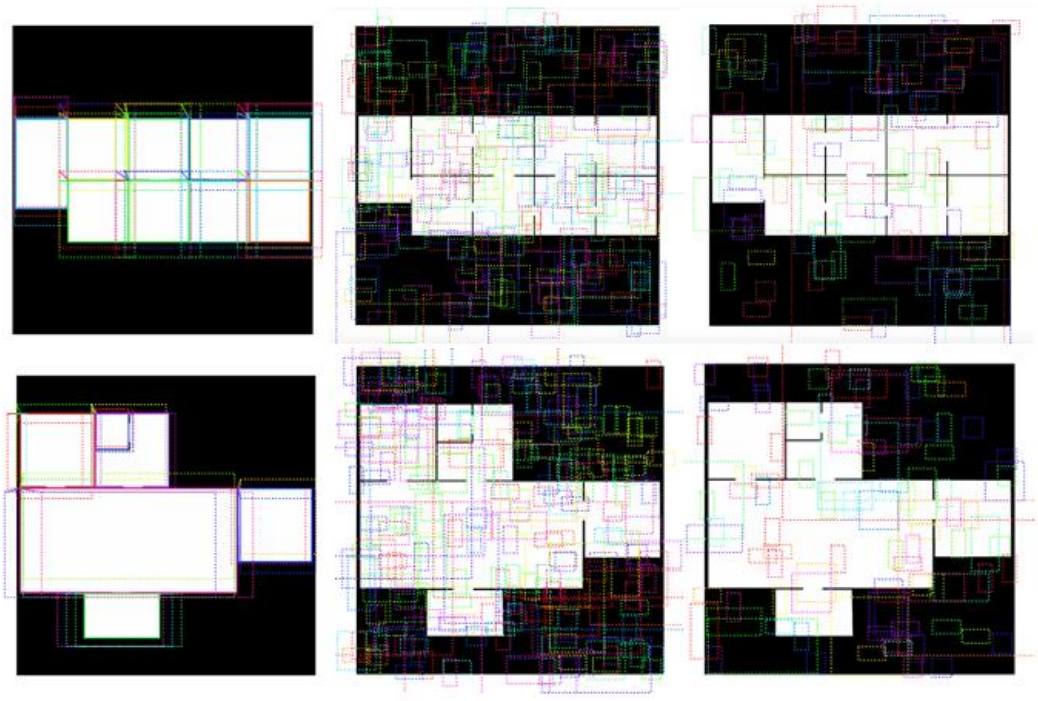

Figure 3.6: Example of anchors selection process: positive (left), negative (middle), and neutral (right).

The loss function related to each region of interest is:

$$L = L_{cls} + L_{bbox} \tag{3.5}$$

This differs from the loss function used in Fast R-CNN due to the addition of normalization terms. This can be rewritten as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{bbox}}\sum_i p_i^* \cdot smooth_{L1}(t_i - t_i^*) \tag{3.6}$$

where L_{cls} is the log-loss calculated over two classes (object and not object), while the L_{bbox} is the same used in Fast R-CNN. The parameters of equation 3.6 are:

- $\boldsymbol{i}$, index of an anchor in the mini batch.
- $\boldsymbol{p_i}$, predicted probability that the anchor *I* is an object.
- $\boldsymbol{p_i^*}$, ground-truth label (1 for positive anchor, 0 for negative one).
- $\boldsymbol{t_i}$, predicted bounding box vector of coordinates.
- $\boldsymbol{t_i^*}$, ground-truth box coordinates of a positive anchor.

The terms N_{cls}, N_{bbox}, and λ are respectively the normalization parameters of L_{cls} and L_{bbox}. The last is a weight that acts as a balancing parameter of the two losses. The regression loss L_{bbox} is activated only if an object is contained in the anchor [19].

3.4 Mask R-CNN

In 2017, Mask R-CNN, an extension of Faster R-CNN has been developed by Kaiming He et al. for *instance segmentation* problems. It adds a third branch that makes it possible to predict an instance's mask. This operation runs concurrently with the two branches that are already present within Faster R-CNN (classifier and bounding boxes regressor). To be more specific, Mask R-CNN will produce a 28 by 28 pixel matrix for each Region of

Interest (*RoI*), which will then be expanded to meet the bounding box's dimensions. For each obtained instance, Mask R-CNN will return the class of the object, bounding box, and a binary mask overlapped on the object. Today, Mask R-CNN continues to be one of the best architectures for instance segmentation.

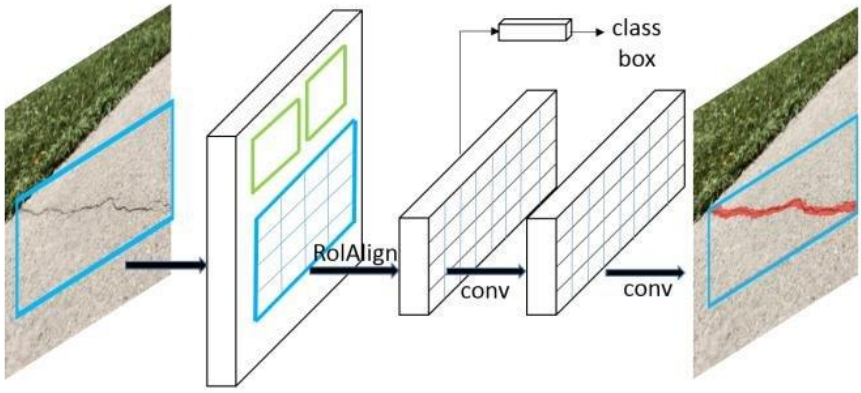

Figure 3.7: Mask R-CNN framework for instance segmentation [21].

The difficulty of instance segmentation comes from the need to accurately recognize all objects in an image while also correctly segmenting each instance. The newly added branch is a *Fully Connected Network* (*FCN*) applied to each region of interest to forecast a segmentation mask pixel-by-pixel [20]. In the original paper, different *backbone* architectures for feature extraction have been evaluated. These include ResNet and ResNeXt, both of which have 50 or 101 layers and whose features are extracted from the last convolutional layer belonging to the fourth or fifth block. For instance, a backbone ResNet-101-C4 indicates a ResNet with a depth of 101 layers and feature extraction starting from the last convolutional layer belonging to the fourth block. The ResNet architecture has been explained more deeply in chapter 2.3.1. A more effective backbone designed by Tsung-Yi Lin et al. [22] is added after the first backbone to obtain more accurate features. This network is called *Feature Pyramid Network* (*FPN*). It has been demonstrated that the ResNet-FPN approach has allowed an excellent gain in terms of accuracy and speed of feature extraction.

3.4.1 Feature Pyramid Network

In object detection tasks, finding objects at different scales can be difficult especially when small items are involved. In CNN deep layers we have low spatial resolution despite having semantically strong information. The spatial resolution is present in the first layers that contain local information but with semantically low features. We can use the same image with downscaled size to recognize objects but processing this pyramid of images is time consuming and requires large memory. As an alternative, it can be built a pyramid of feature maps. On the other hand, feature maps that are near the image layer are made up of low-level structures that are ineffective for precise object detection. *Feature Pyramid Network* (FPN) addresses this feature extraction issue allowing us to generate feature map layers with improved quality information. The FPN proposed is a network which combines layers in a bottom-up and top-down pathway. The bottom-up path is basically the same as a normal CNN's feed forward computation. Going up in the pyramid the spatial resolution decreases by ½ and the semantic value for each layer augments. FPN offers a top-down path for building higher resolution layers out of a semantically rich layer. Following the top-down path, the layers are upsampled by 2 using a Nearest Neighbors upsampling. Even though the reconstructed layers have strong semantic properties, all the upsampling and downsampling have left the object placements imprecise. To improve the detector's ability to predict objects' position, we add lateral links between the reconstructed layers and the corresponding feature maps. Moreover, it serves as a *skip connection* to simplify training (similar to what ResNet does). In the end, a 3×3 convolution is applied to the merged layers so that it reduces the anti-aliasing effect given by upsampling [22].

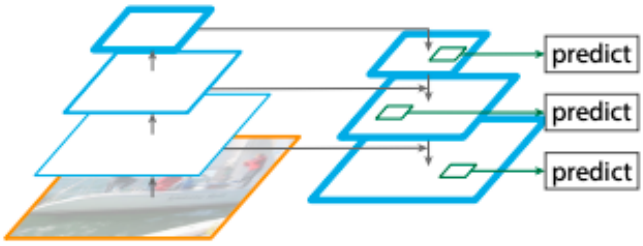

Figure 3.8: Feature Pyramid Network pathways [22].

3.4.2 RoI Align

A significant innovation introduced by Mask R-CNN is the use of RoI Align in place of RoI Pooling for determining the RoI of a feature map. RoI pooling uses regression to determine the position of the candidate boxes, and it often yields floating-point integers. The RoI pooling operation then has two quantization steps since the size of the feature map needs to be adjusted. This causes a misalignment between the original picture pixels and the feature map. Suppose we have an image of dimension 512×512 and an associated feature map of size 100×100. We want to extract a RoI of 40×40 size of the original image. Consequently, a region of pixels must be extracted from the feature map having dimension $n \times n$. The parameter n is computed in the following way:

$$n = \frac{h_{fm} \cdot h_{RoI}}{h_{oi}} = \frac{100 \cdot 40}{512} \approx 7.81$$

where h_{fm}, h_{RoI}, and h_{oi} represent respectively the size of the feature map, region of interest, and original image. Let's suppose to use RoI Pooling, extracting a region 7.81×7.81 in size will not be achievable, and an approximation region of 7×7 will be taken into consideration (the integer part of n). This approximation results in a loss of 0.81 pixels, and so there will be a loss of information since the pooling techniques will not take into consideration the lost pixels. This pixel misalignment problem has been solved with the introduction of the *RoI Align* layer, which does not approximate anymore the dimension of the extracted regions. It is used *bilinear interpolation* to compute the

exact values of the input features. Returning to the previous example, the region extracted has a dimension of 7.81×7.81 using the RoI Align technique.

