POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica (LM-32)

Tesi di Laurea Magistrale

Deep Learning and Augmented Reality application for minimally invasive urologic surgery support

Relatore:
Prof. Pietro Piazzolla

Correlatore:
Prof. Andrea Sanna

Candidata:
Giorgia Marullo
258131

Anno Accademico 2020-2021

Abstract

Artificial intelligence and deep learning are becoming increasingly popular in the medical field. In recent years, neural networks are consolidating as a support tool for doctors in the phases of diagnosis, prognosis, and in the operating room during delicate surgery.

This thesis aims to implement an Augmented Reality application for improving the surgeon's spatial perception during Robot-Assisted Radical Prostatectomy (RARP) stages. RARP is a minimally invasive urologic operation performed with Da Vinci surgical console support. This technology improves surgical precision, reducing post-operative complications and hospital stays. The focus is on finding a technique for real-time automatic registration. The proposed system trains a neural network to estimate the position, rotation, and scale of the object captured by the endoscope and intends to obtain an optimal overlap between the 3D virtual prostate model and its physical counterpart.

This work includes an introduction about thesis aim and context, theoretical notes on deep learning, a literature review of methods for 6D object pose estimation from images, a description of the proposed method and achievements.

Contents

Chapter 1

Introduction

In recent years, Artificial intelligence and Deep Learning are becoming increasingly common in clinical health care. This expanding interest is due to a growth in the volume of data available in digital form and the accelerating computational power, which allow to analyze and extract helpful information from data. In particular, the development of Convolutional Neural Networks has allowed the machines to learn useful tasks from images, such as detecting and characterizing suspicious patterns from radiographs or provide diagnosis and prognosis in different medical fields [1].

Urology was one of the first adopters of Artificial Intelligence for object detection, image classification, segmentation, skills assessment, and outcome prediction for complex urologic procedures [1]. Figure 1.1 shows an overview of the emerging AI applications in this medical field.

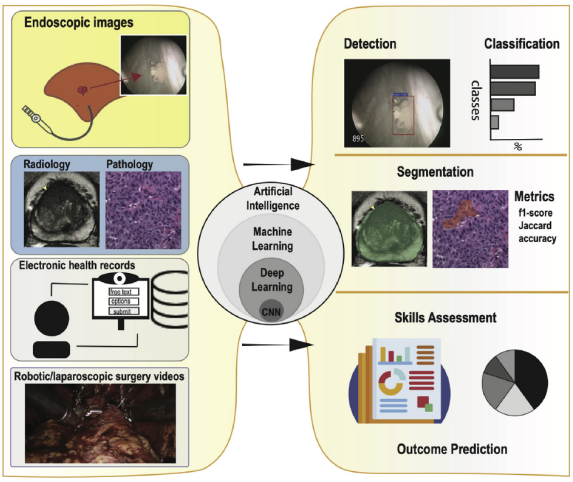


Figure 1.1. Artificial Intelligence in urology. (*Image from* [1])

Recently, minimally invasive laparoscopic and robotic-assisted approaches have replaced many traditional open urologic surgeries [1]. These technologies assist the surgeon in performing more complex and precise tasks improving the precision of the procedure, with consequent benefits for the patient during the post-operative period [2]. This change of paradigm led the surgeon to see the operatory scene through a console and a visor experience, reducing his spatial perception of the surgical environment [2]. This drawback can be mitigate exploiting Augmented Reality. Augmented Reality allows a real-time overlapping between computer-generated images or 3D models and the real environment. In the medical field this technology allows, for example, data visualization on diagnostic or treatment procedures. The main challenge for an Augmented Reality application is the registration process. Registration is the accurate alignment of the virtual model with its physical counterpart [3]. There are different strategies to achieve precise alignment. The first way is to use endoscopic markers placed on the surface of specific structures as points of reference during the surgery. Another option to determine the organ position in real-time is a marker-less approach, which is technically more challenging and time-consuming, and exploits machine learning techniques [3, 4]. The main advantages of Augmented Reality for doctors are [3]:

- The opportunity to view reconstructions directly on the body of the patient, reducing the number of distraction caused when the surgeon has to look away from the surgical site;
- The reduction of surgery time, exploiting AR during pre-operative planning for tailoring incisions and cutting plans;
- It improves the surgeon's spatial perception of the surgical field, avoiding unnecessary manipulations or accidental injuries to inner organs.

This thesis proposes to implement an Augmented Reality application for improving the spatial perception of the surgeon during Robot-Assisted Radical Prostatectomy (RARP) stages. RARP is a minimally invasive urologic operation performed with Da Vinci surgical console support. The RARP procedure comprises five stages [5]:

1. Defatting and incision of the endopelvic fascia;
2. Management of the bladder neck;
3. Vase clamping and nerve-sparing;
4. Surgery by the prostatic apex;
5. Targeted Biopsy.

Some of procedure's stages benefit from AR. The augmented video stream can be accessed directly into the Tile-Pro visualization system of the Da Vinci surgical

console [2], which allows the overlapping of the virtual model onto the endoscopic view [6]. Up to now, three distinct strategies exist for registration, applied in different stages of the procedure [2]:

- *Feature-based*, useful for rapid localization of the prostatic apex;
- *Human-assisted*, to improve precision during nerve-sparing, preserving nerves' functionality;
- *Marker-based*, helpful to locate tissue sampling for post-operatory biopsies.

The focus of this work is finding a new technique for real-time automatic registration. The proposed approach tries to solve the registration problem by using neural networks. The system trains two neural networks to estimate the position, rotation, and scale of the object captured by the endoscope to obtain an optimal overlap between the 3D virtual prostate model and its physical counterpart. The task is challenging for the prostate because it has a simple roundish shape, and its texture is easily confused with the surrounding tissues. The proposed approach employs two different networks: the former implements semantic segmentation to detect the region of interest, and the latter addresses the rotation issue. Either Segmentation Neural Network or Rotation Neural Network exploit a specific dataset for training, each manually created with different strategies. We trained the Segmentation Neural Network through images obtained from surgical videos, while for the Rotation Neural Network, we involved a synthetic dataset. We tested both the networks achieving valid results. In particular, for the segmentation network, we tried nine combinations of neural networks models and semantic segmentation architectures, choosing the best one according to the Intersection over Union metric. We tested the Rotation Neural Network on both synthetic and real images. We tried different combination for X-Axis, Y-Axis, and Z-Axis rotation estimations. The results for this network reached a satisfactory accuracy considering the high complexity of the task.
The following part of the thesis contains four Chapters:

- Chapter 2 introduces basic concepts about Deep Learning and its applications;
- Chapter 3 proposes a literature analysis and classification of 6D pose estimation methods from monocular images;
- Chapter 4 describes the proposed system in detail;
- Chapter 5 focuses on achievements and conclusions.

Chapter 2

Theoretical background on deep learning

2.1 Introduction

In this chapter, we will give theoretical notes on basic concepts about deep learning. The aim is introducing the methods used in this work. As we know, deep learning is a subset of machine learning, which in turn is a subset of artificial intelligence. The term Artificial Intelligence was born at the end of the 1950s. It was related to how modelling the process of the human mind through a calculator. Alan Turing introduced the idea that machines can simulate human being and have the ability to do intelligence things [1]. But the learning theory and the computing machine capacity could not allow the development of effective systems. This situation led to a period of disillusion because exciting ideas did not bring tangible results. In the 1980s, there was again a growing interest in machine learning systems thanks to the increasing capacity of recent computing machines and the consequent realization of first applications. Deep learning became popular after 2010, but it has a very long history. Despite neural networks were born before ten years ago, they have become useful only recently as the amount of available training data has increased, and computer infrastructure (both hardware and software) for deep learning has improved. These two phenomenons are directly proportional to the accuracy of systems, gradually improved over time [7]. In recent years, deep learning has become popular in the computer vision field for solving problems such as image classification, object detection, and face recognition accurately [8]. We involve intelligent software in many applications: understanding speech or images, giving support to scientific research such as to make diagnosis in medicine, or automate specific routines [9].

2.2 Machine Learning

As we said before, deep learning is a specific type of machine learning, so before analyzing it deeply, it is useful to introduce this discipline and the different kinds of learning to make the concepts described in next sections more clear. Machine learning studies algorithms for the detection and the extraction of patterns from raw data, involving knowledge of the real world and making choices that appear personal. These algorithms improve their abilities automatically through experience [9]. To define this artificial intelligence branch, we need three elements: Experience, Task, and Performance Measure. A machine learns when its performance in doing a specific task improves with experience. This process does not need any rules specification by the programmer. It can evaluate all the possible actions and choose the best one. To do so, we should change the philosophy used to define the task. The idea is to simplify the problem by providing some data and the expected output to the computer. In this way, the machine can learn and find the steps between input and output [7]. There are three basic machine learning paradigms: supervised learning, unsupervised learning and reinforcement learning.

Supervised Learning. In supervised learning, the machine receives as inputs some data each one correlated with a specific label. Two examples are classification and regression. For the former, the aim is to learn a function able to predict the tag of unknown data. Inputs can be mono-dimensional or multi-dimensional, where each dimension corresponds to an attribute. The result obtained as output is a discrete category chosen among two or more classes [7]. For example, classification can determine whether or not it will snow tomorrow (binary classification); or if the fruit is an apple, a pear, a cherry, a watermelon (multi-class classification). Regression differs because it predicts a real continuous value. For example, we can use a regression algorithm to predict the amount of snowfall.

Unsupervised Learning. In unsupervised learning, the computer does not receive an explicit label, but the expected output is a group or organization of the data. In this situation, the machine should give a program able to detect patterns in data [7]. An example of unsupervised learning is clustering, which aims to find the best way of grouping data. This paradigm is useful in marketing, for example, to subdivide all the possible targets into clusters and choose the best strategy for each of them.

Reinforcement Learning. In reinforcement learning, an autonomous agent must learn to perform a task by trial and error interactions with a dynamic environment. It uses positive or negative rewards but does not specify how to achieve

the goal. A first strategy finds among the possible behaviours the best performing. A second strategy estimates the possible actions according to the situation [9, 10]. For example, robotics applies this kind of learning.

2.3 Deep Learning

2.3.1 Basic concepts

Deep learning is a technique of machine learning able to solve intricate problems, subdividing complex concepts into simpler ones and representing them with a nested hierarchy of tasks with different levels of abstractions the computer can efficiently perform. This philosophy makes these systems able to operate in complicated real-world environments. Deep learning exploits artificial neural networks with multiple layers between the input and the output layer for learning called deep neural networks [9].

Perceptrons

To get started with neural networks, we introduce a kind of artificial neuron, called a perceptron. This concept was introduced in the 1950s by the scientist Frank Rosenblatt, who referred to Warren McCulloch and Walter Pitts. Understanding how perceptrons work is essential, so we describe them before proceeding with more complex definitions. A perceptron takes several binary inputs, each one correlated with a specific weight, which expresses the importance of that input to the output. It processes this information and produces a single binary output, 0 or 1 [11]. For example, in Figure 2.1, the perceptron receives three inputs x_1, x_2, and x_3 with their weights w_1, w_2, and w_3 and produces the output y.

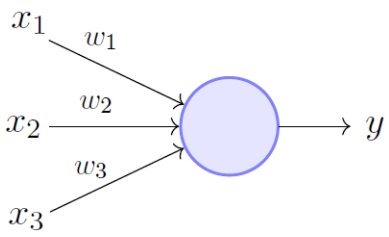


Figure 2.1. Perceptron Model by Minsky and Papert.

Rosenblatt introduces a simple rule to compute this output, comparing a threshold value, represented by a real number, with a weighted sum result. To clarify

this concept, we express it in algebraic terms, as shown in (2.1):

$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq threshold \\ 1, & \text{if } \sum_j w_j x_j > threshold \end{cases} \tag{2.1}$$

A practical example may help us to understand the relationship between inputs and weights. During summer, we could decide whether to go to the beach considering three circumstances as inputs of the perceptron:

1. Does anyone want to come with you? (x_1)
2. Is the weather windy? (x_2)
3. Is there much traffic? (x_3)

x_1 is equal to 1 if the answer is yes, and 0 if not, while x_2 and x_3 are equal to 1 if the answer is no, and 0 if it is yes. Each one of the previous factors could have different importance. We can model this importance by specifying a weight to each input to obtain different outputs. Suppose you love the sea, enough for you to go there also alone. Despite this, you hate wind, so this parameter is fundamental to make your choice. Moreover, the traffic information is more or less significant depending on the distance and the vehicle used to reach the beach. The perceptron used to model this kind of decision-making could have a weight $w_2 = 6$ for the weather, $w_1 = 2$ for the company, and $w_3 = 1$ for the traffic if we suppose you go to the sea by bike. Finally, assume we choose a threshold equal to 5 for the perceptron. With our weights, the perceptron outputs 1 when the weather is good and 0 vice-versa. It makes no difference to the output whether anyone wants to go or the traffic information. By varying the weights and the threshold value, we can get different models of decision-making. For example, assume we instead keep the threshold equal to 2. The perceptron would decide you should go to the sea on a windy day if anyone comes with you and there is no traffic. In fact, the weighted sum will be $2x_1 + 6x_2 + 1x_3 = 2 \times 1 + 6 \times 0 + 1 \times 1 = 3 > 2$. In other words, a lower value means a greater desire to go to the beach. We can weight our parameters according to our needs to obtain different results.

To increase the complexity of the model, we could add more layers of perceptrons, obtaining a structure called multilayers perceptron, shown in Figure 2.2. In this network, we have the first column of perceptrons called the first layer of perceptrons. It can make simple decisions by weighting the inputs. Each perceptron in the first layer has a single output, which is, in turn, an input for all the perceptrons in the next layer. The perceptrons of the second layer work in a more complex and abstract level than the previous one. They weight up the results from the first layer of the network and produce the outputs. Even more complex

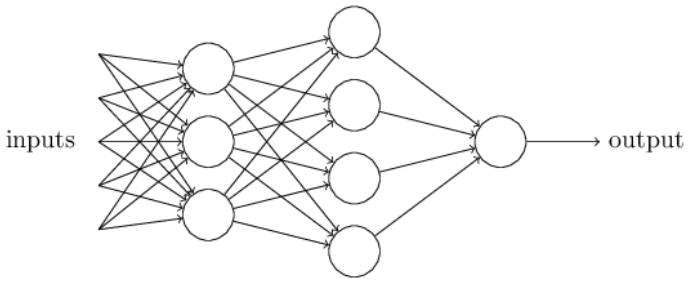


Figure 2.2. Multilayers Perceptron. (*Image from* [11])

decisions can be made by the perceptron in a third layer and so on. In this way, we obtain a sophisticated structure for decision making.