3.4.3 Loss function

The three components that make up the total loss function in Mask R-CNN are the classification loss of candidate boxes, the bounding box regression loss, and the mask loss. The loss function related to each RoI extracted is:

$$L = L_{cls} + L_{bbox} + L_{mask} \tag{3.7}$$

The first two elements on the right side of equation 3.7 are the loss functions related to the classifier and regressor of the bounding boxes that we have already seen in Faster R-CNN. Regarding L_{mask} computation, each RoI is associated with a single ground-truth mask and a sigmoid activation function is applied to each pixel of the mask. For each of the potential K classes, the branch related to mask prediction will produce binary masks with dimensions $n \times n$. Consequently, will provide $K \cdot n^2$ total possible masks, each linked to a separate class. We only apply a per-pixel sigmoid on the k_{th} mask if the ground truth class is *k*. This allows us to define L_{mask} as the average binary cross-entropy loss.

$$L_{mask} = -\frac{1}{m^2} \sum_{1 \le i,j \le m} [y_{ij} log \hat{y}_{ij}^{k} + (1 - y_{ij}) \log (1 - \hat{y}_{ij}^{k})] \tag{3.8}$$

3.5 Evaluation Metrics

As an extension of object detection, *instance segmentation* creates a binary mask for each identified object in addition to localizing it. Instance segmentation algorithms can be divided into two major categories: *single-shot instance segmentation* techniques and *detection base instance segmentation* algorithms. The second method produces masks of greater quality, but the first method is faster. In the next chapter, there will be presented

different instance segmentation algorithms. In this paragraph, we will focus on the metrics used to evaluate the instance segmentation task. Firstly, it will be explained the IoU concept, then it will be exploited the notion of True Positive, True Negative, False Positive, and False Negative for determining Precision and Recall. Finally, it will be presented the COCO evaluation metrics.

3.5.1 Intersection over Union

This metric (*IoU*) is the primary evaluation criterion for measuring the quality of the mask. It is also referred to as the Jaccard index, and it is used to quantify the overlap percentage between the real mask (*target*) and the predicted mask. It is computed as the division between the intersection area and the union area of two bounding boxes (*real and predicted*), in the case of instance segmentation, it computes the intersection and union area between two masks.

$$IoU = \frac{M_t \cap M_p}{M_t \cup M_p} \tag{3.9}$$

Where M_t and M_p are respectively the target and predicted mask. The calculation of the IoU of two masks measures the number of pixels shared between the two masks divided by the total number of pixels present in both masks.

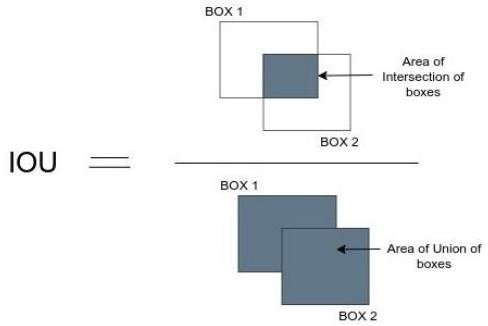

Figure 3.9: Graphical representation of Intersection over Union [23].

The IoU can assume values that are between 0 and 1 (the highest attainable value). It is 1 when the ground truth and predicted mask overlap each other entirely. In contrast, it assumes a 0 value in the opposite scenario. It is important to note that the IoU is always equal to 0 when the expected mask and the ground truth do not overlap. Therefore, if there is no overlap, regardless of how near or how far apart two masks are from one another, the IoU is 0.

3.5.2 Precision and Recall

Mask R-CNN must have the ability to accurately categorize the identified instance in addition to determining the bounding box and mask of each object. More generally, in a classification problem, it is necessary to determine whether an input *x* with an output *y*, is correctly classified by the model. In our case, it will be evaluated if the predicted mask matches the available target mask, given a particular input. It is possible to distinguish the following cases:

- ***True Positive***: the object is classified correctly by the model as belonging to a class *c*.
- ***True Negative***: the object is classified correctly by the model as not belonging to a class *c*.
- ***False Positive***: is an instance that is incorrectly classified by the model as belonging to a class *c*.
- ***False Negative***: is an instance that is incorrectly classified by the model as not belonging to a class *c*.

It is possible to define ***Precision*** as:

$$Precision = \frac{TP}{TP + FP} \tag{3.10}$$

The precision metric determines the accuracy of the model predictions. It represents the percentage of positives correctly classified by the model. For instance, of all objects detected in a picture, how many of these correspond to the ground truth annotation? Analogously, ***Recall*** is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{3.11}$$

It represents, among all the elements belonging to a certain class, the percentage of positives classified correctly relative to the ground truth. Furthermore, to determine precision and recall we need to compute the IoU score over every prediction-target mask pair and define which of these pairs has an IoU bigger than a certain threshold. Nevertheless, the behaviour of our model's *precision-recall curve* is not well captured by calculating a single precision and recall score at the designated IoU threshold. Instead, we can successfully integrate the region under a precision-recall curve using *average precision*.

3.5.3 Average Precision

Precision and recall alone are not enough to determine a classifier's reliability. This is because the increase of one of the two metrics corresponds to the decrease of the other, and vice versa. The two metrics are therefore evaluated in a precision-recall curve that has values for recall on the horizontal axis and values for precision on the vertical axis to appropriately depict the model's classification capability. The area subtended by the precision-recall curve is called *average precision* (*AP*) and will assume a value between 0 and 1. Let us define *p* as a function that calculates the *precision* at a certain *recall* value, the *average precision* is:

$$AP = \int_0^1 p(r)dr \tag{3.12}$$

The *AP* is calculated using an approximation through interpolation to simplify the integral calculation. In 2008, the Pascal VOC2008 Challenge offered to calculate it as the average of the maximum precision values calculated for 11 standard recall levels (0, 0.1, 0.2, …, 1.0) [24], specifically:

$$AP = \frac{1}{11}\left(AP_r(0) + AP_r(0.1) + \cdots + AP_r(1.0)\right) \tag{3.13}$$

It can be reformulated in the following way:

$$AP = \frac{1}{11}\sum_{r=0}^{11} AP_r = \frac{1}{11}\sum_{r=0}^{11} p_{interp}(r) \tag{3.14}$$

Where:

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{3.15}$$

However, this way of calculating the *AP* has two problems. It is not precise, and it is not able to evaluate models with low *AP*. In later Pascal VOC Challenges (2010-2012), it has been modified the method for AP computation [24]. In fact, it is not computed as in the 3.14 equation, but the graph is sampled at different recall values when the maximum precision decreases. The mathematical formulation of *AP* changes as follows:

$$AP = \sum (r_{n+1} - r_n)\, p_{interp}(r_{n+1}) \tag{3.16}$$

Where:

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} p(\tilde{r}) \tag{3.17}$$

The *AP* computation is performed only when there is only a class involved. However, in object detection problems are present more classes $K>1$. In this case, is computed the mean average precision as the mean value of *AP* among all classes:

$$mAP = \frac{\sum_{i=1}^{K} AP_i}{K} \tag{3.18}$$

3.5.4 COCO Metrics

The *mAP* metric used in the Pascal VOC challenge can be used as a reference to assess how well object detectors perform. Over the years, other metrics have been introduced in order to test a model. In particular, during the COCO challenge, is computed the *mAP* using different IoU thresholds and different object sizes.

```
Average Precision (AP):
  AP                % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50        % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75        % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small          % AP for small objects: area < 32^2
  AP^medium         % AP for medium objects: 32^2 < area < 96^2
  AP^large          % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1          % AR given 1 detection per image
  AR^max=10         % AR given 10 detections per image
  AR^max=100        % AR given 100 detections per image
AR Across Scales:
  AR^small          % AR for small objects: area < 32^2
  AR^medium         % AR for medium objects: 32^2 < area < 96^2
  AR^large          % AR for large objects: area > 96^2
```

Figure 3.10: COCO metrics used for evaluation [25].

As we can see from figure 3.10, the acronyms *AP* and *AR* correspond to the mean average precision and mean average recall obtained on all the classes. $AP^{IoU=.50}$ and $AP^{IoU=.75}$ stand for the *mAP* computed considering only the instances that have IoU greater than 0.50 and 0.75. A new metric has been introduced to evaluate the model performance. It is the *AP* calculated at various IoU thresholds belonging to this interval [0.50: 0.05: 0.95]. The *AP*

is evaluated over the entire dataset test with IoU = 0.50, then with IoU = 0.55 till IoU = 0.95. In the end, it is performed the mean between all *AP* values, and it represents the *Average Precision* over 10 IoU thresholds. The *AP Across Scales* is computed taking into consideration the dimension of the object detected. In particular, AP^{small} is computed considering only the instances that have an area smaller than 32^2, AP^{medium} is obtained for objects that have an area belonging to the interval [32^2, 96^2], and AP^{large} is calculated for objects that have an area bigger than 96^2. The area corresponds to the pixels present in segmentation masks. The *Average Recall* (*AR*) is computed considering a fixed number of detections for a single image. For instance, $AR^{max=1}$ is the average recall calculated over all the categories considering a single detection per image. The detections are 10 and 100 for $AR^{max=10}$ and $AR^{max=100}$. Finally, *AR Across Scales* is the same as *AP Across Scales* but with different sizes for the instances detected.