We could also simplify the previous equation describing perceptron through two notational changes. First of all, we can write the weighted sum $\sum_j w_j x_j$ as a dot product, $w \cdot x$, where w and x are vectors whose components are the weights and inputs, respectively. The second change is to move the threshold to the other side of the inequality and to replace it by the perceptron's bias b, where $b \equiv -threshold$. With these two changes, we can rewrite the perceptron rule as:

$$output = \begin{cases} 0 & \text{if} \quad w \cdot x + b \leq 0 \\ 1 & \text{if} \quad w \cdot x + b > 0 \end{cases} \tag{2.2}$$

We can think the bias as a measure of how simple it is to get the perceptron to output a 1. Or in more biological terms, it is a measure of how effortless it is to get the perceptron to fire. If this value is high, the perceptron will output a 1 very easily. But if the bias is a very negative value, then it is difficult for the perceptron to output a 1.

With this new notation, we could rewrite the preceding example in the following form:

- x is a vector containing the inputs, $\mathbf{x} = \begin{bmatrix} x_1 = 1 \\ x_2 = 0 \\ x_3 = 1 \end{bmatrix}$;
- w is a vector containing the weights, $\mathbf{w} = \begin{bmatrix} w_1 = 2 \\ w_2 = 6 \\ w_3 = 1 \end{bmatrix}$;
- b is equal to -5.

The output will be $2x_1 + 6x_2 + 1x_3 - 5 = 2 \times 1 + 6 \times 0 + 1 \times 1 - 5 = -2 < 0$. So the perceptron does not fire and outputs 0.

Sigmoid Neurons

In many modern works on neural networks, the sigmoid neuron is the most used neuron model. A neural network learns from parameters and gradually improves itself by varying weights. The learning process works if a small change in some weight or bias causes only a little corresponding change in the network output. We can describe this behaviour with the following scheme (Figure 2.3):

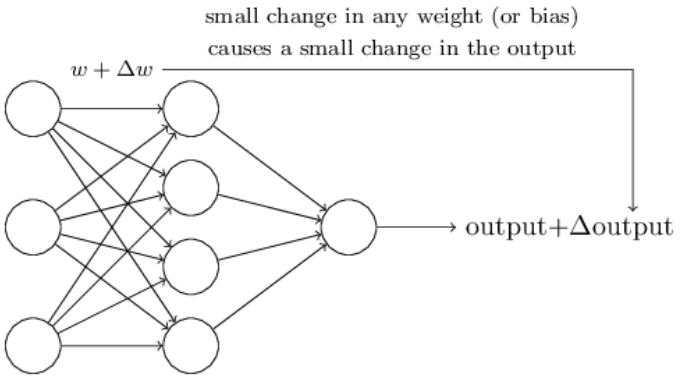


Figure 2.3. The key concept of the learning process. (*Image from* [11])

The perceptron, having only a binary value as output, does not work well enough in this situation because a little change in weights could change radically from 0 to 1 and vice-versa the result. This flip may change, in turn, the behaviour of the rest of the network in a very complicated and uncontrollable way. This problem belongs to the absence of an intermediate value for the perceptron output and stops a gradual improvement. This reason leads to the introduction of a new kind of artificial neuron called sigmoid neuron. Sigmoid neurons are comparable to perceptrons, but a network of sigmoid neurons can learn because a little change in their weights causes only a small change in their output. To describe a sigmoid neuron, we can refer to the same scheme of the perceptron, shown in Figure 2.1.

As a perceptron, the sigmoid neuron has n inputs, $x_1, x_2, ..., x_n$, but they are not binary and can also assume any value between 0 and 1. Each data in input has its weight w_i, and there is an overall bias b. Also, the output is not binary, but it is $\sigma(z)$, where $z = w \cdot x + b$ and σ is the sigmoid function, defined by:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.3}$$

So, given the inputs, $x_1, x_2, ..., x_n$, their weights, $w_1, w_2, ..., w_n$ and the bias b, we obtain:

$$output = \sigma(z) = \sigma(w \cdot x + b) = \frac{1}{1 + exp\Big(-\sum_j w_j x_j - b\Big)}. \tag{2.4}$$

If we suppose $z \equiv w \cdot x + b$ is a large positive number, then $e^{-z} \approx 0$ and $\sigma(z) \approx 1$. In other words, when $z = w \cdot x + b$ is a large and positive number, the output of the sigmoid neuron is approximately 1. On the other hand, if $z = w \cdot x + b$ is a very negative number, $e^{-z} \to \infty$, and $\sigma(z) \approx 0$. So the behaviour of a sigmoid neuron closely approximates a perceptron. The two models of artificial neurons differ a lot only when w is an intermediate value.

Graphically, we can represent sigmoid neuron and perceptron behaviour with the Sigmoid Function and the Step Function, when the former is a smoothed out version of the latter (Figure 2.4):

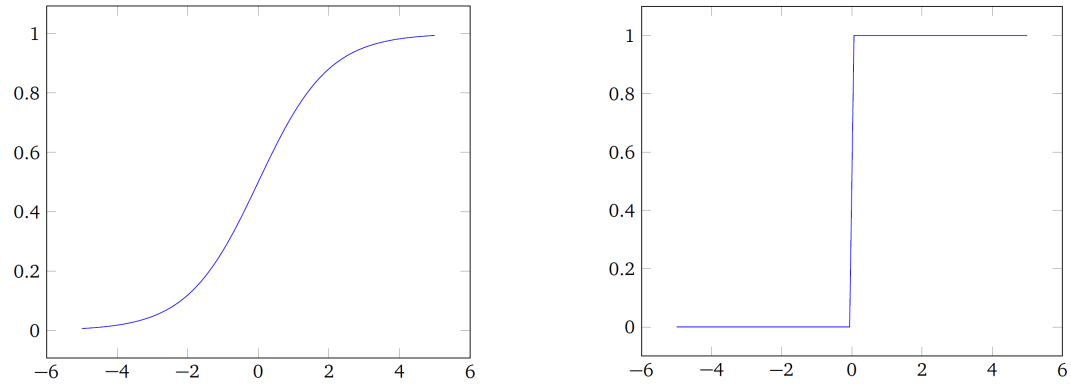


Figure 2.4. Sigmoid Function and Step Function. (*Image from* [11])

Observing these functions, the output response to small changes in inputs and bias for sigmoid neuron and perceptron may be more explicit. The smoothness of the Sigmoid Functions means that the output allows values between 0 and 1. The Step function, given an input, can instead provide only two values, 0 or 1, as the perceptron.

Sigmoid and Step functions are part of the so-called activation functions. Their role is to define the output of a node given a single or a set of inputs.

2.3.2 Neural Networks

To build a real application, we cannot use the described neurons singularly. We need to connect them, creating a collection of linked neurons called neural network. A neural network architecture contains three kinds of layers of neurons [11]: input layer, output layer, and hidden layer.

Input Layer. It is the leftmost layer in the network and includes the input neurons.

Output Layer. It is the rightmost layer in the network made by output neurons.

Hidden Layer/Layers. It is a middle layer of the network. Despite the term "hidden" sounds a little mysterious, it simply means that the neurons belonging to this layer are neither inputs nor outputs. In a neural network, we could have one or multiple hidden layers. For example, in the figure below (Figure 2.5), we have two hidden layers.

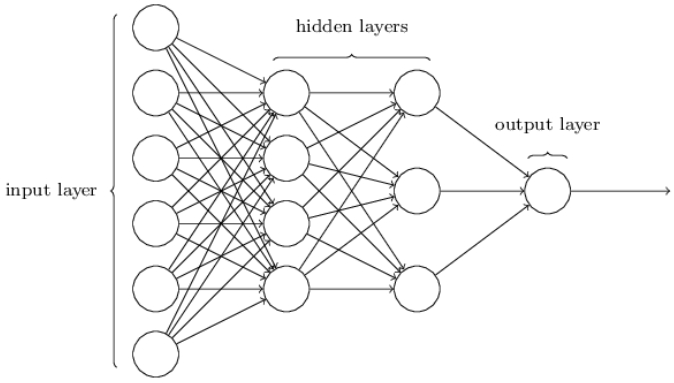


Figure 2.5. Neural network architecture. (*Image from* [11])

The design of the input and output layers of a neural network is straightforward. But we cannot say the same thing for the hidden layers because we cannot describe their design process with a few simple rules. For this reason, in recent years, researchers have defined many design heuristics for helping people. These heuristics avoid trying all the possible combination of hidden layers each time, but they provide an idea of a potentially correct network configuration for a particular problem. For example, heuristics can suggest how to trade off the number of hidden layers against the time required to train the network and so on.

We can divide neural networks into two main classes: feedforward or recurrent described briefly below.

Recurrent Neural Network. This first model of neural networks processes sequential data. A recurrent neural network makes possible feedback loops between neurons. These loops represent the influence of the present value of a variable on its value at a future time step. A neuron output only affects its input in the future, not instantaneously, so loops here do not cause problems [9, 11].

Feedforward Neural Network. These networks have no loops because they always feed forward the information. So, the output of a layer becomes the input

to the next one. The presence of loops could allow situations in which the input of the σ function depends on the output. For this reason, feedforward neural networks do not allow loops [11].

Recurrent neural networks have been less influential than feedforward networks, maybe because their learning algorithms are generally less powerful. But they are interesting because they are much closer than feedforward networks to the behaviour of our brains.

Learning process

Defined a specific architecture for our network, we need a data set to learn from, called training dataset, and a method to train the network for executing a precise task. For understanding how the learning process works, we have to describe three fundamental concepts:

- Cost Function;
- Gradient Descent;
- Backpropagation.

Cost Function. As already said, we need an algorithm able to find weights and biases such as the network output approximates the desired one, $y(x)$, for all training inputs x. We define a cost function to quantify how well our network is achieving this goal:

$$C(w,b) \equiv \frac{1}{2n} \sum_x (y(x) - a)^2. \tag{2.5}$$

In the equation (2.5): w indicates the set of all weights, b all the biases, n is the total number of training inputs, and a represents the vector of network predicted outputs and depends on x, w and b. We call C the quadratic cost function, also known as the mean squared error or just MSE. Observing our quadratic cost function, we see that $C(w,b)$ is non-negative since every term in the sum is non-negative. Moreover, the cost $C(w,b)$ becomes small, $C(w,b) \approx 0$, when the desired output $y(x)$ is approximately equal to the predicted one, a, for all training inputs, x. In other words, our training algorithm works well if it can find weights and biases so that $C(w,b) \approx 0$. On the other hand, a high cost means that the predicted output is not close to the correct one for many inputs [11]. In practice, the cost function represents the difference between the predicted output and the actual one. The quadratic cost function works perfectly well for understanding the basics of learning in neural networks, thanks to its smoothness. We could also choose other functions for computing the cost. Our training algorithm should find a set of weight and biases minimizing as much as possible this cost. To understand

how we can reduce the difference between the predicted and the expected output, we need to introduce the gradient descent algorithm.

Gradient Descent. Gradient descent is the process which allows us to move towards the cost function minimum [11]. In a simple case, we could be able to locate the minimum just observing the function graph. But in a complicated function of many variables, we cannot have a visual representation, so we need to find another solution. One possible option is calculating the minimum analytically using derivatives. This solution might work when C is a function of just one or two variables, but it could become a big issue when we have many more variables. When we work with neural networks, the cost functions depend on billions of weights and biases in a very complicated way. So for neural networks, also the calculus does not work. At the moment, we do not consider neural networks, and for understanding the algorithm, we imagine C as a function of just two variable, v_1 and v_2. Our aim is finding the global minimum of $C(v_1, v_2)$. Let's start imaging our function as a simple valley, and a ball rolling down the valley slope. We could randomly choose a starting point for the imaginary ball, and then simulate its motion as it rolled down to the bottom. We could do this simulation by computing derivatives of C. To do so, we assume it is possible to choose how the ball should roll. If we suppose to move the ball a small amount Δv_1 in the v_1 direction, and a small amount Δv_2 in the v_2 directions, C changes as follows:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2. \tag{2.6}$$

To ensure the ball rolling down into the valley, we have to make ΔC negative, choosing Δv_1 and Δv_2 opportunely. For understanding how to make this choice, we define a vector of changes in v, $\Delta v \equiv (\Delta v_1, \Delta v_2)^T$, and denote the gradient of C as the vector of partial derivatives:

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T. \tag{2.7}$$

With these definitions, we can rewrite the expression (2.6) for ΔC as:

$$\Delta C \approx \nabla C \cdot \Delta v. \tag{2.8}$$

We call ∇C the gradient vector because it relates changes in v to changes in C. Moreover, the equation lets us see how to choose Δv to make ΔC negative. In particular, we suppose to choose

$$\Delta v = -\eta \nabla C, \tag{2.9}$$

where η is a small, positive parameter known as the learning rate. Replacing Δv with this new definition in (2.8) we obtain: $\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \|\nabla C\|^2$. Because $\|\nabla C\|^2 \geq 0$, this guarantees that $\Delta C \leq 0$. So if we change v according to (2.9), C will always decrease, never increase, as we wanted. In this way, we can use (2.9) to define the "law of motion" for the ball in our gradient descent algorithm, by computing Δv and then move the ball's position v by that amount:

$$v \to v' = v - \eta \nabla C. \tag{2.10}$$

To make another move, we should use this update rule again. Doing so, over and over, we will decrease C until we reach the global minimum. In summary, the gradient descent algorithm works repeatedly computing the gradient, and gradually moving towards the global minimum of the function. To make the process described easier to understand, in Figure 2.6, we propose a visual representation of the algorithm:

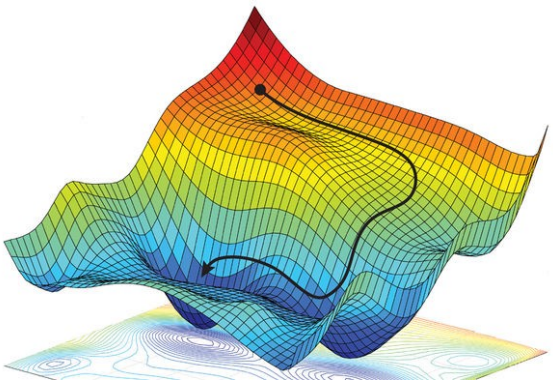


Figure 2.6. Visual representation of the gradient descent algorithm. (*Image from* [12])

The gradient descent algorithm works well only if we correctly choose the learning rate η, such that (2.8) remains a good approximation. The learning rate should be neither too big nor too small because in the former case we would obtain $\Delta C > 0$,while in the latter case the gradient descent algorithm would work very slowly since the changes in Δv would be tiny.