Chapter 4

Implementation

This part of the thesis project presents the software implementation used for accomplishing the objective of segmenting rooms in floor plan images. Before diving into the implementation of this network, the first part of the work has been spent learning information and concepts about Machine Learning, Deep Learning, and instance segmentation task. This has been done to have a strong background for handling the actual problem. All these notions are partially reported in the first part of this work. In this second part, there will be explained the tools employed for implementing the neural network. The segmentation task is performed using a suitable deep learning model, in particular, a Mask R-CNN model is employed. The implementation is based on the Matterport repository [26], it is written in Python, and it is adapted to our *instance segmentation* problem. The TensorFlow framework is used, and to practice with this tool simple models have been implemented to understand the functionality, data preparation and splitting, training, and testing. This chapter is divided into three parts, where the first is an in-depth analysis of the segmentation task, the second explains how the dataset has been constructed and the third is about the training process of the model.

4.1 Instance Segmentation

As stated in the previous chapter, instance segmentation is an extension of object detection where a binary mask is created for each individual instance in addition to object localization and classification. In the following part are presented the three main segmentation techniques:

- *Semantic segmentation* aims to categorize picture pixels into a collection of groups without distinguishing between distinct object instances. It divides up

every object in the image that belongs to the same class but does not distinguish between various examples of the same class. For instance, in medical scans, we can identify every cancer cell, but we cannot differentiate one cancer cell from another.

- *Instance segmentation* distinguishes every instance in the visual input that is a component of the same class. It combines two tasks: accurately segmenting each instance while also properly detecting each object in the scene. It can be interpreted as semantic segmentation and object detection together. In medical scans, compared to semantic segmentation, we can precisely predict each cell's shape and discern one cell from another.

- *Panoptic segmentation* connects the two previous approaches: instance and semantic segmentation. On a given image, it does semantic and instance segmentation, connecting the results of the two to create a single image. Intelligent systems should be able to comprehend the visual scene at both class instance level and at the pixel level. Scenes are categorized into *stuff* and *things* using panoptic segmentation. *Things* are specific examples of the foreground classes, such as pedestrians, automobiles, cancer cells, etc. *Stuff* is, for instance, a sky or a sidewalk – generally speaking, some background class. Panoptic FPN and EfficentPS are two examples of network designs.

Instance segmentation is the technique that is used for achieving our thesis objective. The handling of occluded objects of the same class presents the main obstacle in the instance segmentation problem. To put it another way, instance segmentation must correctly assign pixels to the same class, and distinguish between distinct instances and separate instances that overlap. The key is to perform segmentation on bounding boxes rather than processing the entire image is the main concept to assist the neural network in completing that operation. These solutions are referred to as *two-stage instance segmentation*

algorithms. *Mask-RCNN* is one example of a two-stage instance segmentation technique (or *detection base* segmentation algorithm). There are neural networks, such as *PolarMask++* or *YOLACT*, that can do *one-shot instance segmentation* tasks.

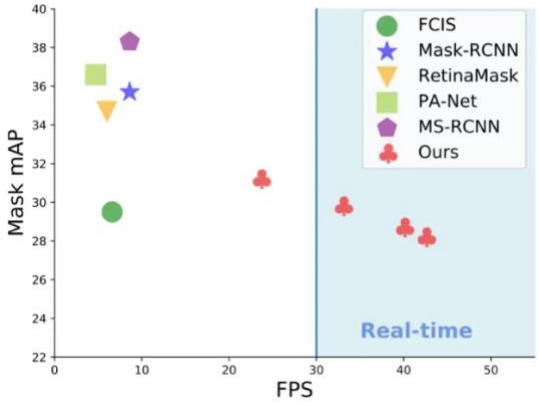

Figure 4.1: Instance segmentation methods: two-stage and one-shot approaches (the red sign represents the *YOLACT* technique) [27].

One-shot refers to the network's simultaneous processing of object classification and object segmentation. On the other hand, two-stage instance segmentation algorithms perform segmentation head-on object proposals having initially detected objects with bounding boxes. Methods based on one-shot segmentation are faster than two-stage techniques, but their results are worse in terms of Average Precision.

4.2 Dataset description

In this paragraph, it will be presented the dataset used for instance segmentation of floor plans, but before getting started the main datasets for image segmentation are presented. Although instance segmentation is the focus of the thesis, other datasets primarily employed in image classification are also examined. This is crucial since several model modules utilized in the work are trained or pretrained using datasets unrelated to the main task.

4.2.1 Datasets overview

In paragraph 2.3, a little overview of the ***MNIST*** dataset has been made. It is made of handwritten digits that go from 0 to 9 and it contains 70.000 images which are divided into 60.000 for training and 10.000 for validation. The acronym MNIST stands for Modified National Institute of Standards and Technology database and each image present in this set is a greyscale picture of 28×28 pixels in size. MNIST is an extremely popular dataset that is suitable for classification issues with 10 classes. The primary use of this dataset is as a first test to determine whether a model performs properly. A model must perform well on the MNIST, but this is not enough to ensure excellent network behaviour in more complex contexts. The fashion MNIST dataset contains 10 different kinds of wearing clothes. It was necessary to introduce the ***FMNIST*** because models began to perform too well for the MNIST, consistently obtaining accuracy levels higher than 99%.

Another well-known dataset for computer vision tasks is the ***VOC***, which has images for classification, segmentation, recognition, and detection. Essentially, most of the models are tested on this dataset. In terms of segmentation, there are 21 classes total, plus a background class. There are 1464 photos for training and 1449 for validation. It is present an extension of this dataset (*PASCAL Context*) which contains 400 classes and compared to VOC, it is more oriented on segmentation problems.

The biggest dataset in terms of images is represented by ***ImageNet***, it is made up of more than 14 million photos that are split into more than 20.000 categories with the aim of recognizing objects. This database serves as a crucial building block for CNN and computer vision training. There is an annual competition called the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) whose goal is to determine which model performs the best. This dataset was essential for training CNNs on GPUs as well; the first model to be trained on this hardware took place in 2012 at the annual ILVRSC conference. Even though ImageNet is a dataset for object recognition, many segmentation

models use a CNN backbone trained on ImageNet, which speeds up convergence for the segmentation model's training.

Finally, a further large dataset utilized for object detection and segmentation tasks is ***MS COCO*** (meaning *Microsoft Common Objects in Context*). There are 328.000 photos with almost 1.5 million (labelled) instances. More than 200 000 photos have been separated into the training, validation, and test sets for the MS COCO detection challenge, totaling 80 classes.

4.2.2 Floor plan dataset

Regarding the publicly available datasets of floor plans, here are listed some of them. The first dataset is *CVC-FP*, which includes 122 pictures of floor plans that represent four different floor plan types. The actual information consists of structural symbols like rooms, walls, doors, and windows at the instance segmentation level [28]. Another dataset is *Rent3D*, which includes 215 floor plan images. Though instance segmentation can be simplified by the ground truth, some pre-processing is necessary because some object masks are not defined there [29]. Finally, the largest dataset (*CubiCasa5k*) contains 5000 photos with a ground truth for instance segmentation of more than 80 symbols. In April 2019, the dataset was made available. The above-mentioned datasets are out of our scope because the goal of this thesis is to parse floor plan images obtained from a lidar sensor placed on mobile robots. The key objective was finding a suitable dataset to use as training data for Mask R-CNN. This network needs to be able to generalize and segment an indoor environment into various rooms. The first approach was to find a dataset made of grid maps of indoor structures, but since there were not available this method has been discarded.

HouseExpo

After long research, a large-scale dataset has been found: HouseExpo [30]. While Deep Learning algorithms have recently advanced, some researchers are attempting to adapt

learning-based approaches to the activities performed by mobile robots, which demand a significant amount of data. This dataset has been developed with the intent of accommodating this demand. It is made by a large-scale indoor layout dataset with 35.357 2D floor plans and 252.550 rooms without furniture.

Figure 4.2: HouseExpo dataset samples. White pixels represent free space while black ones represent obstacles (Walls) [30].

Numerous researchers have tried to use deep learning methods on mobile robots in recent years. The absence of substantial datasets with a variety of samples, however, is one of the fundamental challenges in training deep neural networks. On the one hand, the 2D floor plan dataset sizes that are currently available are constrained. The MIT campus dataset and KTH campus dataset have the largest 2D floor plan collections that we are aware of, with 775 and 165 floor layouts, respectively. In addition to their small sample sizes, another issue is the lack of diversity in their samples. Because the MIT and KTH datasets were gathered from campus buildings, the location of rooms follows a specific distribution that might not be present in other, more commonly used environments, such as homes and offices. As a result, the variety of samples and the range of possible application scenarios are constrained. Furthermore, neither of these two datasets takes into account how important room connectivity is. Robots are usually initialized at a

random place at the start of each episode when they are in the training stage. They are likely to be unable to learn if the dataset has a poor connectivity level, meaning it contains numerous isolated rooms that cannot be reached from any other neighbouring rooms. Therefore, these works are assessed either in simple simulated environments that lack realism in terms of spatial organization or in a small number of related scenarios that are weak in the capability of generalization because there is a lack of a large-scale 2D floor plan dataset. The HouseExpo dataset has been built on the SUNCG dataset one of the most frequently used 3D environment datasets in the computer vision community. It is made up of 45.622 manually created 3D house models, and was initially developed to aid in semantic scene completion, a task for concurrently producing a 3D voxel representation and semantic labels using a single-view observation [31]. The process for building the HouseExpo dataset is the following: from the SUNCG dataset, we first extract a 3D structure model called s_i. It should be noted that the required interior map cannot be considered the projection of the top view of s_i into a 2D plane because the lintel prevents it from accurately depicting the connectivity between rooms. Then, at heights h_g and h_d, respectively, we get the ground cross-section plane P_g and the door cross-section plane P_d. As a result, by subtracting P_d from P_g, it is simple to establish the doors' location set L. Additionally, since the rooms in P_g are closed, we can easily determine the contour of the home and obtain the interior layout P_g by adding obstacles to the outside of the perimeter. Furthermore, given knowledge of door location set L, the doors are eliminated from P_g.