Up to now, we explained the gradient descent algorithm for a function of just two variable, but we could easily extend the process in cases of many more variables.

Applying gradient descent to neural networks means to find the weights w_k and biases b_l which minimize the cost equation (2.5). In other words, our "position" now has components w_k and b_l, and the gradient vector ∇C has corresponding components $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_l}$. Rewriting the gradient descent update rule in terms of

components, we obtain:

$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k}, \tag{2.11}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}. \tag{2.12}$$

We can use this rule to learn in a neural network by repeatedly applying it to find the minimum of the cost function. The problem here is the computational cost because to estimate the gradient ∇C we should compute the gradients ΔC_x separately for each training input x and then average them. But when the number of training inputs is very high, this can take a long time, and the network learns very slowly. To speed up learning, we can use another process called stochastic gradient descent. The idea is estimating the gradient ∇C by computing ∇C_x for a small sample of randomly chosen training inputs. By averaging over this small sample, we can estimate the actual gradient ∇C. In this way, we could exploit this estimation to speed up the gradient descent and the learning process. More precisely, stochastic gradient descent randomly picks out a small set m of random training inputs, $X_1, X_2, ..., X_m$, called mini-batch. The mini-batch size, m, should be large enough, allowing us to estimate the overall gradient ∇C by computing gradients just for the randomly chosen mini-batch and then averaging them, as shown below:

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}, \tag{2.13}$$

For the specific case of neural networks, given the weights w_k and the biases b_l, we can write the stochastic gradient descent learning process as follows:

$$w_k \to w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}, \tag{2.14}$$

$$b_l \to b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l}, \tag{2.15}$$

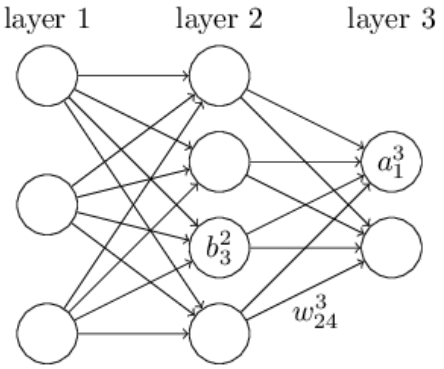
where the sums refer to all the training samples X_j in the current mini-batch. Then we choose another random mini-batch and train the network again. We repeat this process until there are no more training inputs, which is said to complete an epoch of training. At that point, we start over with a new training epoch. Of course, through a statistical process, we cannot obtain a perfect result. But it does not matter, because the focus is moving in a general direction for decreasing the cost function C, we do not need an exact computation of the gradient. In practice, stochastic gradient descent is a commonly employed powerful method for learning in neural networks.

Backpropagation. Backpropagation is a fast algorithm for computing the gradient of the cost function. It was introduced in 1986 by David Rumelhart, Geoffrey Hinton, and Ronald Williams, and today represents a basic concept of learning in neural networks. According to [13], backpropagation "repeatedly adjusts the weights of the connections in the network to minimize a measure of the difference between the actual output vector of the net and the desired output vector". In other words, backpropagation aims to minimize the cost function by adjusting weights and biases. In this way, the backpropagation algorithm remodels the overall behaviour of the network.

Useful notations. Before discussing backpropagation, we start with some useful notations [11]. We will use:

- w^l_{jk}, the weight from the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer;
- b^l_j, for the bias of the j^{th} neuron in the l^{th} layer;
- a^l_j, for the activation of the j^{th} neuron in the l^{th} layer.

For clarity, in the following diagram, we show an example of the described notations:



With these notations, we can write the activation equation as:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right), \tag{2.16}$$

where the sum is over all neurons k in the $(l-1)^{th}$ layer. Often we will use the compact form $a^l_j = \sigma(z^l_j)$, where $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$ is called the weighted input to the activation function for neuron j in layer l. So, the activation in the current layer is related to the activations in the previous layer.

Preliminary assumptions about the cost function. For backpropagation to work, we need to make two main assumptions about the form of the cost function [11]. To do so, we use as an example again the quadratic cost function, defined in (2.5). With the new notation described, the equation becomes:

$$C(w,b) \equiv \frac{1}{2n}\sum_x (y(x) - a^L(x))^2, \tag{2.17}$$

where L denotes the number of layers in the network; and $a^L = a^L(x))$ is the vector of activations output when x is input.

For applying backpropagation, we need to assume as follows:

1. We can write the cost function as an average $C = \frac{1}{n}\sum_x C_x$ over cost functions C_x for individual training examples, x. We need this assumption because for simplicity the algorithm computes the partial derivatives $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$ for a single training example and then recover $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ by averaging over training examples;

2. We can write the cost as a function of the outputs from the neural network: $C = C(a^L)$

The quadratic cost function satisfies both the assumptions.

The backpropagation algorithm. Backpropagation aims to understand how changing the weights and the biases to modify the cost function [11]. To do so, we need to introduce an intermediate quantity, δ^l_j called the error in the j^{th} neuron in the l^{th} layer, defined as follows:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}. \tag{2.18}$$

We denote δ^l the vector of errors connected with layer l. Backpropagation allows us to compute first δ^l for each layer, and then relate those errors to the partial derivatives, $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$.

For simplicity, we first describe the algorithm considering a single training example, $C = C_x$ [11]:

1. **Input** x**:** set the corresponding activation a^1 for the input layer.

2. **Feedforward:** for each layer, $l = 2, 3, ..., L$, compute z^l and $a^l = \sigma(z^l)$.

3. **Output error:** compute the output error vector δ^L.

4. **Backpropagate the error:** for each layer, $l = L-1, L-2, ..., 2$, compute δ^l, which is a function of the error in layer $l+1$.

5. **Output:** the gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

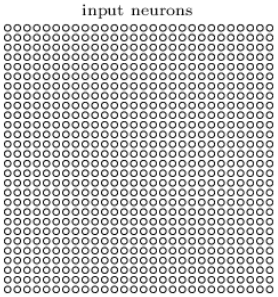
The algorithm shows why we call the process backpropagation. We compute the error vectors backwards, starting from the final layer. We can combine the algorithm with the stochastic gradient descent. To do so, we have to subdivide the input data in mini-batch of m training examples and, for everyone, apply the following steps for each epoch of training [11]:

1. **Input a set of training examples**
2. **For each training example** x**:** set the corresponding input activation $a^{x,1}$, and performs the following steps:
   - **Feedforward:** for each layer, $l = 2, 3, ..., L$, compute $z^{x,l}$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error:** compute the vector $\delta^{x,L}$.
   - **Backpropagate the error:** for each layer, $l = L-1, L-2, ..., 2$, compute $\delta^{x,l}$.
3. **Gradient descent** for each layer, $l = 2, 3, ..., L$, updating the weights and the biases according to the corresponding rules, (2.14) and (2.15).

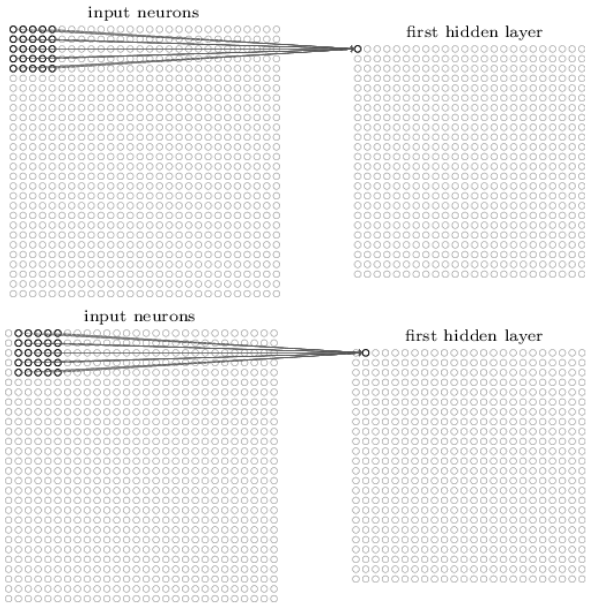
2.3.3 Convolutional Neural Network

LeCun et al. introduced Convolutional neural networks in 1998 [14]. Convolutional neural networks have a singular architecture which makes them particularly able to work with images, exploiting their spatial structures, and fast to train. This fastness allows us training deep and many-layer networks. Consequently, this kind of networks reaches remarkable results for image classification and image recognition. Convolutional neural networks employ three basic concepts: local receptive fields, shared weights, and pooling.

Local receptive field. In a fully-connected network, where every neuron links to every neuron in adjacent layers, we depict the inputs as a vertical line of neurons. Instead, we can imagine a convolutional neural network input as a matrix of neurons, having the same number of rows and columns of the number of pixels corresponding to the image width and height. The neurons values correspond to the intensities of the pixels of the image. For example, considering a 28 x 28 pixels image as in [11], we have:



As in a classic neural network, we connect the inputs to a layer of hidden neurons. In this case, we will not connect every input pixel to a hidden neuron, but we will group pixels in small regions of the input image. We link all the input neurons belonging to an input layer region to a neuron in the first hidden layer. We call the chosen region the local receptive field for a specific hidden neuron. We assign a weight for each connection between an input neuron and the hidden one, and an overall bias for each hidden neuron. In other words, a specific hidden layer learns about its local receptive field. The local receptive field is a window which slides, by one neuron at a time, across the entire input image, and each step associates a new region to a hidden neuron in the first hidden layer. We will proceed in this way until we cover all the input neurons. It is possible to set a different stride length, moving the local receptive field n pixels to the right (or down). If we suppose to start from the top-left corner and to move by one neuron at a time, the first two steps are the following:

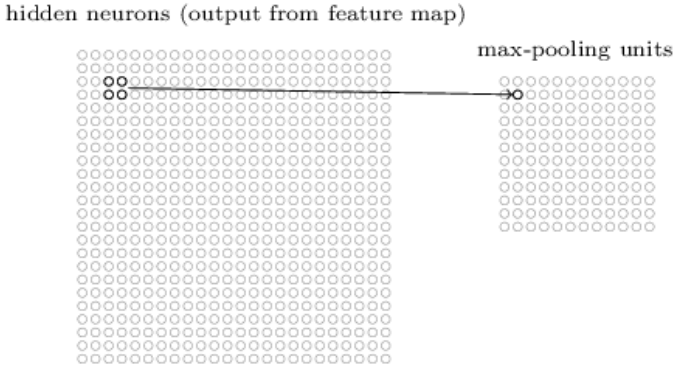


Shared weights and biases. We said that a hidden neuron has a unique bias and a weight for each input connection. In an algebraic form, for the j, kth hidden neuron the output is:

$$\sigma\left(b + \sum_{l=0}^{n}\sum_{m=0}^{n} w_{l,m} a_{j+k,k+m}\right), \tag{2.19}$$

where σ is the neural activation function; b is the bias shared by all the previous layer neurons; $w_{l,m}$ is an $(n+1) \times (n+1)$ matrix of weights; and $a_{x,y}$ is the input activation at position x, y. The operation in (2.19) is also known as a convolution, from which the name convolutional neural networks. We can write the equation as $a^1 = \sigma(b + w * a^0)$, where $*$ represents the convolution operation; a^0 and a^1 are the sets of input and output activations from one feature map, respectively. Since weights and bias are the same for every first layer hidden neurons, the previous expression means that all these neurons detect precisely the same feature, for example, an edge or a specific shape, but at different locations in the input image. For this reason, we call the *feature map* the one from the input layer to the hidden layer, defined by the so-called shared weights and the shared bias [11]. In other words, we can say that these elements represent a kernel or filter. The simple network described can detect only a single feature, but working with images, we need many more feature map to consider a convolutional layer complete.

Pooling layers. We have just described convolutional layers, but convolutional neural networks can also include pooling layers. They usually follow convolutional layers for simplifying their information. A pooling layer takes the convolutional layer output and generates a condensed feature map, which is a feature map with a lower dimension. A known procedure is max-pooling which outputs the maximum activation of the $n \times n$ input region [11]. As an example, if $n = 2$, we have:



If there are many features, we have to apply max-pooling to each feature map separately. Another approach is L_2 pooling, which outputs the square root of the

sum of the squares of the activations in the $n \times n$ region. Both the methods aim to compress information from the convolutional layer.

Summing up, convolutional neural network exploits convolutional layers to extract features from the input image. Then, we can compress the information through pooling layers and generate the desired output. As we have just explained, the convolutional networks architecture is different from the fully connected one described in the previous sections. However, we can adapt all the concept introduced until now, such as gradient descent and backpropagation, to this architecture.

2.3.4 Neural Networks in practice

After understanding how neural networks work, we give some notes on how we can treat them in practice. Enabling the network to execute a specific task, for example, the classification of fruits from an image, we need to create a big dataset of fruit images. Then, we have to split the original dataset into three different groups, useful for making our network able to learn. We use almost all the data to create a training set and divide the residual part, in turn, into the so-called validation set and test set. The next step is defining the details, as the number of hidden layers for the architecture, and the *hyper-parameters*. These are the parameters that do not change during the process, such as learning rate, batch size, and the number of entire iterations on the training set, called epochs. Now, our network is ready to learn. To do so, we need an iterative process: we first set weights and biases randomly and feed our network with the training set images; it predicts an output and compute a cost function to understand how this output is close to the correct one. Then, through gradient descent and backpropagation processes, the network adjusts weights and biases to minimize the cost function. This process should proceeds until the model reaches a specific level of accuracy, that is when the difference between the predicted output and the actual one is low enough. In practice, the network repeats the process n times, where n is the number of epochs set at the beginning. It means that we should try different values for hyper-parameters until we reach a good result. We can run the model at the end of each epoch of training on the validation set, to evaluate it. Finally, when we think the network has learned enough, we can test its behaviour. We examine the model using images which it has never seen before, belonging to the test set. In this way, we can understand if he is properly working, and it can generalize what it has learned on never seen images.

Training a neural network is a complex process, and some problems can occur before finding the optimal combination of parameters. The most frequent issue and the most troublesome to avoid in neural networks is **overfitting** or overtraining. It happens when our model works well for the training data but fails to generalize to new situations. This problem is maybe due to the lack of training data which

leads the model to work only in that specific situation [11].