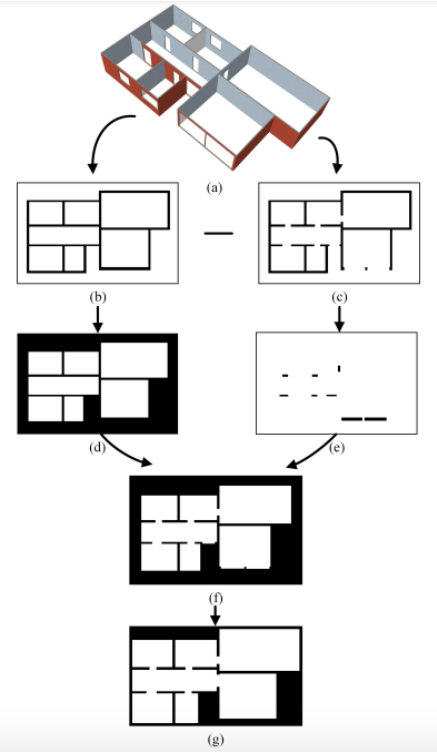

Figure 4.3: Generation process of HouseExpo dataset [30].

Once defined how the dataset is built, we can observe how it is used in our task. In the repository available online, each image is associated with an *id* name for identification and a corresponding JSON file containing annotations about images.
The format of the data is the following:

- ***id***, which corresponds to a unique house ID.
- ***room_num*** corresponds to the number of rooms in the house.
- ***bbox***, containing the bounding box of the entire house (*min*: (*x1, y1*), *max*: (*x2, y2*)).
- ***verts***, each couple (*x, y*) corresponds to a vertex location (*in meters*).
- ***room_category*** represents the category of the room and the corresponding bounding box.

This annotation is not appropriate for our instance segmentation task. Moreover, the dimension of the images is variable, and it could be a problem for the Mask R-CNN training. Some pre-processing operations have been done, for instance, each image has been resized to 1024×1024 pixels size maintaining the aspect ratio and adding zero padding as the contour of the image if necessary. This operation was necessary to perform the network training. As stated before, the annotations are not adequate for the segmentation task since JSON files contain only the bounding box of the rooms. Some pre-processing operations were tried (modifying meters in pixels, shrinking the list of room categories etc.) but none of them led to good results. From the HouseExpo dataset is randomly extracted a subset of images (4224 to be precise) and it has been created a unique JSON file containing the instance segmentation ground truth for each image. The ground truth contains only one class: Room. This decision is made because, after the modifications to the images, it is not possible to distinguish the room categories and also because our scope is to parse the floor plans into different rooms. The process for creating the JSON file containing the annotations for every single image is made on https://www.makesense.ai/ and it took more or less 2 months of work. Make Sense is a software that allows us to upload a set of images and perform segmentation tasks by drawing polygons on particular instances. Once all the masks have been drawn, it is possible to export the annotation file in the format that you prefer, for example, *YOLO, VOC XML, VGG, JSON*, and *CSV*. In our case, the annotation file is exported as a JSON file in COCO format. For segmentation purposes, the annotations are:

- ***segmentation***, a list of points (*x, y* coordinates) that define the object shape.
- ***area***, which is the area of the mask measured in pixels.
- ***iscrowd***, it defines whether the segmentation is for a single object (iscrowd=0) or a group of objects (iscrowd=1).
- ***image_id***, it defines a particular image in the dataset.
- ***bbox***, it represents the bounding box of an instance, and it has the following format (*x* position top left corner, *y* position top left corner, width, height).
- ***category_id***, it is a single category that is included in the categories section.

- ***id***, it corresponds to the id number of each annotation (*unique* with respect to the other annotations in the dataset).

In the following figure are reported some samples of segmentation performed.

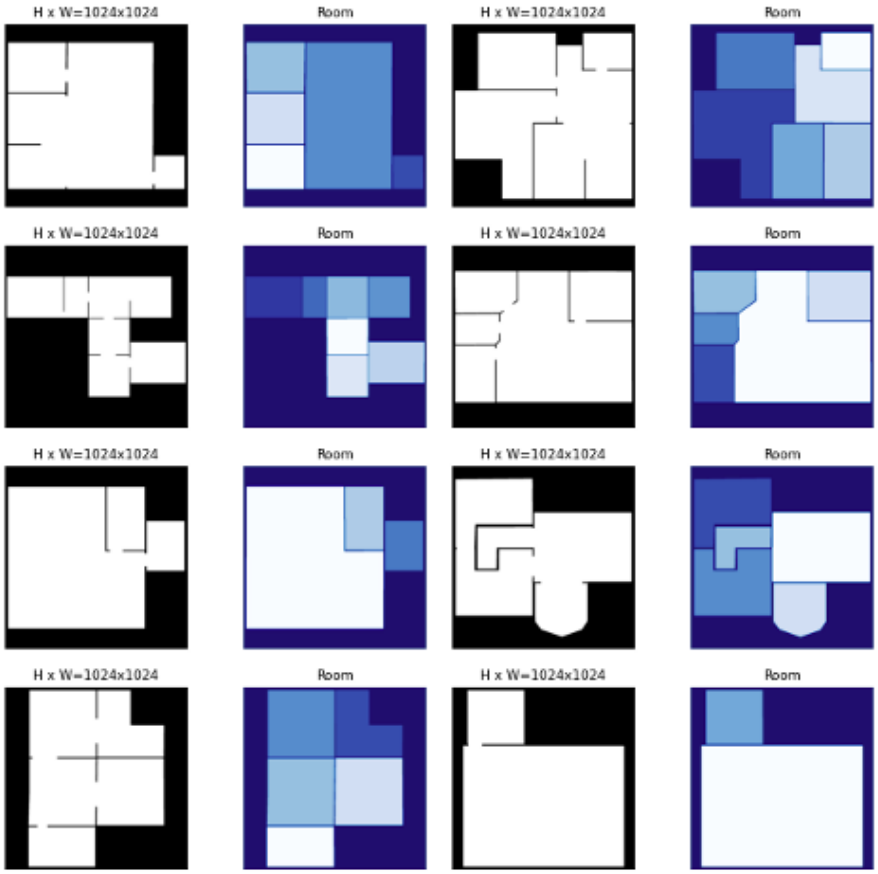

Figure 4.4: Graphical representation of floor plans (left) and corresponding ground truth (segmentation of the rooms).

The final dataset contains 4224 images, and it is split according to the following percentage: 80% (3380 images) is allocated for training the Mask R-CNN, 10% (422 images) is used for validation, and the remaining 10% (422 images) is used for testing. The Pareto principle served as some inspiration for the following 80%-20% split of the training data, which is also frequently observed in the literature.

4.3 Model Training

The training phase of the model is without a doubt the most time-consuming and important part of the process. Mask R-CNN has several parameters whose variation affects the outcome more or less significantly. A model is deemed successful if it can accurately learn the instances contained in the training set while preserving a strong generalization of the information acquired. Naturally, it is expected that loss functions associated with the training set will tend to decrease as the number of epochs rises. When there is a considerable increase in the validation loss, it indicates that the model is overfitting, which occurs when it learns the examples from the training set so well that it is unable to operate effectively in a more generic environment. To produce a suitable model, it is important that the loss function relative to the validation set (validation loss) does not expand with the increase of the epochs. When these issues arise, it is necessary to identify their causes, which may be numerous: Too high/low *Learning rate*, *backbone* of the model too complex or simple for our problem, *weight decay* high or low, the *learning rate decay* policy has little impact on learning. It could be necessary to change the *image shape's* dimensions, which reflect the size of the image that Mask R-CNN scaled. Moreover, the possibility of modifying only some weights can occur (*head*, *4+ layers, all* and so on). The standard configuration was initially used. The most significant parameters are shown below:

Hyperparameter	Value
Backbone	Resnet101
Image shape	[1024 1024 3]
Batch size	1
Learning rate	0.001
Steps per epoch	500
Validation steps	50
Weight decay	0.0001
Network trained	Head

Table 4.1: Hyperparameters for the first training.

The primary goal of the initial training session was to examine how the Mask R-CNN hyperparameters affected the final model. Each model created during this phase took 50 training epochs with 500 steps each to complete. The training process is made on Google Colab, it is a free tool in the Google suite that allows you to write python code directly from your browser. An online platform that offers a cloud hosting service for Jupyter notebooks where you can create rich documents that contain lines of code. The GPU used is a Tesla T4 with 16 GB of RAM. The pre-trained COCO weights have been utilized to initialize the network before performing the Mask R-CNN training. Initially, the process of training was very slow, and to speed up the process the image shape parameter has been modified to [512, 512, 3]. This change has allowed halving the time necessary for the training of the model, but the loss function and the validation loss function have a worse descending curve. It has been assessed how well the network performed after the backbone was changed. The validation loss tended to grow with a higher slope, therefore even though the training speed was increased by using resnet50 instead of resnet101, the learning was not improved. Due to the use of previously trained weights, it was required to consider numerous times whether or not to train the model by updating only the weights associated with the final levels of the network, or better fully connected layers, and "freezing" the weights associated with the previous layers. To highlight which layers will be involved in the training process, it will be used *head* to update only the weights related to fully connected layers, while *all* to update the weights related to the whole network. Hybrid configurations have often been tested: one approach is the modification of the learning rate at different epochs, another one is the variation of the layers involved in the weight updating process (for example, on *k* periods of training, in the first *k/3* epochs will be updated only the weights of the head, then in the following *k/3* epochs the 4+ layers will be trained, finally for the remaining epochs all the weights of the network will be updated). This type of training led to different results in terms of loss function and validation loss. After identifying the hyperparameter settings that reduced the loss function and the validation loss function, the work has proceeded to analyse the trainings obtained on a higher number of epochs, to check its long-term performance. In the following section will be shown the configurations and the value of the losses for the last

trainings. These are the models that optimally perform the segmentation task. The configurations used for the first trainings are not reported since did not lead to good results and were mainly done to tune the hyperparameters.

All the models use the ResNet101 backbone since it performs better than the ResNet50. They are all trained for 500 steps per epoch and use 50 validation steps. They are initialized with coco weights, and the whole network is trained except for model 1. It is trained the head for the first 50 epochs, 4+ layers for the next 30 and lastly all the layers for the final 30 epochs. Generally, the learning rate is initiated at 0.001 and then divided by 10 after a certain number of steps. All these variations are better explained in the following table.

	Model 1	***Model 2***	***Model 3***	***Model 4***	***Model 5***
Hyperparameter	***Value***	***Value***	***Value***	***Value***	***Value***
Backbone	Resnet101	Resnet101	Resnet101	Resnet101	Resnet101
Image shape	[512 512 3]	[512 512 3]	[256 256 3]	[1024 1024 3]	[1024 1024 3]
Batch size	6	6	10	2	2
Learning rate	0.002 (1-50) 0.0002 (51-80) 0.00001 (81-110)	0.001	0.001	0.001 (1-30) 0.0001(31- 50)	0.001 (1-30) 0.0001(31- 50) 0.00001(51-70)
Steps per epoch	500	500	500	500	500
Validation steps	50	50	50	50	50
Weight decay	0.0001	0.0001	0.0001	0.0001	0.0001
Epochs	110	50	50	50	70
Network trained	Head (1-50) 4+ Layers (50-80) All (80-110)	All	All	All	All
Max GT instances	100	100	100	60	60
Best Loss	0.1752	0.2504	0.3171	0.1497	0.1501
Best Val. Loss	0.3126	0.2519	0.3590	0.1834	0.1914
Training Time	1d 10h 52m	6h 42m	8h 10m	1d 6h 34m	2d 2h 16m

Table 4.2: Hyperparameters configuration of Mask R-CNN models.

TensorBoard made it possible to track the development of the training procedure by looking at how the loss functions associated with the training set and the validation set change as the number of epochs rises. Since the training process is made at different steps (the time available to perform the training on Google Colab is limited), the model is initiated with the best weights of the previous training of the same model.

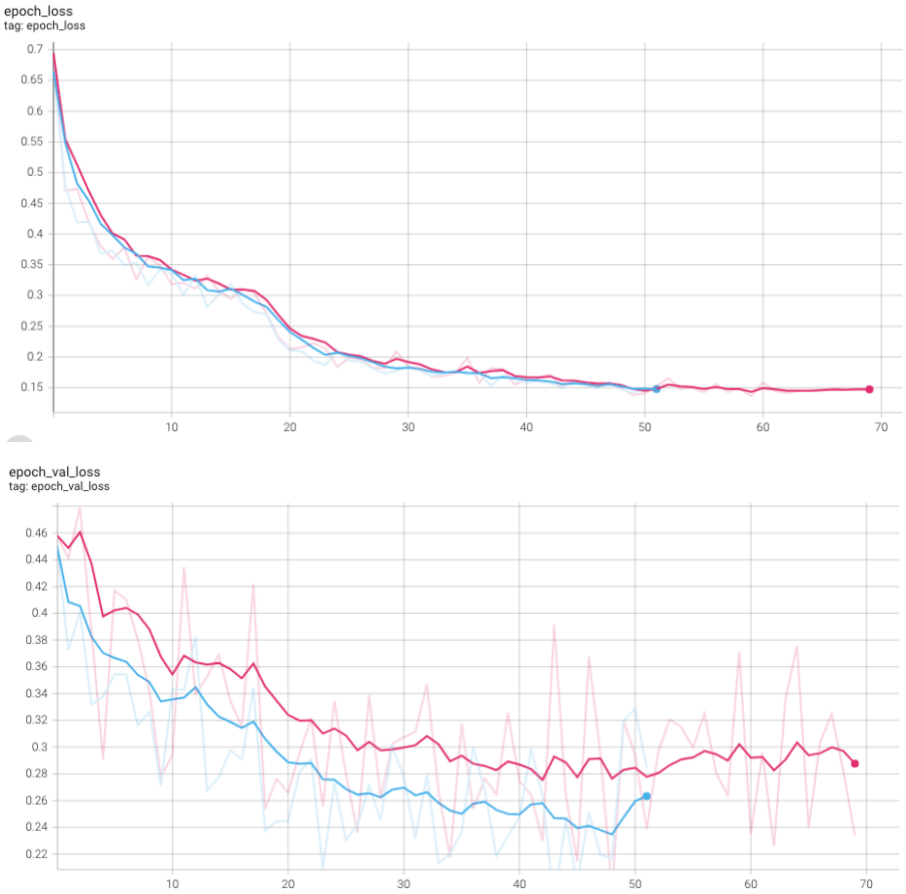

Figure 4.5: Visualization of the comparison between two configuration models, in particular Model 4 (blue) and Model 5 (magenta).

As can be seen from figure 4.5, the loss functions related to the training set and validation set are going down with the increase of the epochs. Model 4 has the same descending path on the training set but a better validation loss. Moreover, the number of epochs

necessary to obtain the best results in terms of validation loss is 44 in the Model 4 case, and 49 in the Model 5 case. These results are obtained considering a decreasing *learning rate*: it is set initially as 10^{-3} till arriving at 10^{-4} for model 4 and 10^{-5} for model 5. In this way, the learning task is improved since maintaining the learning rate fixed at one value, leads to overfitting problems. This adaptive learning rate allows us to obtain a good capacity of generalization. It is possible to observe that training directly all the layers of the Mask R-CNN performs better than training firstly the head and then the remaining layers. Furthermore, if the images are given as input to the network with 1024×1024 pixels size, the performances are better than the configurations that have a smaller image shape.

Chapter 5

Results and Analysis

It is not enough to just compare the two loss function graphs to come to the conclusion that model A is preferable to model B. Therefore, it was important to determine the metrics associated with the best models, which had been selected based on the performance of the two loss functions. The metrics that are used to evaluate the model's performances are the same mentioned in paragraph 3.5. In particular, Intersection over Union, Precision and Recall. These are utilized to compute Average Precision and Average Recall. The validation set and test set datasets are used to assess each model's performance, thus it will be important to compare the segmentation masks and bounding boxes that were generated for each instance with the corresponding ground truth values. In the computation of the Average Precision, it has been varied the Intersection over Union threshold.

5.1 Average Precision

Before computing the Average Precision on the whole test set, a custom callback has been developed to calculate the Average precision on a certain period of time during the training phase, this has allowed us to detect some problems related to parameter tuning. Hereafter is shown the code of the callback.

```
def _calculate_mean_average_precision(self):
    mAPs = []

    # Use a random subset of the data when a limit is defined
    np.random.shuffle(self.dataset_image_ids)

    for image_id in self.dataset_image_ids[:self.dataset_limit]:
        image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(self.dataset, self.inference_model.config,
                                                                         image_id)
        molded_images = np.expand_dims(mold_image(image, self.inference_model.config), 0)
        results = self.inference_model.detect(molded_images, verbose=0)
        r = results[0]
        # Compute mAP - VOC uses IoU 0.5
        AP, _, _, _ = utils.compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"],
                                       r["class_ids"], r["scores"], r['masks'])
        mAPs.append(AP)

    return np.array(mAPs)
```

Figure 5.1: Custom callback of mean Average Precision computation.

The results of training each network with different parameters and image shapes on the validation and test datasets are shown in the following table. The average precision on the test set and the validation dataset is relatively balanced, indicating that the model has learned to perform the segmentation task.

	Average Precision					
	Validation set			***Test set***		
Models	AP^{50}	AP^{75}	$AP^{[0.50:0.95]}$	AP^{50}	AP^{75}	$AP^{[0.50:0.95]}$
Model 1	0.911	0.865	0.824	0.909	0.862	0.815
Model 2	0.966	0.923	0.820	0.965	0.931	0.820
Model 3	0.933	0.831	0.676	0.941	0.820	0.677
Model 4	0.973	0.946	0.857	0.977	0.952	0.861
Model 5	0.969	0.939	0.853	0.979	0.958	0.861

Table 5.1: Average precision results obtained on validation and test set.

It is possible to conclude that the optimal models are 4 and 5 which obtain the best performances on both datasets. Further analysis shows how model 5 has slightly better performance in terms of *AP* but the training process is more or less 2 times bigger than the training of model 4. It is therefore reasonable to draw the conclusion that, in terms of overall performance, the configuration of the fourth model is the best for the problem under consideration but also the fifth configuration gives optimal results. On the following page, there will be compared some figure about the floor plans' original images, the ground truth with the segmentation masks and finally the masks predicted by Mask R-CNN. Furthermore, it will be depicted the differences between the mask computed by models 4 and 5 on some images that are chosen randomly from the test set. As can be seen from this qualitative evaluation, the networks can segment some images perfectly, in particular floor plans that contain rooms that have rectangular and square shapes.

Structures that present rooms with a more complex shape or that show particular connectivity are more difficult to segment.