There are different examples of overfitting situations. For example, if we run our model for many epochs, after n epochs the validation accuracy may saturate to a definite value even if the validation loss keeps decreasing. So it seems the network is still learning, but it is only an illusion. Another overfitting situation occurs when the classification accuracy on the training data grows to 100% while the test accuracy presents a lower value. It means the network is learning singularities of the training set, but it cannot generalize the main ones [11]. There are many techniques to prevent overfitting, such as:

- **Increasing the amount of training data**, adding new items or using **data augmentation:** it increases the size of the training set by applying random transformations on the images, such as changing colour properties, rotation, scale, and flipping [15];
- **Early Stopping:** we employ the validation set to evaluate the validation accuracy after each epoch and when it saturates we stop the process [11];
- **Regularization:** it is a process to reduce overfitting keeping the size of the network and the training set fixed, by adding an extra term to the cost function called the regularization term [11];
- **Dropout:** it is a particular technique for regularization, because it does not modify the cost function, but the network itself, by deleting a set of hidden neurons randomly and temporarily for each mini-batch [11].
- **Batch Normalization:** this technique normalizes the input layer by adjusting and scaling the activations to have a mean output activation of zero and standard deviation of one [16]. Batch normalization helps to speed up the learning process, and it has a slight regularization effect, so it reduces overfitting.

2.4 Deep Learning and Computer Vision

Computer Vision aims to build autonomous systems able to perform some of the human visual system tasks [17]. The development of CNNs has influenced the Computer Vision field and has led to a paradigm change for some tasks. Since neural networks are trained rather than programmed, applications require less expert analysis and exploit the enormous amount of data available [18]. Moreover, CNN models are more flexible because they can be re-trained using a custom dataset for another use case, while Computer Vision algorithms are domain-specific [18]. Deep Learning improves many computer vision problems, such as object detection, motion tracking, action recognition, human pose estimation, image classification, and

semantic segmentation [19, 18]. These progressions are due to improvements in computing power, memory capacity, power consumption, image sensor resolution, optics, and data availability, which allow Computer Vision engineers to achieve greater accuracy with Deep Learning techniques [18].

Our system, detailed in Chapter 4, leverages on semantic segmentation resulting from a specifically trained Convolutional Neural Network. This approach has proven more performative, in terms of accuracy and generalization, when implemented through a neural network instead of traditional Computer Vision algorithms.

2.4.1 Semantic Segmentation

Semantic segmentation is a significant task in computer vision and employs convolutional neural networks, allowing deep learning to surpass other approaches. To get started, we propose an introduction of the topic referring to [20]. Semantic segmentation is a technique for associating a label or a class with every pixel in an image, based on what it represents. It is also known as pixel-wise semantic segmentation since it works pixel by pixel. This algorithm aims to recognise a set of pixels relating to a specific object, dividing an image in two or many category groups. For example, in a bedroom image, we can segment the bed, nightstands, wardrobe, carpet, lamps, and so on. Semantic segmentation can be a beneficial alternative to object detection because it allows the object of interest to occupy multiple areas in the image at the pixel level. This technique identifies with precision irregularly shaped objects, in contrast to object detection, where targets must fit within a bounding box. Due to its characteristics, semantic segmentation is more precise than object detection and, consequently, helpful for many applications which need high accuracy, such as:

- **Autonomous driving:** for separating the road from obstacles like sidewalks, pedestrians, poles, and other cars;
- **Industrial inspection:** for detecting defects in materials;
- **Satellite imagery:** for identifying different types of land, such as mountains, rivers, lakes, deserts;
- **Medical imaging:** for analyzing and performing diagnostic tests, automating the process and reducing the time required;
- **Robotic vision:** for identifying and navigating objects and paths.

According to [20], the training process of a semantic segmentation network to classify images applies the following steps:

1. Create and analyse a collection of pixel-labelled images;
2. Create a semantic segmentation network;
3. Train the network to classify images into pixel categories;
4. Assess the network accuracy through a standard loss function called cross-entropy, comparing each pixel of the output with the corresponding pixel in the actual segmentation image.

To describe the task more in detail, we follow the discussion of [15]. In particular, semantic segmentation aims to take a W x H x 3 RGB image or a grayscale image of size W x H x 1 and output a W x H matrix containing the predicted class identifiers corresponding to all the pixels. As an example, we show an image (Figure 2.7), where, for clearness, the segmentation map as a lower number of items than the image pixels:

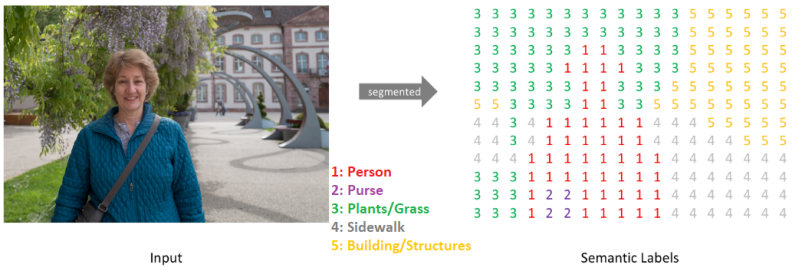


Figure 2.7. Input image (left) with its semantic labels matrix (right). (*Image from* [21])

Convolutional neural networks for segmentation

We can use a convolutional neural network for semantic segmentation, both the input and the output will be images. We need to build a model with several convolutional layers, non-linear activations, pooling layers and batch normalization, a technique speeding up the learning process adapting the previous layer output distribution so that the subsequent one could process it efficiently. The initial layers learn low-level concepts, such as edges and colours, while the deep level layers learn higher-level concepts, such as different items. Thus, the lower level neurons contain information about a small region on the image, while at a higher layer, they hold data for a large image area. For semantic segmentation, we need to preserve the spatial information hence no fully connected layers are used. Usually, an encoder-decoder structure [15] is used, shown in Figure 2.8. First, the encoder downsamples the input using convolutional and pooling layers. The convolutional

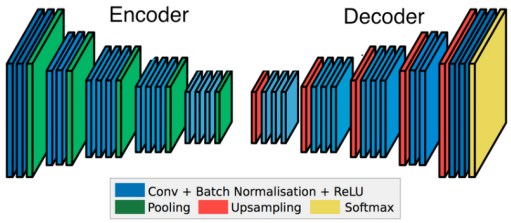


Figure 2.8. Encoder-Decoder Architecture. (*Image from* [22])

layers exploit batch normalization to improve the process and the Rectified Linear Unit (ReLU) activation function. The ReLU is null for negative values and has a linear trend for positive ones. This activation function is the most commonly used in deep learning models because it has two important properties: it does not reach saturation thanks to its linear trend and avoids a phenomenon called vanishing gradient, that is, where there are many layers, after some error backpropagations, the derivative could have no more relevant values [23]. The encoding procedure outputs a low-resolution tensor (a multidimensional array) with high-level information. Then the decoder performs the opposite process through deconvolutional and upsampling layers: increase the resolution again and obtain low-level data. When we train this architecture for semantic segmentation, the encoder outputs a tensor containing information about the objects and their shape and size. The decoder takes this information and produces the segmentation maps. Before computing the result, the decoder uses a final layer with a Softmax activation function. This Softmax layer normalizes the output to a probability function so that all the values sum to 1 [11].

Skip connections. If we use simply the encoder-decoder structure, we could loss low-level information, producing inaccurate boundaries in segmentation maps. To solve this problem, we could add skip connections, which allow the decoder to access directly to the necessary information generated by the encoder layers. In this way, we concatenate the intermediate encoder outputs and the decoder intermediate layers inputs at appropriate positions [15], as shown in Figure 2.9.

Transfer learning. Transfer learning is a method which allows us to exploit the convolution layers of pre-trained models in the encoder layers of the segmentation model. It is a popular choice in computer vision because these models contain meaningful information and avoid a custom model, which could be full of error and less performant [15].

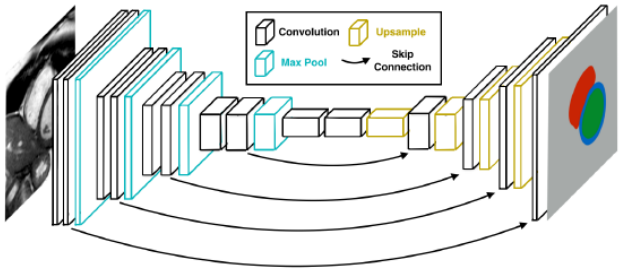


Figure 2.9. Encoder-Decoder with skip connections. (*Image from* [24])

Models and Architectures. There are several models available for semantic segmentation. We should choose the model architecture depending on the use case. In this paragraph, we give an overview of semantic segmentation models and architectures mentioned in Chapter 4.

- **ResNet:** This model proposed by Microsoft has several applications as a pre-trained model. ResNet has a large number of layers along with residual connections which make its training possible [15].
- **VGG-16:** Before ResNet, VGG was the standard pre-trained model in for a large number of applications. This model proposed by Oxford has fewer layers than ResNet, so it is much faster to train but less accurate [15].
- **MobileNet:** Google proposed this method and optimised it for having a small model size and faster inference time, that is the required time to predict a result using a trained algorithm. In can run on mobile phones but due to the small size, it could have low accuracy [15].

After selecting the base network, we have to choose the architecture for segmentation. We consider three popular segmentation models:

- **PSPNet:** The Pyramid Scene Parsing Network learns better global context representation of a scene. First, the image is passed to the base network to get a feature map. Then the feature map is downsampled to different scales. Then, the pooled feature maps are then convoluted, upsampled to a specific scale and concatenated together. Finally, another convolution layer produces the final segmentation outputs [25]. Here, the smaller objects are captured well by the features pooled to a high resolution, whereas the features pooled to a smaller size detect the large ones (Figure 2.10). It requires a large model input size, around 500x500.

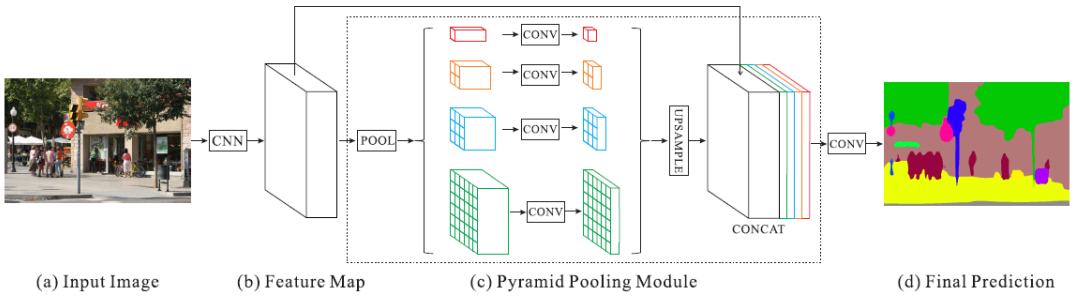


Figure 2.10. PSPNet Architecture. (*Image from* [25])

- **UNet:** The UNet architecture adopts an encoder-decoder framework with skip connections (Figure 2.11). The encoder and decoder layers are symmetrical to each other [26].

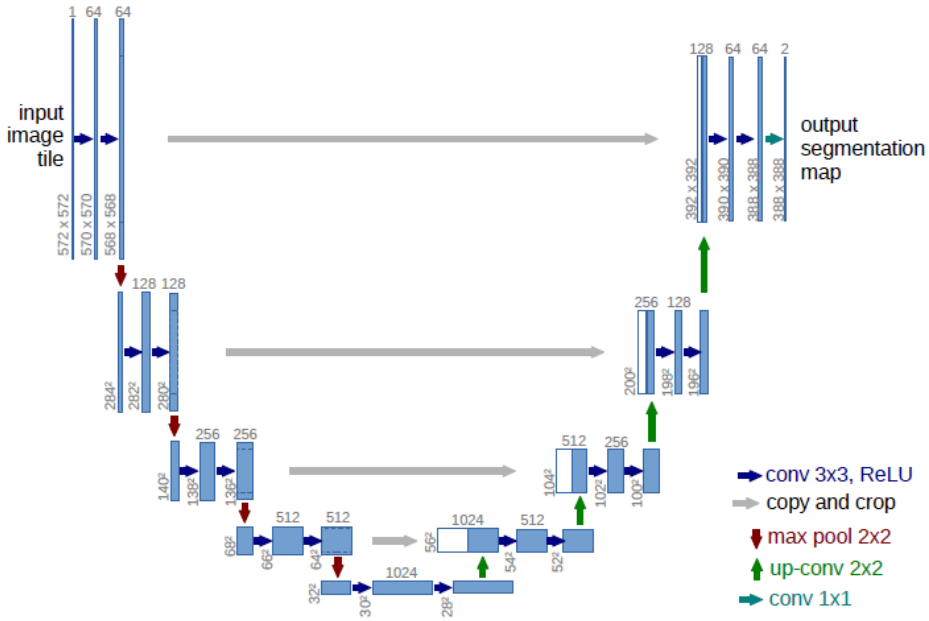


Figure 2.11. UNet Architecture. (*Image from* [26])

- **SegNet:** The SegNet architecture adopts an encoder-decoder framework without any skip connections (Figure 2.12). The encoder and decoder layers are symmetrical to each other. The upsampling operation of the decoder layers exploits the max-pooling indices of the corresponding encoder layers [27].

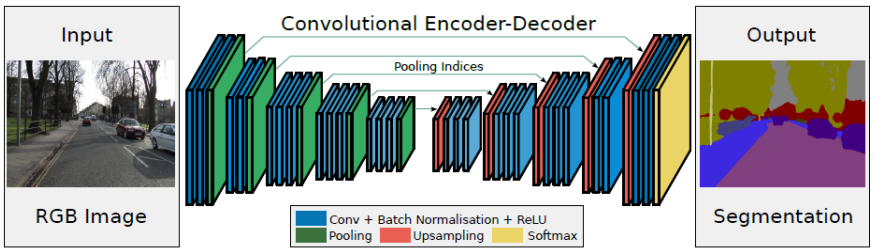


Figure 2.12. SegNet Architecture. (*Image from* [27])

For images containing indoor and outdoor scenes, PSPNet is the best choice, as the objects are often present in different sizes. For medical images, UNet works better because skip connections allow this model to capture also tiny details. UNet could also be useful for indoor/outdoor scenes with small size objects. For simple datasets, with large size and a small number of items, UNet and PSPNet could be too slow. Here simple models such as SegNet could be sufficient [15].

Chapter 3

Literature analysis

3.1 Introduction

Estimating the 6D position of an object from an image is a central problem in Computer Vision. It concerns many domains such as robotics, autonomous driving, medicine, industrial inspection, and virtual/augmented reality applications, extensively used in the entertainment and medical care industry [28, 29, 30]. The problem itself is simple and consists of determining the 3D rotation and translation of an object whose shape is known relative to the camera, using details observable from the reference image. However, achieving a solution to this problem is difficult [28]. Due to auto-occlusions or symmetries, the objects cannot be clearly and unequivocally identifiable, assuming an ambiguous position. Moreover, the image conditions are not always optimal in term of lighting and occlusions between the objects represented in the picture [30, 31, 32]. In these situations, it is often necessary to add an earlier stage of semantic segmentation or object detection for identifying the area of the image which contains the object, before estimating its position.