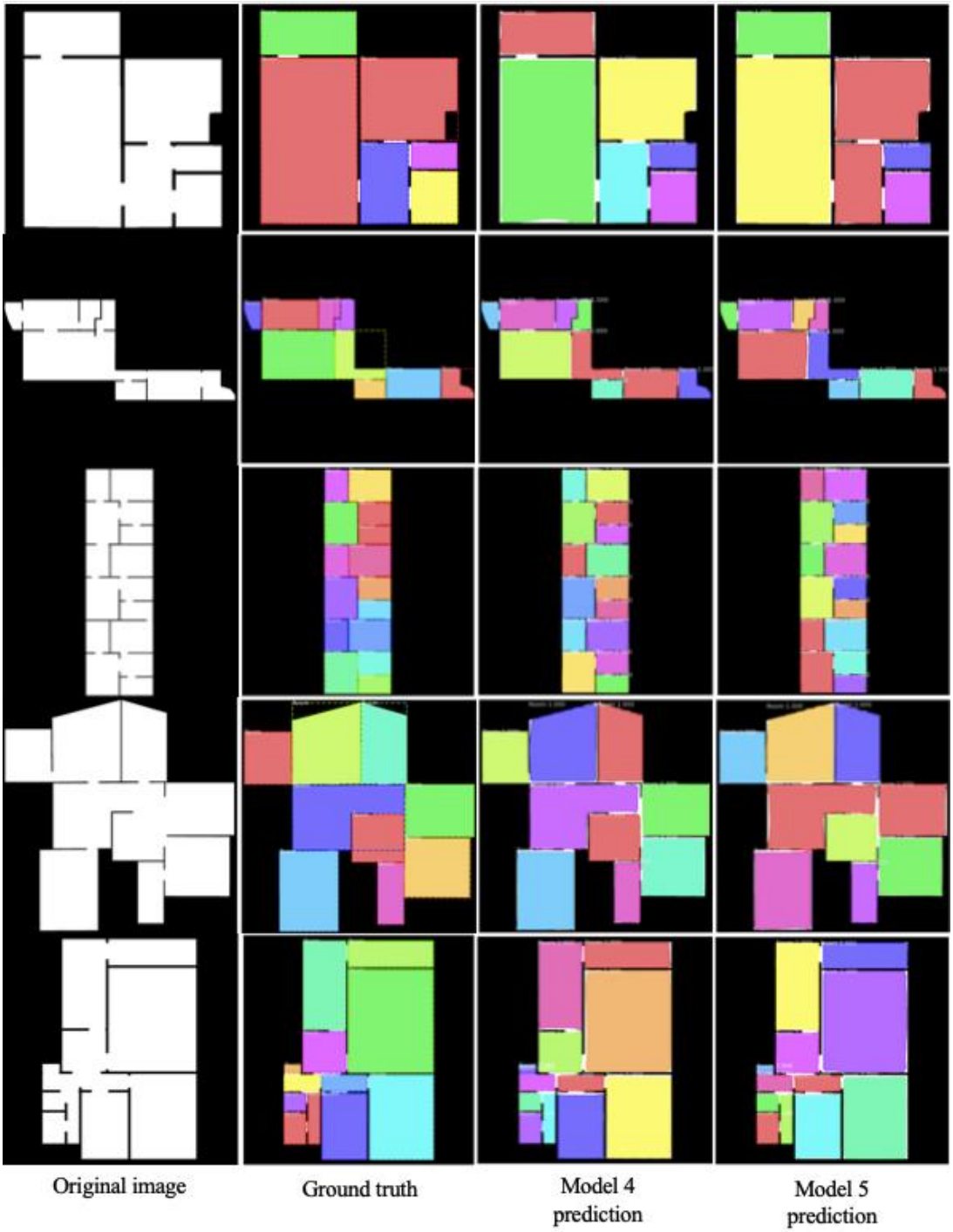

Figure 5.2: Representation of floor plan images belonging to the test dataset. Original image, ground truth, and predicted masks.

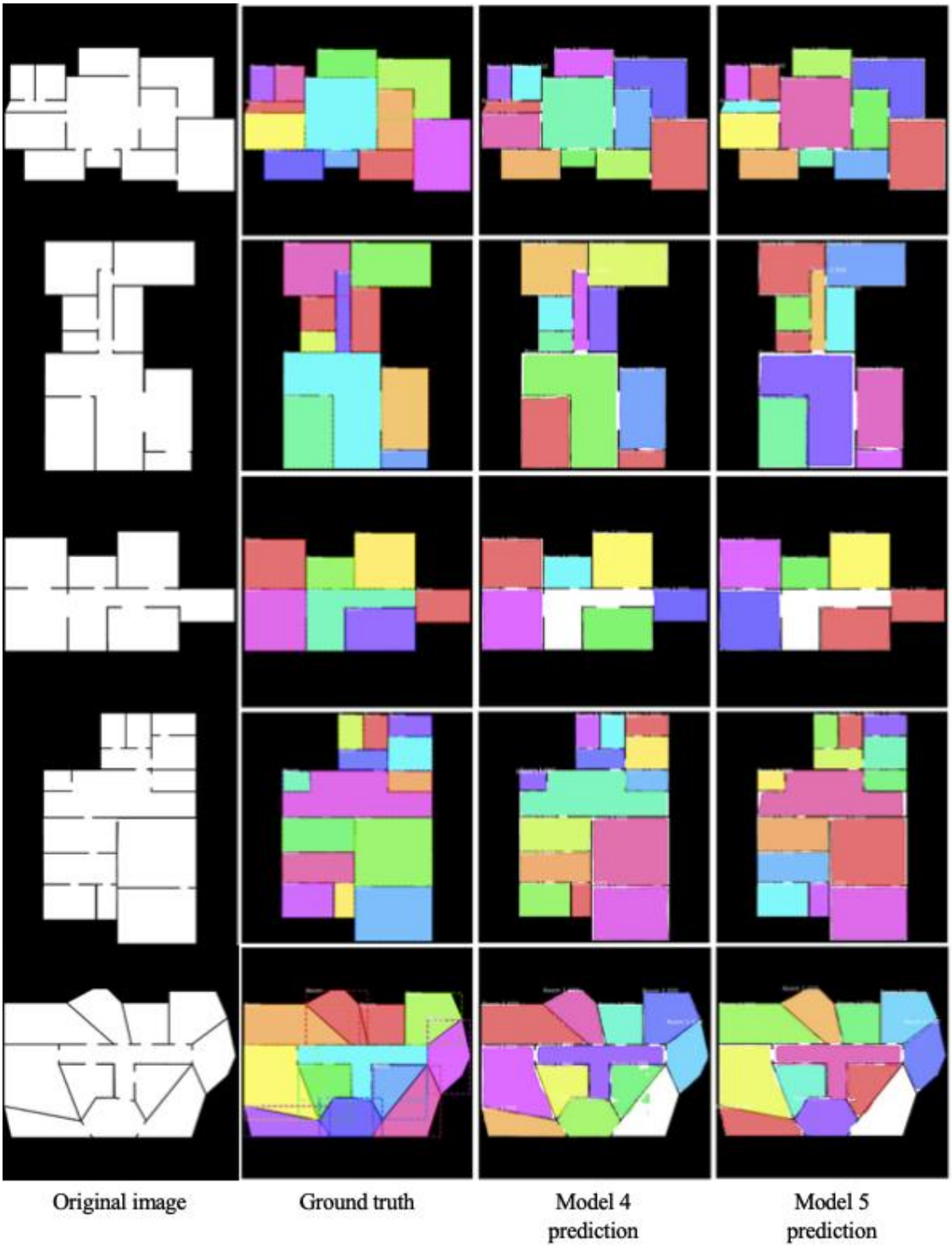

Figure 5.3: Representation of floor plan images belonging to the test dataset. Original image, ground truth, and predicted masks. In the 3rd and 5th images, some instances are not detected.

5.2 Qualitative test

In the previous paragraph, we have seen the performance of models 4 and 5 on the test dataset obtained by splitting the entire dataset. The models perform quite well on that type of structure. To evaluate the performance on new images that do not belong to the initial dataset, some new floor plan maps have been tested. In particular, it had been taken into consideration the work "Room Segmentation: Survey, Implementation, and Analysis" [3]. This evaluation uses different maps taken from their GitHub repository. The floor plan images are obtained from a gazebo simulation of several kinds of office environments. All of the floor designs were created both with and without furniture to evaluate the impact of modifications to the original architecture. In this test will be considered only structures that present maps without furniture. Before predicting the masks of these indoor environments, the images have been pre-processed to be compliant with pictures of the dataset.

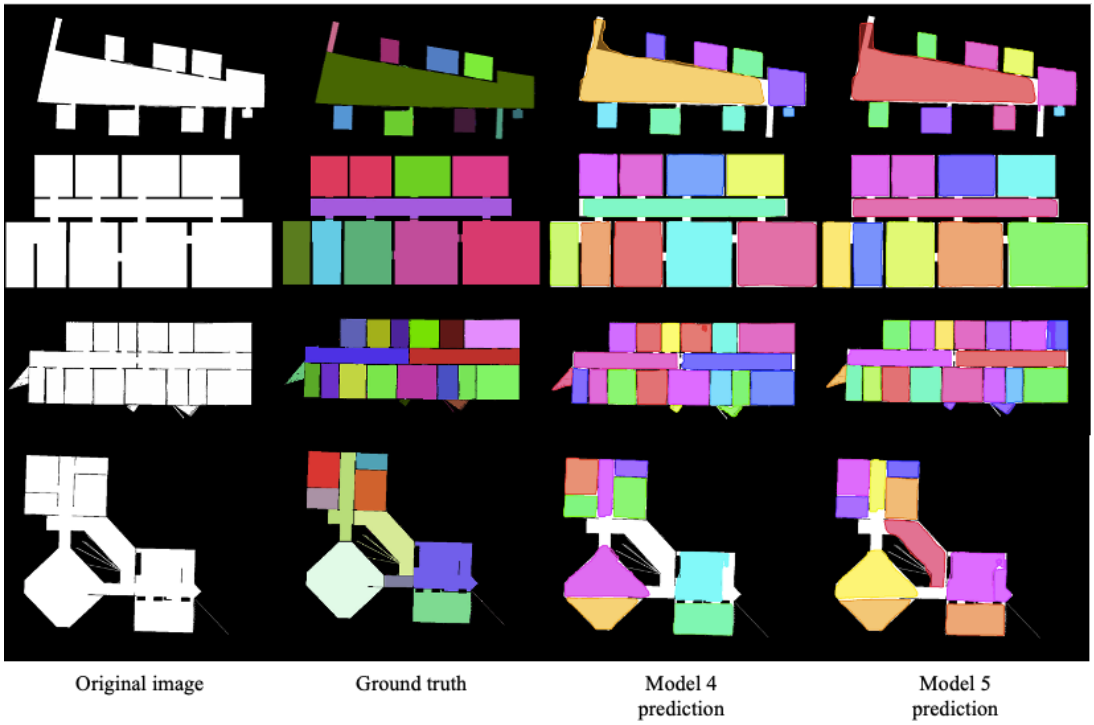

Figure 5.4: Representation of room segmentation of models 4 and 5 on the new set of structures.

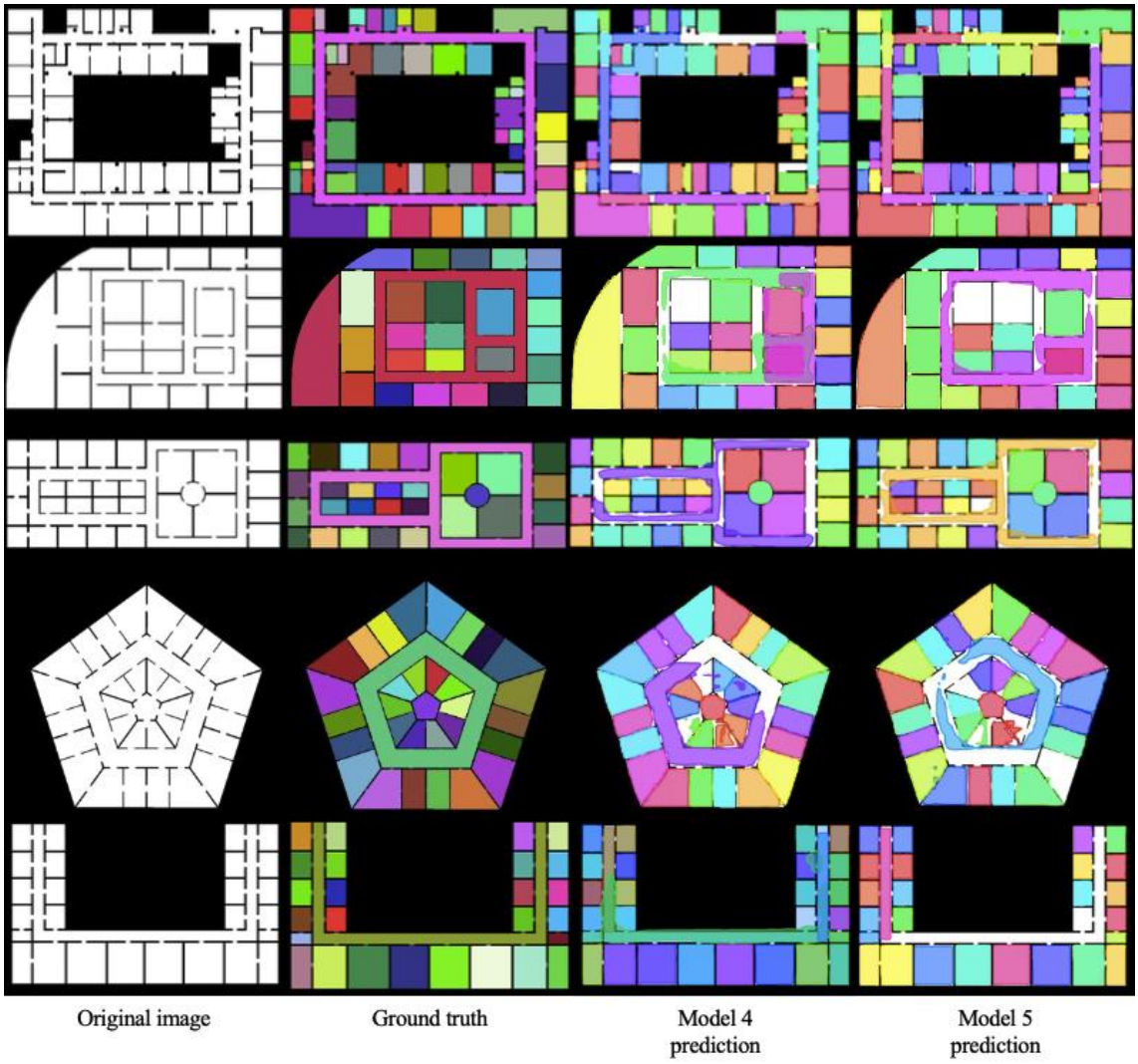

Figure 5.5: Representation of room segmentation of models 4 and 5 on the new complex layout.

It is possible to notice that the models correctly predict most of the instances present in the figures. The simpler structures depicted in figure 5.4 are properly segmented into the various rooms. Some spaces are not correctly divided, this is because these rooms have a particular shape, for instance, the pentagonal corridor present in figure 5.5. Analyzing the results obtained, the quality of room segmentation is reasonably good, even though the mask shape can be improved by applying some post-processing refinement algorithm. For example, can be employed GrabCut and Watershed algorithms from the OpenCV library.

5.3 Map acquisition and test

In robotics, simultaneous localization and mapping (*SLAM*) is the computational problem of building or updating a map of an unknown environment, while simultaneously keeping track of the location of an agent within it. A further test has been carried out to determine if the segmentation performed by the neural network is also effective on a processed occupancy grid map. First of all, thanks to the use of the robotics simulator Gazebo, a structure consisting of 9 rooms is built. This space is constructed using *gazebo_ros_pkgs*, a collection of ROS packages that give the appropriate interface to simulate a robot in the created 3D world. The following packages are included in the ROS 2 gazebo_ros_pkgs:

- *gazebo_dev*: Offers a Cmake configuration for the ROS distribution's default Gazebo version.
- *gazebo_msgs*: Data structures for messages and services used by ROS2's Gazebo.
- *gazebo_ros*: Offers practical C++ classes and functions that other plugins, such gazebo_ros::Node, conversion tools, and testing utilities, can utilise. It also offers a few plugins that are generally helpful.
- *gazebo_plugins*: A group of Gazebo plugins that let ROS2 access sensors and other functionalities. For instance, the gazebo_ros_camera publishes ROS2 photos, while the gazebo_ros_diff_drive interface enables ROS2 users to control and observe differential drive robots.

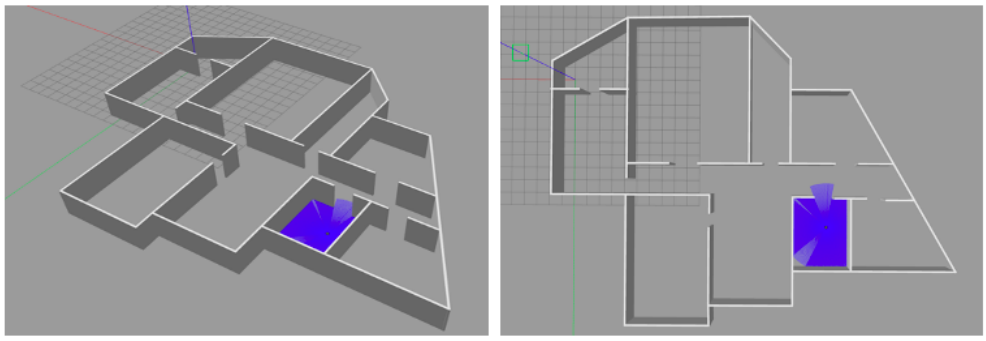

Figure 5.6: Simulated building on Gazebo.

Once the indoor structure is built, a file world that serves our needs is generated. Then, the robot interface and state publisher are launched. Nav2 is utilized for robot navigation in the environment created. After launching the SLAM (SLAM toolbox), it is possible to move the robot by setting a goal through RViz. Furthermore, it can be launched the teleop_twist_keyboard node which takes the commands from the keyboard and publishes them as twist messages to move the robot. A mobile differential robot is employed during the mapping and the laser scan is displayed. Since the range of the laser scan is 3.5 meters, some spaces in the bigger rooms are not mapped. This problem can be solved by employing a more powerful lidar sensor that has a bigger laser range. This allows us to achieve a better map scan. After the entire mapping of the building is performed, the updated grid map is saved. The following figure represents the map obtained.

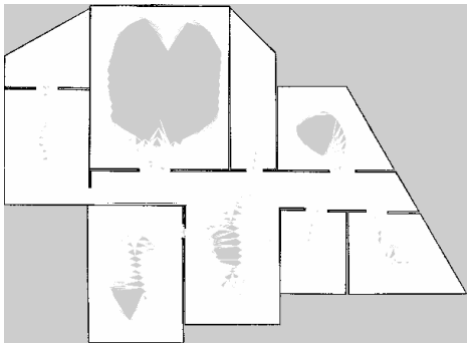

Figure 5.7: Map generated by the robot in simulation.

The map generated is processed to obtain a floor map similar to the images belonging to the dataset. The part of the map that is not detected in the rooms is set to white, while the exterior part is set to black (representing obstacles). The final map to be segmented is represented in the following image. The processed map is utilized to test the trained models. As can be seen from the results, the masks predicted by the models are very accurate. The building structure is segmented almost perfectly, and all the instances are detected. Model 5 can also recognize the corners of the T-shaped room, while the model 4 is not detecting it perfectly.