Although the researchers have studied this problem for many years, it experienced a kind of rebirth with the advent of deep learning. Old pose estimation methods were based on geometrical approaches, trying to establish correspondences between 3D models and the corresponding 2D images of objects using local features annotated manually. With texture-less or geometrically complex objects, it is not easy to select local features. In these cases, despite matching takes much time, it may fail and providing a result that is not always accurate [33].

In opposition to these methods, the researchers introduced other strategies, relying on 2D object representations from different points of view, and comparing them with the original image to establish the position and orientation. These methods are very susceptible to variations in lighting and occlusions even if they

can manage texture-less objects, and require many comparisons to reach a certain accuracy level, increasing the execution time [29]. With the diffusion of Deep Learning, researchers have introduced new strategies for achieving the goal, improving traditional methods, making them more efficient and performing. The basic idea of systems involving convolutional neural networks is to learn a mapping function between the image, and the object 6D position, from images having three-dimensional position annotations. These methods can reach very high levels of precision but need many data to train the network accurately and to be able to work well in real cases. Alternatively, we can use neural networks to execute the most critical steps of traditional methods. Combining them, we can join the advantages of the various strategies in the final solution [33].

Referring to the methods mentioned above this literature review focuses on the classification of 6D position estimation methods from a single RGB image into three categories, described in detail in the following sections: template-based methods (Section 3.2), feature-based methods (Section 3.3), and direct prediction or learning-based methods (Section 3.4).

The methodology used for selecting the articles involves different strategies to establish among the hundreds of existing papers about the topic the ones suitable for the review. In general, we selected the most recent only papers, from 2016 on. A second discard criterion concerns input data. We decided to consider only those systems having a single RGB image as input and optionally a 3D model of the object. Multiple or RGBD cameras are hard to use, they are expensive, the calibration process becomes very complicated, and the equipment may become too heavy. As a result, we decided to ignore articles including stereo images, RGB-D images or data from sensors, such as LIDAR sensors since they supplement the input with depth information reducing the problem complexity.

Specifically, we searched on Google Scholar, IEEE, ACM Digital Library, Springer and Science Direct using keywords such as "6D pose estimation from RGB images", "Viewpoint prediction", "Position and Orientation estimation", "3D point anchoring", "Automatic Registration". From this first research, we selected 27 articles. Then, starting from the papers bibliography and their citations, we obtained 29 new references. To further improve the set of studies to investigate, we considered a literature review published in January 2020 [34], reaching a total of 70 articles. This review differs from the one in issue for two main reasons:

1. It is more wide-ranging, as it considers as input depth information, both Mono and Stereo RGB images, RGB-D images, data from LIDAR sensors;

2. It classifies the articles, according to the mathematical model used, in five categories: classification-based methods; regression-based methods; methods combining classification and regression; template-matching based methods; point-pair feature matching based methods.

Finally, although all the articles examined satisfied all the requirements, we decided to make a further screening. To do so, we based on the number of citations of each paper, referring to the number reported by Google Scholar. In general, we considered articles with less than 5 citations not relevant or probably related to a narrower research field.

3.2 Template-based methods

These methods include a first stage, executed off-line, which build a template database from a 3D model of the object. This database consists of a set of synthetic renderings, obtained by varying position and orientation. A result is a group of patches from different points of view. We could imagine them as distributed over a virtual sphere surrounding the 3D model of the object. The second stage is a test phase, executed on-line to establish the 6D position. So the current image is compared with all the patches belonging to the database generated in the previous step employing a sliding window algorithm. These systems use a similarity value to compute the best match, chosen by the method itself [33, 31, 35, 34]. We show an example of how these methods work in Figure 3.1.

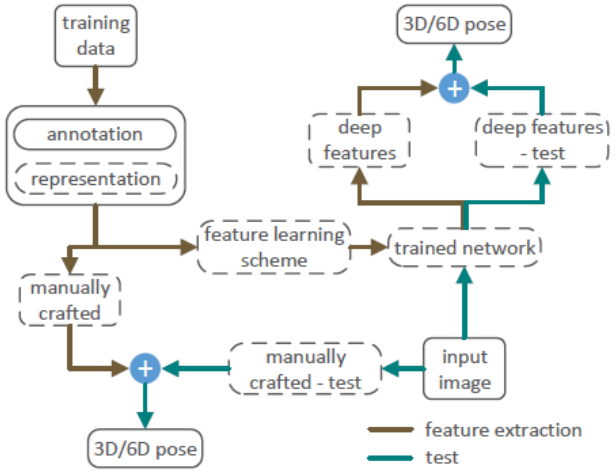


Figure 3.1. Schematic representation of template matching methods: the dashed-line represents a step not employed by all the systems. (*Image from* [34])

Advantages:

1. They work well in case of texture-less objects [29, 33];
2. If the database is exhaustive, they can achieve high accuracy [36].

Disadvantages:

1. They are very sensitive to variations in lighting and occlusions between items, as these circumstances affect the rate of similarity, which is very low when the lighting is low or when the object is occluded [29, 33, 37];
2. The execution speed is inversely proportional to the number of elements belonging to the template. However, this number is directly proportional to the accuracy of the method. We need a rich set of images to cover as many positions of the object as possible and to have a high probability of obtaining the correct pose. Therefore, a trade-off between performance degradation and desired accuracy is required. Many approaches implement changes to the cost functions by adding ad-hoc terms to solve these problems [38, 36].

In [39] Ulrich et al. compare the test images with the template set basing on edge features. This method estimates first the discrete position. Then it is refined using the 2D match and the corresponding 3D camera position using the Levenberg-Marquardt algorithm (LMA). The system receives a 3D CAD model of the object as input from which it automatically generates a hierarchical model. The user can specify a range of possible object positions in front of the camera. The generation of the hierarchical model includes only important geometric information for the recognition process. The main task is to detect a hierarchy of 2D views, for finding the object in the image in an efficient way. During the recognition process, the system evaluates each candidate and calculates the best position employing a geometric distance measurement. The authors tested the model on metal components with unrepresentative textures and shiny materials. Although exhaustive research ensures to find the 2D view with the best match, the process is too slow for real-time applications.

To reduce the execution time problem, typical of template-based methods, Konishi et al. [40] introduce a new feature and efficient Hierarchical Pose Trees (HPT) search algorithm. The proposed feature called Perspectively Cumulated Orientation Feature (PCOF) uses orientation histograms extracted from images with randomly generated 2D projections from 3D CAD data; the template applies this feature to handle a specific range of 3D object positions. Hierarchical pose trees (HPT) are constructed by clustering the 3D object poses and reducing templates resolution. They accelerate position estimation by using a coarse-to-fine strategy with a pyramid structure. The method is robust to position changes; it decreases the number of templates without loss of accuracy and reduces execution time. However, it requires a position refinement step, done using a Perspective-n-Point (PnP) algorithm, which calculates the 6D position from matches between 2D features on the test image and 3D points on the CAD model.

In [41] the authors propose a method for estimating the 3D position of textureless objects given coarse position information by a detector. The definite position is then estimated using edge matches, and the similarity measurement is encoded using a previously calculated linear regression matrix, learned from examples rather than being calculated analytically. The template database is constructed by sampling the various viewpoints from the object's viewing sphere, as shown in Figure 3.2. For each position, a specific linear predictor is trained, which reduces

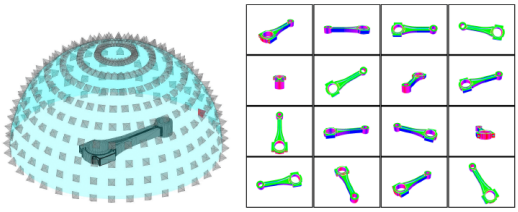


Figure 3.2. Templates generation. (*Image from* [41])

the distance function between the object's edge-map and the selected pose. For the distance between edges, the authors introduced the Real-time Attitude and Position Determination (RAPID) algorithm, in a basic version, RAPID-LR, and in an improved one, RAPID-HOG.

6D position estimation strategies are fundamental for tracking. We use pose estimation for tracking initialization and pose recovery when the algorithm loses the object due to occlusions or when it comes out of the camera's point of view. For this purpose, in [42], a new approach is proposed using a segmentation strategy based on a consistent local colour histogram. These descriptors can be trained on-line in a few seconds by moving the object in front of the camera. This method has good performance with cluttered backgrounds, heterogeneous objects and occlusions; it uses both real and synthetic data for training and does not require any refinement stage.

As mentioned above, older methods apply geometric approaches. In this context, the method proposed in [43] from an image and a 3D model, generates patches at various scales and renders for different rotation values respectively. It applies colour transformation and vectorization to images for obtaining a more compact representation for the best match calculation.

In [28] Muñoz et al. use a new approach employing the Cascade Forest Template (CFT), which combines the advantages of template-based and part-based methods. The 3D model presented as a set of viewpoint-dependent templates, but the template's similarity function, as in [41], is trained from examples rather than being calculated analytically. They use regression forests for each template to learn the misalignment between the initial layout and the current one. This method is

suitable for complex texture-less objects when the distribution of brightness over the surface makes points of interest or appearance-based descriptors calculation complex. Figure 3.3 shows an overview of the method.

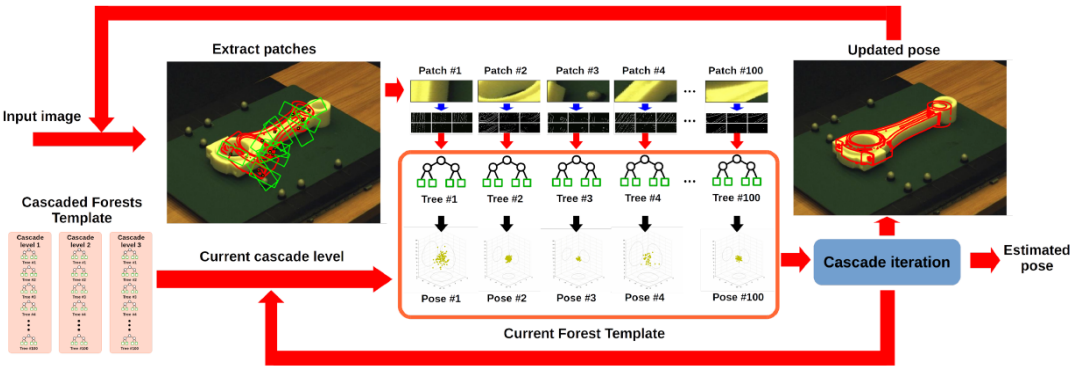


Figure 3.3. Pose estimation using CFT. (*Image from* [28])

Unlike most methods that try to recover the position of a known instance, called instance-based methods, in [44] the authors follow a different approach, as it is a category-based method. It works at the level of object categories trying to estimate the position of unknown instances. The procedure involves two steps: it first learns the template of an object category, assuming the training images have the bounding boxes of the objects annotated. Then the position of the camera is estimated using a Structure from Motion (SfM) method. For each viewpoint, they derive a template from the contours detected inside the bounding box for that point of view. The bounding boxes are normalized, assuming the same size as the template, the edges are copied to the template and averaged: for each pixel of the template, the average frequency of falling on an edge is computed to obtain a shape map. In the second step, the system performs shape matching between all the edges in a given image and the template. For matching, the authors introduce features called Bags Of Boundaries (BOB), which look for matching on a summary of edges, rather than on each of them. The goal is to find the best match between edges set and the template under an arbitrary projection (rotation, translation, 3D scale). The parameters of this projection are involved in the estimation of the final pose.

One of the problems to face in applications requiring the exact position of an object is the management of symmetrical objects. Identical views could generate ambiguities in the estimation process. In [45] Corona et al. attempted to address this problem through appropriate strategies. The inputs are an object, unseen in the training phase and its CAD model, and the objective is to calculate its position in the image by looking for a match between it and a rendered view

of the CAD model itself. Instead of simple RGB renderings, the system uses renderings of depth maps to overcome discrepancies between the real image and the texture-less CAD model. As mentioned earlier, the most recent methods use neural networks to make the algorithms adopted more efficient and performing. In this case, the neural network receives an RGB image and a depth map for each viewpoint, corresponding to the model renderings. Two branches are then involved: one for the object pose estimation, and one for the CAD model symmetry orders computation. The authors introduce a particular loss function to manage the complexity of item having rotational symmetry.

In [46] Massa et al. divide the process into an off-line and an on-line stage for which the same neural network, CaffeNet, is used. The former obtains features from CAD model rendering, consisting of images of the object from multiple views. The latter extracts additional features from the image and then compares them with those calculated off-line to obtain the matching one. Figure 3.4 shows a system overview.

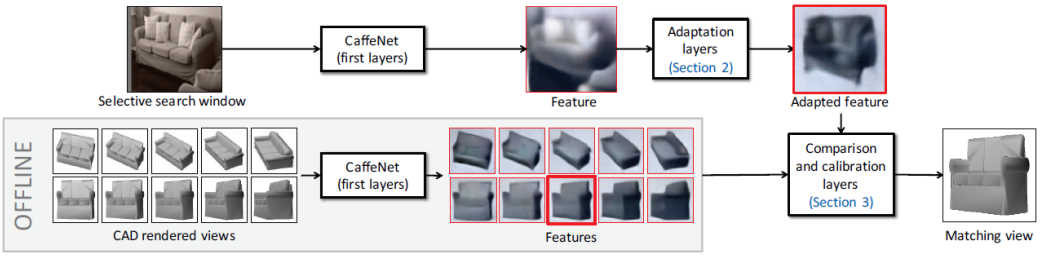


Figure 3.4. CaffeNet system overview. (*Image from* [46])

3.3 Feature-based methods

Methods in this category take advantage of local features (keypoints, grey values, edges or intersections of straight lines) extracted from the regions of interest or all pixels in the image and then compared with the features found on a 3D model of the object to establish 2D-3D matches [29, 36, 39]. Therefore, the pipeline includes two stages: the first stage extracts local features and compares them with 3D keypoints. The second stage involves 2D-3D correspondences to solve a geometric problem, e.g. via the PnP algorithm, to obtain the 6D position [33]. These techniques combine traditional Computer Vision approaches with Neural Networks. Neural Networks are employed in different stages of the pipeline to improve the overall performance of the system. Figure 3.5 presents a schematic illustration of these methods.

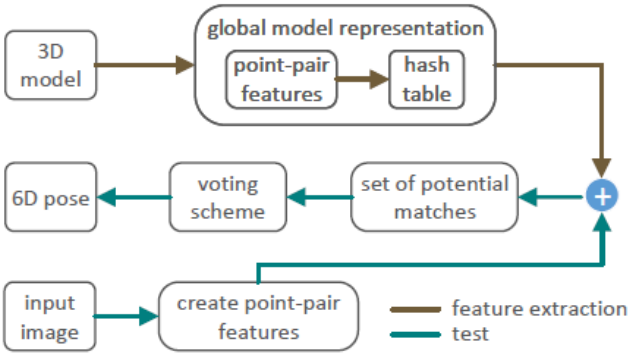


Figure 3.5. Schematic representation of feature matching methods. (*Image from* [34])

Advantages:

1. They are fast and robust to the occlusions between objects and cluttered scenes [29].

Disadvantages:

1. Objects should have rich, well-defined and distinctive textures for computation of local features [29];
2. They do not work well with symmetrical objects [29];
3. The quality of extracted keypoints directly affect the accuracy of position estimation [29];
4. Usually, these methods require a multi-state pipeline which takes much time to perform the task because 2D-3D matches generate a coarse 6D position, so they generally need a supplementary stage to obtain the final pose [29, 37].

In [47, 48] the authors use neural networks to improve the first stage of the pipeline, i.e. the calculation of features, and use a shape fitting algorithm for determining the final position. In particular, the system in [47] proposes a pipeline including object detection, keypoint location, and pose refinement. Object detection is considered a solved problem, so each object is assumed to be already surrounded by its bounding box, obtained through a well-known algorithm, such as Faster-RCNN. The convolutional network architecture used to locate keypoints has a stacked hourglass architecture, shown in Figure 3.6. It employs two consecutive hourglass structures, the first receives the RGB image as input and provides an intermediate version of a heatmap set as output, one for each keypoint, whose

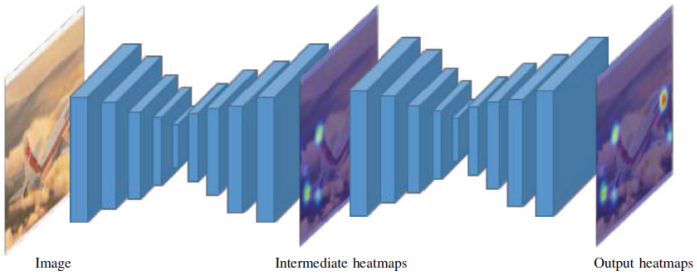


Figure 3.6. Overview of the stacked hourglass architecture. (*Image from* [47])

intensity is directly proportional to the probability of the respective keypoint to be in that position. The second hourglass structure receives the result as input for refinement. If a 3D model of the object is not available, the algorithm generates a deformable shape model for each category, using 3D CAD models with annotated keypoints.

In [48] Peng et al. introduce a neural network called Pixel-wise Voting Network (PVNet), shown in Figure 3.7, to predict the 2D-3D correspondences by regression of pixel-wise vectors to keypoints, representing the directions from each pixel of the object to the keypoints and voting for keypoint locations via RANSAC. The output is a spatial probability distribution for each keypoint, then fed to a PnP algorithm to obtain the final result. This work is robust to occlusion while running at a real-time frame rate.

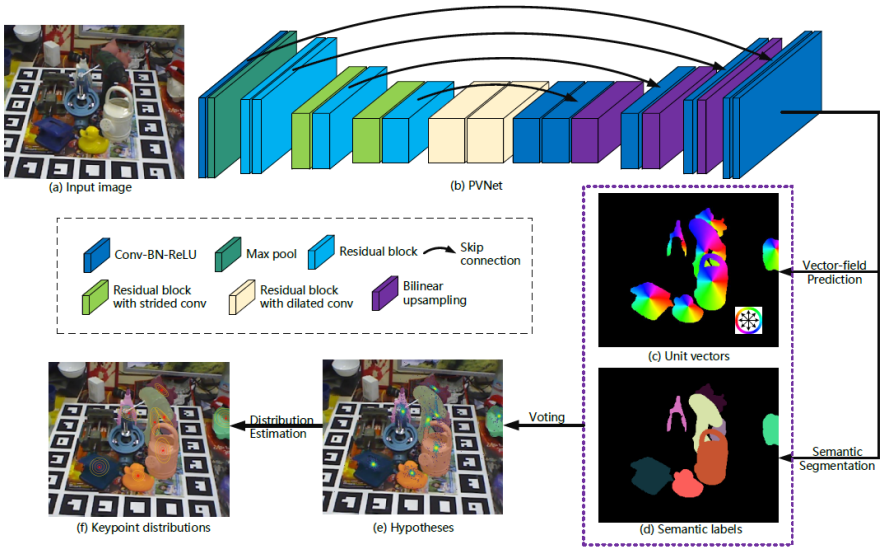


Figure 3.7. Overview of PVNet keypoint localization. (*Image from* [48])

In [49] Chen et al. calculate features, such as class semantic, instance semantic, shape, context, location, assuming that objects are on a plane perpendicular to the image plane and at a certain distance from the camera known from calibration. The system relates to autonomous driving, and Figure 3.8 shows an overview. The 3D candidates receive a score based on these features, and an algorithm sorts them according to this score. Finally, a CNN evaluates again only the most promising ones to provide the final proposals in output.

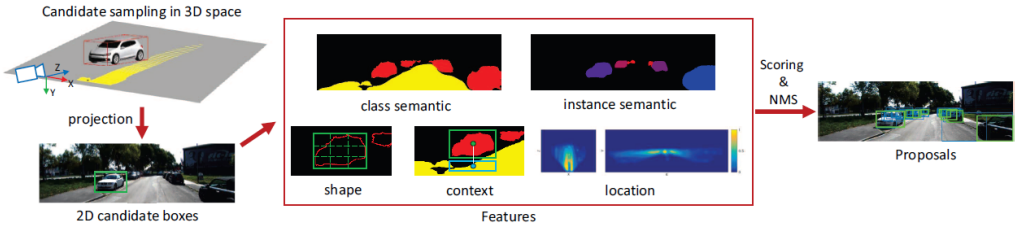


Figure 3.8. 3D object detection for autonomous driving. (*Image from* [49])

In [50] Zhao et al. plan to locate the target object using YOLOv3, select a set of keypoints on the target object as points on the surface, and then train a ResNet101-based keypoint detector (KPD) to locate them. The 6D pose is then retrieved using a PnP algorithm fed with the 3D keypoints correspondences. Figure 3.9 shows the process.

Figure 3.9. Visualization of Zhao et al. pipeline. (*Image from* [50])

Following the pipeline categorisation of the previous methods, also in [33, 51] Zhao et al. use geometrical algorithms to refine the final output. For the first part, instead, they use neural networks able to implement both object detection and keypoint estimation, as Figure 3.10 displays. In [33] they introduce an end-to-end framework with a ResNet architecture trained with viewpoint transformation information and salient regions. The goal is to learn geometrically and semantically consistent viewpoints. In [51] the same authors propose OK-POSE (Object Keypoint-based pose estimation) network, which learns 3D keypoints from relative transformations between pairs of images rather than from explicit 3D labelling information and 3D CAD models.

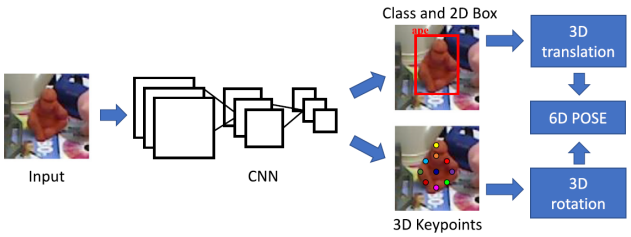


Figure 3.10. Overview of Zhao et al. systems. (*Image from* [33])

In some cases, to remedy the lack of training data, systems employ synthetic images to train the network. In this context, Nath Kundu et al. in [52] introduce a pipeline divided into two parts: a convolutional network learns the local descriptors position invariant to obtain the corresponding keypoints; a second convolutional network by joining the information coming from multiple correspondence maps, provides in the output the final pose estimation (azimuth, elevation, tilt). The algorithm couples input image with synthetic images obtained from rendering a 3D model, with annotated 3D keypoints, used as a template for that specific object class. They train a keypoint descriptor by tracing the positions of the annotated 3D keypoints on the synthetic images. Then, for each pair, the system generates keypoint matching maps to feed the second network. It uses the correlation between the rendered image keypoints and the spatial descriptors obtained from the real one. This model has been tested only on indoor objects such as a chair, sofa, table, bed, so there is no certainty that it can work well in a more general context.

The approach proposed by Chen et al. in [53] focuses on metallic targets, texture-less and with shiny materials, so complex to treat. The process of pose estimation, shown in Figure 3.11, includes three stages: object detection, feature

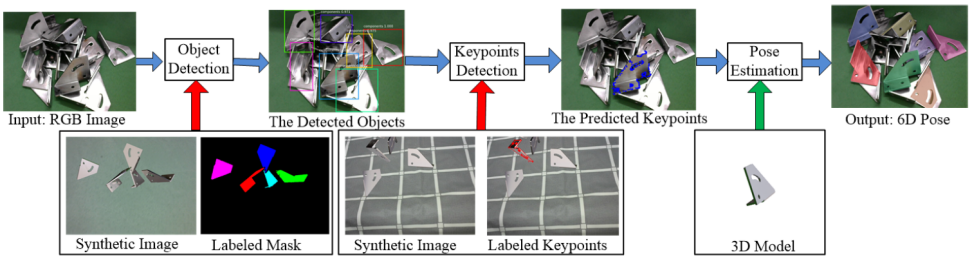


Figure 3.11. Pose estimation for texture-less shiny objects. (*Image from* [53])

detection, and pose estimation. They train a neural network, Mask-RCNN, already known and with good performance for object detection. The network has

synthetic images obtained through Blender, and their respective masks, automatically labelled by a new method introduced, as inputs. The keypoint estimation method uses a "stacked hourglass" convolutional network, trained with synthetic images and corresponding keypoints, automatically labelled. In the third stage, the system uses keypoints for the segmentation and 6D pose estimation. The method employs PnP with RANSAC to remove the outliers, and additionally a CAD model of the object comparing its projected keypoints with those predicted to optimise the output.

3.4 Direct prediction or learning-based methods

These methods predict 6D pose using CNNs [33, 37]. They need a training phase which requires large amounts of labelled data. The employment of CNNs for the 3D position and rotation estimation produces significant improvements. Deep-learning based methods can be of two types: one-stage and two-stage, depending on the use of a further step to refine pose parameters throw Perspective-n-Point (PnP) algorithm (Figure 3.12) [31]. PnP could provide worse results when correspondences are degenerated because of occlusions [29]. In general, two-stage CNNs are more accurate than single-shot ones, particularly on small objects and multiple objects. Computing the cell size and the number of items occupying the same cell is challenging in single-shot object detectors. Moreover, where there are many objects, occlusions between them affect the precision of some single-shot methods, which employ correspondences between an object's 3D bounding box corner and its 2D projection [29].

Figure 3.12. One-stage and two-stage methods. (*Image from* [31])

Advantages:

1. They are powerful and can provide excellent results [35];
2. They have high performance even if the object is partially occluded or in case of cluttered backgrounds [36].

Disadvantages:

1. They require a time-consuming training process [36];
2. They are not very robust to severe occlusions because covering the space of all possible occlusions with real images is unmanageable [54];
3. Their ability to generalize is still a problem in some cases [31].

The introduction of specific strategies tries to solve the lack of training data problem. These strategies involve synthetic images for training or particular processes such as data augmentation, known as Domain Randomization in the context of 6DoF pose estimation. As described by [54], this requires complement data with semi-realistic synthetic images. To do so, they render a 3D model of the object on a real background and then apply different augmentation techniques, such as varying lighting conditions, contrast, blur, and occlusion by removing small image blocks and replacing them with monochrome patches. While Domain Randomization improves the pose estimation accuracy, its benefits on real test images remain limited, mostly because existing Domain Randomization strategies do not tackle the severe occlusions problem, which is one of the main challenges in pose estimation. The following figure (Figure 3.13) shows a schematic overview of these methods.

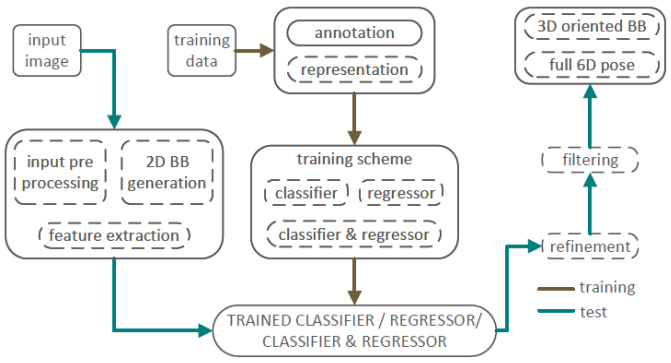


Figure 3.13. Schematic representation of learning-based methods: the dashed–line represents an optional step. (*Image from* [34])

We can classify learning-based methods into three categories [33]:

- Bounding box prediction and PnP algorithm-based methods (Section 3.4.1);
- Classification-based methods (Section 3.4.2);
- Regression-based methods (Section 3.4.3).

3.4.1 Bounding box prediction and PnP algorithm-based methods

These methods use a pipeline for estimating the 6D pose consisting of a CNN architecture for detecting the object category and obtaining the position of the object projected bounding box vertices [33]. The methods belonging to this category are two-stage, i.e. in a first stage, they regress the projection of the corresponding 3D keypoints of the target object in the 2D image and then calculate the actual 6D pose using PnP [31]. All the methods described have the last stage in common, so they differ only on how they prepare data, i.e. the 2D-3D correspondences fed as input to the PnP algorithm for obtaining the final estimate. These methods require expensive manual annotations on bounding boxes [33].

In [55], Rad and Lepetit propose BB8, a cascade of multiple CNNs for object pose estimation task. First of all, the pipeline localizes the object within the image through a CNN that performs the semantic segmentation. A second CNN receives the segmentation result as input and predicts the eight corners of the 3D Bounding Box projections. Finally, after PnP, a third CNN per object is trained to refine the pose. The method handles symmetric items by narrowing the rotation range around the training samples axis of symmetry from 0 to the symmetry angle. Since it employs multiple separated CNNs, BB8 is not end-to-end and is time-consuming for the inference.