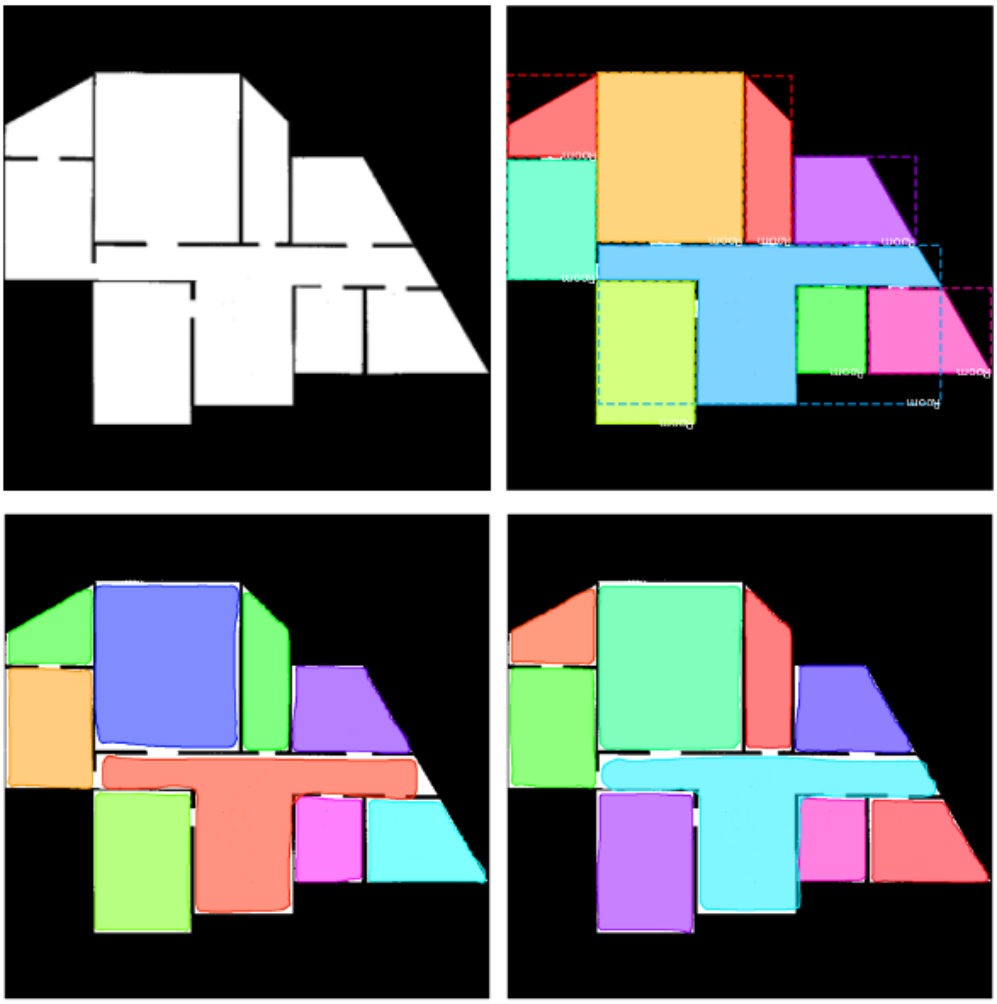

Figure 5.8: Representation of predicted masks on the map generated in simulation. Processed images (top left), ground truth (top right), model 4 prediction (bottom left), model 5 prediction (bottom right).

Chapter 6

Conclusions

The primary objective of the thesis is to investigate a deep learning approach capable of carrying out a segmentation task of floor plan images. In this project, a Mask R-CNN is evaluated on a dataset obtained from a large set of floor plan maps. The network is then trained on a portion (80%) of the entire dataset (4224 images) and evaluated on the test set obtained partitioning the remaining part of the dataset. Generally, it is noted that the models produced have excellent classification and segmentation capabilities for images with a low/medium number of instances. On the other hand, some rooms are not segmented perfectly when the instances present in an image are numerous. The models are tested also on new floor plan images that do not belong to the dataset. These are firstly pre-processed and then used to perform the segmentation task. The results are good but can be improved by incrementing the dataset with different types of structures. Models 4 and 5 obtain respectively an Average Precision of 0.977 and 0.979 on the test set (with IoU = 0.5). Finally, the 2 configurations are tested on a map generated by a mobile robot in the Gazebo simulation environment. Some problems still affect the task, in fact, the segmentation algorithm is influenced by the dataset used for the training process. Firstly, the floor plan images are taken without furniture, and this could be a negative aspect since this work is thought to segment images obtained from Lidar data acquisition. Generally, there are obstacles in an indoor environment that do not allow the mobile robot to perform a perfect mapping operation. Moreover, glazed walls do not help to take good measurements when only a lidar sensor is utilized. The generalization to these cases can be considered by adding new images to the training dataset.

This project sets the stage for several potential follow-up projects:

- The previous approach can be improved by using a dataset containing structures with furniture and obstacles. This would allow us to have a better generalization

and will improve the results when a noisy occupancy grid map is obtained by data acquisition.

- It is possible to perform a post-processing operation on the masks. For instance, masks can be adjusted to fill out spaces that are near the walls. Another example could be the application of an edge detection algorithm and then combining the results obtained from 2 networks.
- It can be integrated with other sensors (i.e. camera) to perform object detection and so the classification of the rooms.
- The navigation component of a mobile robot can be developed, enabling it to navigate autonomously in SLAM and perform the segmentation once the mapping is finished.

Given that the state of the art is continually being updated, carrying out the same task using a neural network with a more recent implementation could result in an even greater improvement in the performance of the produced models.

Bibliography

[1] The Economist. 06-05-2017. “The world’s most valuable resource is no longer oil, but data.”. Available at: https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data. Accessed: 19-10-2022.

[2] Branka Vuleta. 28-10-2021. “How Much Data Is Created Every Day? +27 Staggering Stats”. Available at: https://seedscientific.com/how-much-data-is-created-every-day. Accessed: 19-10-2022.

[3] R. Bormann, F. Jordan, W. Li, J. Hampp and M. Hägele. "Room segmentation: Survey, implementation, and analysis," 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 1019-1026, doi: 10.1109/ICRA.2016.7487234.

[4] *Arthur Samuel (computer scientist)*. (2022). *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/Arthur_Samuel_(computer_scientist). Accessed: October 25, 2022.

[5] T. Mitchell. *Machine Learning*. McGraw Hill, 1997. isbn: 978-0-07-042807-2.

[6] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd edition. O’Reilly Media, Inc., 2019. isbn: 9781492032649.

[7] Cornell University News Service records, #4-3-15. Division of Rare and Manuscript Collections, Cornell University Library.

[8] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[9] Chatzilygeroudis, Konstantinos et al. “Machine Learning Basics.” Intelligent Computing for Interactive System Design (2021).

[10] Enes Zvornicanin, 06-11-2022, “Relation Between Learning Rate and Batch Size”, Available at: https://www.baeldung.com/cs/learning-rate-batch-size.

[11] Pragati Baheti, 21-10-2022, “Activation Functions in Neural Networks [12 Types & Use Cases]”, Available at: https://www.v7labs.com/blog/neural-networks-activation-functions.

[12] Albelwi, Saleh, and Ausif Mahmood. 2017. "A Framework for Designing the Architectures of Deep Convolutional Neural Networks" Entropy 19, no. 6: 242. https://doi.org/10.3390/e19060242.

[13] Matthew Stewart, 05-06-2019, “Advanced Topics in Deep Convolutional Neural Networks”, Available at: https://towardsdatascience.com/advanced-topics-in-deep-convolutional-neural-networks-71ef1190522d.

[14] Yani, Muhamad & Irawan, S, & Setianingsih, Casi. (2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry’s Nail. Journal of Physics: Conference Series. 1201. 012052. 10.1088/1742-6596/1201/1/012052.

[15] Russ Tedrake. 20-10-2022. "Robotic Manipulation", "Perception, Planning, and Control", Available at: http://manipulation.mit.edu.

[16] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik. (2013). “Rich feature hierarchies for accurate object detection and semantic segmentation”. Available at: https://arxiv.org/abs/1311.2524.

[17] Shilpa Ananth. 05-08-2019. “Fast R-CNN for object detection”. Available at: https://towardsdatascience.com/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022.

[18] Ross Girshick. (2015). "Fast R-CNN". arXiv: 1504. 08083. Available at: http://arxiv.org/abs/1504.08083.

[19] Shaoqing Ren et al. (2015). “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. arXiv: 1506.01497. Available at: http://arxiv.org/abs/1506.01497.

[20] Kaiming He et al. (2017). “Mask R-CNN”. arXiv: 1703. 06870. Available at: http://arxiv.org/abs/1703.06870.

[21] Tan, Chengjun & Uddin, Nasim & Mohammed, Yahya. (2019). Deep Learning-Based Crack Detection Using Mask R-CNN Technique.

[22] Tsung-Yi Lin et al. (2016). "Feature Pyramid Networks for Object Detection". arXiv: 1612.03144. Available at: http://arxiv.org/abs/1612.03144.

[23] Vineeth S Subramanyam. (2021). “IOU (Intersection over Union)”. Available at: https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef.

[24] Jonathan Hui. (2018). “mAP (mean Average Precision) for Object Detection”. Available at: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[25] Picture source: https://cocodataset.org/#detection-eval.

[26] Abdulla, W. (2017). Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow. In GitHub repository. Github. https://github.com/matterport/Mask_RCNN.

[27] Anmol Dua. (2019). “YOLACT (Real time Instance Segmentation)”. Available at: https://medium.com/@anmoldua/yolact-important-points-125268f475d0.

[28] L.-P. de las Heras, O. Terrades, S. Robles, and G. Sanchez. (2015). “Cvc-fp and sgt: a new database for structural floor plan analysis and its groundtruthing tool”. *International Journal on Document Analysis and Recognition*.

[29] Chenxi Liu, A. G. Schwing, K. Kundu, R. Urtasun and S. Fidler, "Rent3D: Floor-plan priors for monocular layout estimation," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3413-3421, doi: 10.1109/CVPR.2015.7298963.

[30] Tingguang Li, Danny Ho, Chenming Li, Delong Zhu, Chaoqun Wang, Max Q.-H. Meng*, Fellow, IEEE. (2019). “HouseExpo: A Large-scale 2D Indoor Layout Dataset for Learning-based Algorithms on Mobile Robots”. Available at: https://cs.paperswithcode.com/paper/houseexpo-a-large-scale-2d-indoor-layout.

[31] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. (2017) “Semantic scene completion from a single depth image,” IEEE Conference on Computer Vision and Pattern Recognition (CVPR).