The authors of BB8, together with Oberweger, worked on [56], trying to manage position estimation in case of severe occlusions. Since calculating 2D projections using a traditional CNN would be very sensitive to occlusions, the proposed solution calculates heatmaps from small patches independently and then combines them to obtain robust predictions. Similar patches may belong to different positions of the object and thus correspond to different heatmaps, so they also propose a solution to deal with such ambiguities. The training of the network employs both real and synthetic images with annotated positions and their masks. Figure 3.14 shows an overview of the method.

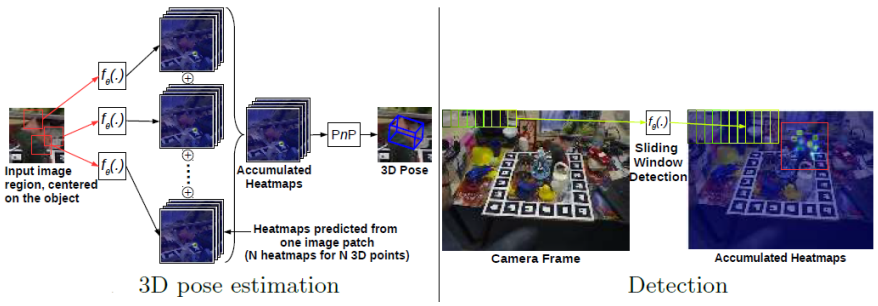


Figure 3.14. Oberweger et al. method to manage severe occlusions. (*Image from* [56])

In [57, 58], Liu and He exploit the advantages of BB8 for regression. They try to avoid the use of PnP for reducing errors and implementation consumption. The former introduces a novel layer, called the Collinear Equation layer, which follows the region layer to provide a 2D projection of 3D bounding box angles and a new representation of the 3D rotation. The latter exploits a new algorithm, called Bounding Box Equation, introduced to achieve accurate and efficient translation.

Most techniques treating the object as a global entity and calculating a unique pose estimate are potentially vulnerable to occlusion. In contrast, in [59], Hu et al. introduce a segmentation based 6D pose estimation framework in which each visible part of objects contributes to the 2D keypoints estimation performed by a local pose predictor. The candidate should be a rigid object, and its 3D model should be available. A CNN regresses 2D projections of predefined 3D points, as the eight vertices of the bounding boxes. The architecture has two streams, shown in Figure 3.15, one for segmentation and the other for regressing 2D keypoints positions, which employs the same encoder and separate decoders. Finally, according to a confidence measure, the system finds the set of 3D-2D correspondences, fed into a RANSAC based PnP strategy to obtain the final pose. Figure 3.15 shows an overview of the method.

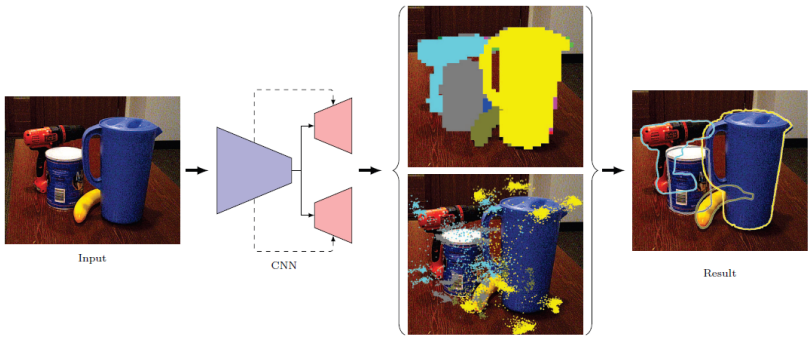


Figure 3.15. Segmentation-driven 6D object pose estimation. (*Image from* [59])

In [60], Li et al. introduce CDPN. This method treats separately the prediction of rotation and translation to obtain a more robust and accurate result, able to handle occluded or texture-less objects. It resolves rotation from coordinates through PnP, using a two-stage object-level coordinate estimation and a Masked Coordinate-Confidence Loss (MCC loss); translation is estimated directly from the image using a Scale-Invariant Translation Estimation (SITE). The approach is very accurate, fast and scalable.

Pix2Pose introduced by Park et al. in [61] predicts the 3D coordinates of each object pixel without using textured training models. The method estimates 3D

coordinates and errors per pixel using an auto-encoder architecture. These pixel-wise predictions are then used in multiple stages to calculate 2D-3D matches and obtain the final pose, using PnP with RANSAC iterations. The method uses appropriate strategies for managing occlusions and symmetries, such as a novel loss function called transformer loss.

In [35], Zakharov et al. introduce DPOD: Dense Pose Object Detector, shown in Figure 3.16. Given an RGB image, an encoder-decoder network regresses the mask and 2D-3D matches. The training phase works with both real and synthetic data. A finishing step, implemented via a CNN, from a coarse proposal predicts the refined one. The real image and rendering are the inputs of two parallel branches. To estimate the final position system employs the difference between the calculated feature tensors.

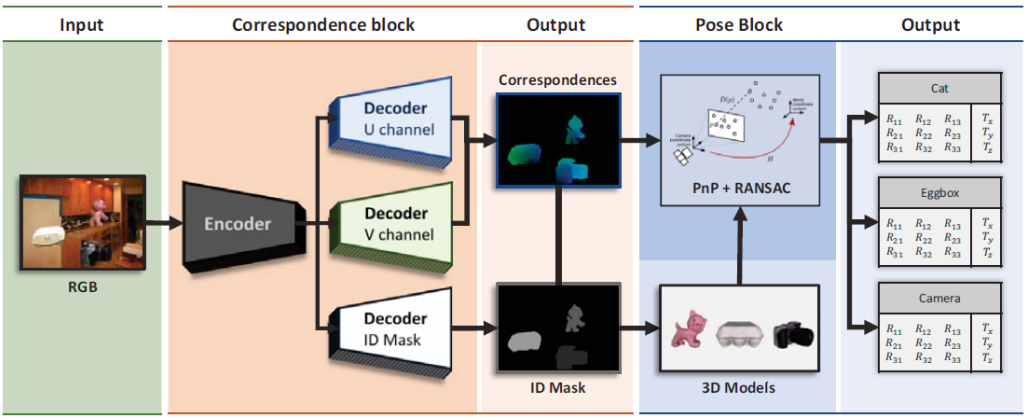


Figure 3.16. DPOD pipeline. (*Image from* [35])

In [54] Li et al. introduce a robust 6-DoF position estimation approach to manage the problem of occlusions and the lack of labelled real images using a Domain Randomization (DR) strategy, which implements changes so that the model works well in the real world even under complex conditions. The method receives an image as input and uses a network to locate the pixels of the object. Next, a Self-supervised Siamese Pose Network (SSPN) outputs the coordinates and segmentation information.

The methods proposed in [62, 63] can work in real-time. The former is an end-to-end framework, uses CNNs to obtain 2D-3D matches and works both with texture-less objects and in case of occlusions between objects. The latter exploits a client-server architecture for robots, shown in Figure 3.17, which uses YOLOv3 for object detection, keypoint detector and pose estimation.

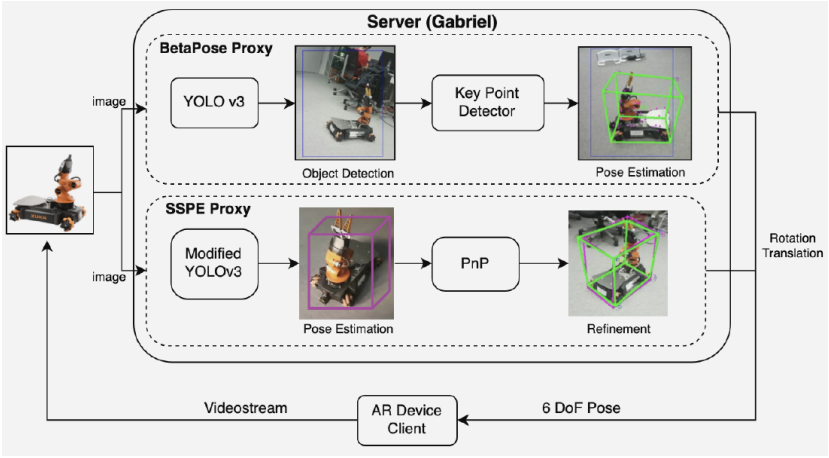


Figure 3.17. Kästner et al. mobile robots application workflow. (*Image from* [63])

Finally, in [64], Liu et al. introduce a new network called TQ-Net. The authors propose a process divided into three stages, displayed in Figure 3.18: in the first

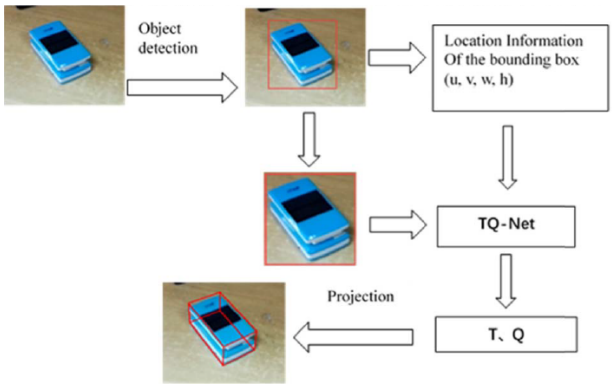


Figure 3.18. TQ-Net method pipeline. (*Image from* [64])

one an object detection algorithm locates the target and obtains its bounding box, resized and, together with its position information, fed into TQ-Net to predict the translation vector T and the quaternion Q. Finally, Q is converted into a rotation matrix R and, through a projection algorithm, the eight points surrounding the object are projected onto the image and connected to form a 3D bounding box to mark the 3D position. TQ-Net is easily implemented, runs in real-time efficiently and accurately and works with all previous CNN-based object detection methods.

3.4.2 Classification-based methods

They convert the 6D position estimation into a classification problem by discretising the pose space. These approaches use CNNs to obtain a probability distribution in the pose space and associate it with the 3D model information to regress the 3D position and rotation [33].

In SSD-6D [65] the authors extend SSD detection framework [66] to 3D detection and 3D rotation estimation based on a discrete viewpoint classification rather than direct regression of rotations. This method uses a neural network to obtain object recognition from the image with its 2D bounding box, and a set of the most frequent 6D poses for that instance is provided for each. It decomposes a 3D rotation space into discrete viewpoints and in-plane rotations, so the rotation estimation is treated as a classification problem. However, to get good results, it is required to find an appropriate sampling for the rotation space. Furthermore, the approach SSD-6D does not directly output the translation. To estimate the translation needs an offline stage for each object to precomputes bounding boxes for all possible sampled rotations. This precomputed information is used together with the estimated bounding box and rotation to estimate the 3D translation. The system selects the best alternative comparing the score for a certain viewpoint obtained by the network and the 6D assumptions. The approach is slow and predicted poses are inaccurate since they are only a rough discrete approximation of the real poses. The output parameters, trained on synthetic renderings, are moved to the 6D space, employing an ICP-based refinement along with the utilization of the intrinsic camera to produce better results.

In [67], Su et al. introduce a neural network trained by rendering synthetic 3D objects superimposed on real images. The trained neural network can then estimate the viewpoints of items in real situations. The following figure (Figure 3.19) shows an overview of the system.

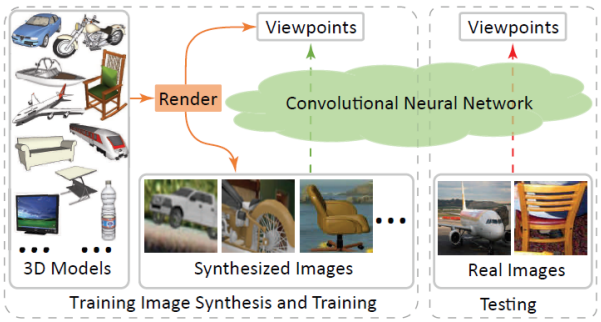


Figure 3.19. Su et al. system overview. (*Image from* [67])

The method proposed in [68] combines the robustness of CNNs with high-resolution instance-based 3D pose estimation. The model is trained with synthetic training data, automatically generated from 3D models of objects and environment images, then combined to create specific training sets for object detection and viewpoint estimation. The model uses a modular architecture, consisting of a detector and viewpoint estimator, which can easily replace with a different implementation. The output of the architecture does not directly provide a 6DoF pose. We need a method to combine the intrinsic parameters of the camera and the 3D model. To do so, we can use PnP, for example.

In [69], the authors introduce GS3D. This system, shown in Figure 3.20, receives an RGB image as input and consists of the following steps: a modified Faster R-

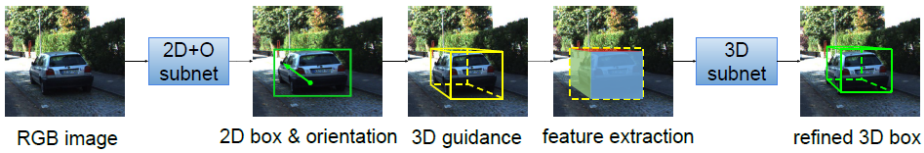


Figure 3.20. Overview of GS3D. (*Image from* [69])

CNN detector, based on a CNN called 2D+O, classifies the rotation from RGB images and the 2D Bounding Box parameters. Then, the 2D bounding box and the orientation obtained are used together with a knowledge of the guidance scenario to generate a basic cuboid called guidance, then projected onto the image plane. The approach involves features extracted from the bounding box and visible surfaces to eliminate ambiguities. Another CNN called 3D Subnet then receives these features to refine the guidance. 3D detection is considered a classification problem and takes advantage of a classification loss to train CNN classifiers and features.

The method proposed in [70] makes an accurate estimate of a vehicle's 3D pose from an image, but it works in a narrow and non-interesting area of research.

Finally, in [71], He et al. use an improved VGG16 network consisting of three stages, which, given an RGB image as input can output a representation of the object and its position even in the case of very cluttered scenes. In the first stage, VGG16 is the backbone network. The second stage extracts feature, and outputs a feature vector, containing a large number of descriptors. This stage converts the original problem into a classification problem. In this way, it is possible to obtain the label and position of the object. Since acquiring large quantities of images for training is very tedious, the authors introduce a data synthesis approach to quickly generate a lot of labelled data with accurate pose parameters.

3.4.3 Regression-based methods

They consider 6D pose as a regression problem and use CNNs to estimate the position [33]. They directly regress the 6D pose parameters of the target object from the input image [31]. Usually, there is a preliminary stage of object detection, and then this information is used to simplify the position estimation process [72]. These methods belong to the category of one-stage methods, i.e. they design a neural network which receives an input image for training and solves the posed problem by learning the rotation and 3D translation of the object represented in it [31].

PoseCNN [73] is one of the current top performers for this task in RGB images. Xiang et al. designed a fully convolutional neural network composed of two stages to jointly segment objects, estimate the rotation, and their distance from the camera, as displayed in Figure 3.21. The first stage extracts feature maps with

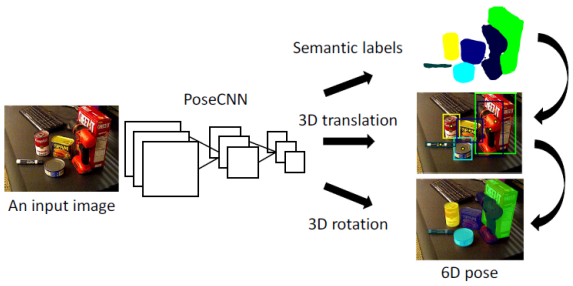


Figure 3.21. Schematic representation of PoseCNN method. (*Image from* [73])

different resolutions from the input image. All the tasks performed by the network will share these feature maps. The second stage integrates the features generated in the previous step with more specific ones. The network outputs semantic labels, 3D translation, and 3D rotation, at the same time. This method estimates the 3D translation using hough voting, which returns the centre of the object and the distance of the target from the camera, while calculates the 3D rotation by returning the quaternion. PoseCNN is trained on real data and proposes a new loss function, called ShapeMatchLoss, useful for pose estimation of rotationally symmetric objects. The authors also proposed a new dataset: YCB-Video dataset. However, by relying on a semantic segmentation approach to localize items, PoseCNN cannot address input images containing multiple instances of the same object and may require further refinement steps to improve the accuracy.

In [74] Tekin et al., based on YOLO [75] and BB8 [55] ideas, introduce YOLO6D, a neural network with fully convolutional architecture capable of efficient and precise object detection and pose estimation without refinement. It receives an RGB image as input, then divided into a regular 2D grid containing fixed-size cells.

Each cell predicts the 2D positions of the 3D bounding box corners projected on the image. The output consists of a 3D tensor which reports a vector for each cell containing 2D corner positions, class probabilities, and a confidence value associated with the prediction. As for BB8, the key feature here is to perform the regression of reprojected bounding box corners in the image. The advantages of this parametrization are compactness and that it does not introduce a pose ambiguity as happens for a direct regression of the rotation. Moreover, in contrast to SSD6D [65], it does not suffer from pose discretization resulting in much more accurate pose estimates without refinement.

Most approaches separate the object detection phase from the pose estimation one, by making them run on two separate networks, as shown in the following figure (Figure 3.22). These methods require resampling the image at least three

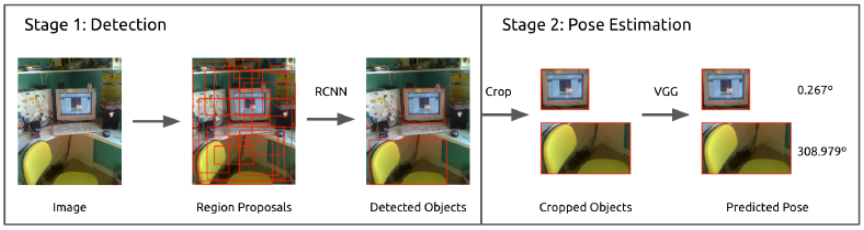


Figure 3.22. Two-stage approaches scheme. (*Image from* [76])

times: 1. to find region proposals; 2. for detection; 3. for pose estimation. The method proposed by Poirson et al. in [76] does not require resampling of the image and uses convolutions to detect the object and its position in a single forward step. It provides acceleration in execution time because it does not require image resampling, and the computation for detection and pose estimation is shared. The scheme employs a Single Shot Detector, as displayed in Figure 3.23.

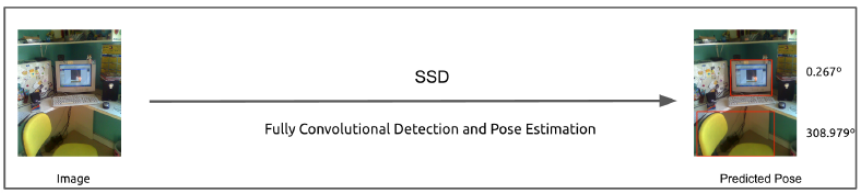


Figure 3.23. Single shot detection and pose estimation. (*Image from* [76])

The method proposed in [77] focuses only on rotation between the object and the camera using a modified version of the VGG-M network. The pipeline contains a featured network and a pose network: the former receives the image and outputs

the image feature descriptor; the latter computes the rotation employing the result of the previous stage and the object category label.

In [78], Mousavian et al. estimate the position and size of an object 3D bounding box from its 2D bounding box and surrounding pixels. This method uses a detector extended to regress the orientation and size of the item by training a CNN. These predictions are combined with geometric constraints to produce the final 3D pose, estimating the translation and 3D bounding box. It, therefore, differs from other methods, which try to regress the pose directly. For orientation regression, the authors propose a new discrete-continuous formulation called MultiBin, which requires the discretization of the orientation angle in n overlapping bin. For each of them, the CNN estimates the probability that the output angle belongs to the current box and calculates the rotation correction, applied to the orientation of the bin's central radius to obtain the output angle. The method does not require any pre-processing stage or 3D models of the object.

The methods described below, although less important than the previous ones, has relevance in research. In [29], Liu et al. estimate the position of an object from an indoor image. It proposes a combination of networks in a sequence: ResNet101, RPN (Region Proposal Network), FCN (Fully Convolutional Network) and allows to obtain as output: classification, bounding box, translation and rotation of the object (Figure 3.24).

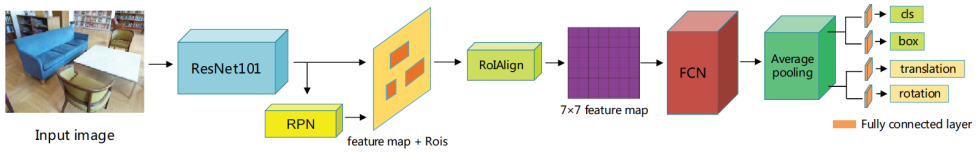


Figure 3.24. 6D object pose from indoor images. (*Image from* [29])

In [37], Do et al. introduce Deep-6DPose to detect, segment and estimate the 6D poses of objects in the input image. Deep-6DPose is an end-to-end deep learning pipeline, consisting of an RPN to derive the Regions Of Interest and a Mask-RCN. It decouples pose parameters into translation and rotation, regressing the latter through a Lie algebra representation. This method predicts the z component and requires post-processing to recover the full translation. However, the CNN architecture consists of six fully connected layers, so needs excessive memory. Moreover, the multi-task loss function has four terms, which have to be balanced carefully.

In [30, 79], the authors use a particular neural network called denoising autoencoder for 6D laying estimation, trying to learn representations from rendered 3D model views. AAE (Augmented Autoencoders) [79] concentrates on pose estimation and training from synthetic models to implicitly learn from rendered 3D

model views while using already computed SSD [66] detection bounding boxes as input.

In [80], Hara et al. consider object seen approximately sideways, is in the centre of an image, with an orientation represented by a value between 0° and 360°. The network processes the image by applying a series of transformations, followed by a unit which estimates the orientation and produces the final predictions. The authors propose three approaches for estimating the rotation: the first two represent angles as points on a unitary circle and trains a regression function, these differ only in the loss function used; the third approach employs the discretization process. The following figure (Figure 3.25) represents a schematic representation of the third approach.

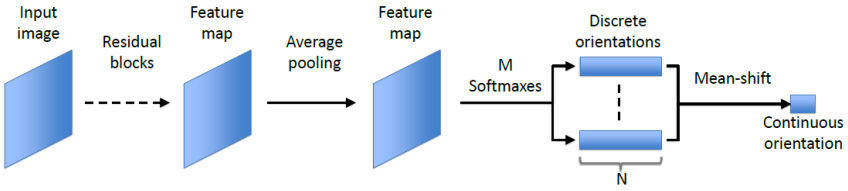


Figure 3.25. The network architecture for the discretization based approach. (*Image from* [80])

In [81], Rambach et al. train a CNN to directly regress the object 6D pose using only single-channel synthetic images with improved edges, obtained from rendering the 3D object. It uses a modified version of the PoseNet architecture [82], introducing a new loss feature to facilitate the training process.

The network used by Xu et al. in [83] consists of two parts: one for the generation of the 2D region proposal using an RPN and the other for the simultaneous prediction of position, orientation, dimensions of 2D objects and 3D poses. Thanks to a stand-alone module for disparity estimation and 3D point cloud calculation, the approach introduces the multi-level fusion scheme. First, the disparity information is encoded with a front view feature representation and merged with the RGB image to improve input. Then, features extracted from the original input and the point cloud are combined to refine object detection. For the 3D location, an extra stream predicts position information directly from the point cloud. The algorithm can directly output 2D and 3D object detection results from the RGB image.

In contrast to other CNN-based approaches for pose estimation, which require many data to be trained, in [84] training is done only with synthetic position data and then it can work well with real data. The process consists of two cascading components, shown in Figure 3.26: a segmentation network (DRN: Dilated

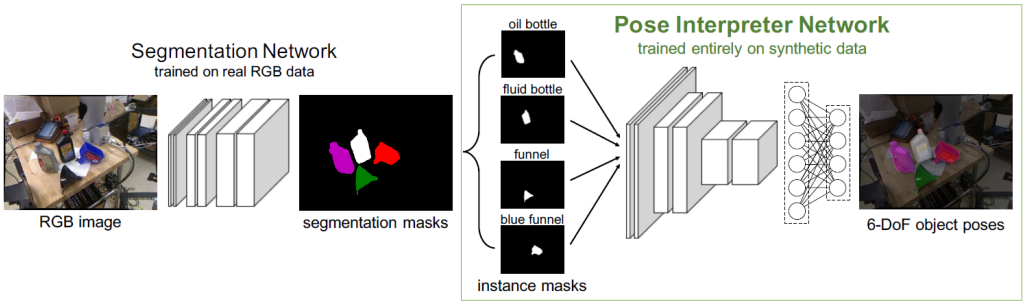


Figure 3.26. Object pose estimation with Pose Interpreter Networks. (*Image from* [84])

Residual Network) which, given an input image, generates the object segmentation masks. The image and the segmentation result are the inputs of a pose interpreter network, a CNN consisting of a Res-Net-18 extractor feature followed by a multilayer perceptron. This network estimates the position of each object. The approach is sensitive to segmentation errors and occlusions, especially for orientation prediction, but is more robust when trained with data containing not severe occlusions. The model works in real-time (20 Hz) on live RGB data; it does not require depth information or ICP refinement.

Finally, in [85], Ku et al. introduce MonoPSR, a method for 3D Object Detection that uses suggestions and shapes reconstruction. First, using the fundamental relationships of a pinhole model camera, 2D detections made by a reliable detector are used to generate 3D proposals for each object in the scene. The 3D position of these proposals is pretty accurate and reduces the final 3D bounding box regression difficulty. Simultaneously, the system predicts a point cloud in a coordinate system centred on the object to learn the local scale, shape information, and reinforce the 2D-3D consistency.

In this last part, we describe the most recent methods and, for this reason, less known. The method proposed by Montserrat et al. in [86] consists of several stages. Given the input image, the system first uses Mask R-CNN to detect and segment objects of interest. The Multi-View Matching Network (MV-Net) estimates the 6D pose: it receives six rendered images containing the detected object in six different poses and the zoomed target image to estimate the initial pose. This network consists of 6 branches, and their results are combined to provide translation and rotation. Finally, the Single-View Matching (SV-Net) network refines the estimated position: it receives in input the target with its mask and the rendered image (obtained from rendering the object in the previous estimate) with its label.

In [87] Hu et al. assume the objects are rigid and their 3D model is available.

The proposed network directly regresses the position from groups of 2D-3D correspondences associated with each keypoint. In detail, after having established the 2D-3D matches, through a segmentation-based CNN, the system uses three main modules to infer the pose: a local feature extractor; a feature aggregation module and a global inference module. The output is the final pose estimated as a quaternion and a translation.

The CNN proposed in [88] compute both the mask and the 6D pose. The system is divided into two distinct networks to overcome the effects caused by the lack of training data: segmentation network and pose estimation network, displayed in Figure 3.27.

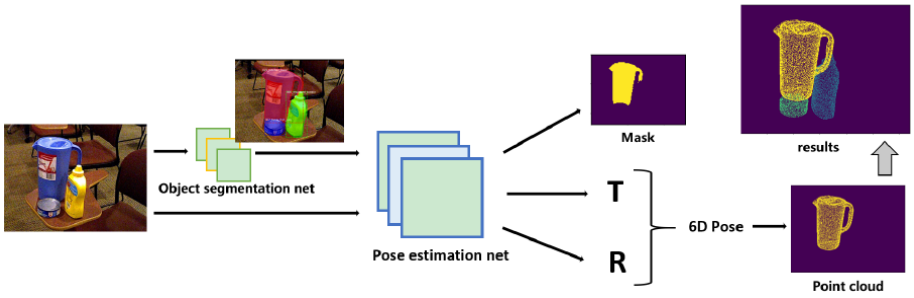


Figure 3.27. Pose estimation model by Wang et al. (*Image from* [88])

The model proposed in [89] is automotive related. A DCNN estimates vehicle position starting from images of the vehicle and patches obtained from 3D models renderings. The system maintains a speed compatible with real-time ($\geq$30 frames per second). The network used is a ResNet-18 without the last layer, used to encode images of cars of varying sizes.

The goal of [90] is to build a system able to understand the point of view of an object never seen before using small available information. Rather than using a predefined coordinate frame or semantic similarity, it starts from a single image with a specific viewpoint used as a reference. Given the input, the goal is to estimate the relative rotation of a second image. During training, the system needs pairs of images with annotated viewpoints. The approach consists of two stages of learning and one stage of inference. It learns first how to predict the shape of an object from two views and their relative positions. Then it trains a Shape Network to estimate the object shape from another point of view. Using the trained Shape Network, a discriminator predicts the degree of misalignment between the two images. During inference, it finds the relative position that best aligns the two inputs. The algorithm removes the input image backgrounds for clarity.

In [38], Liu et al. directly regress the 3D pose of the candidates using a neural

network. Figure 3.28 shows an example of how the system works. Rendered binary images are used in the training phase to generate triplets. The triplets are fed to a triplet network to capture the features, while the positions are reference information. During the testing phase, the candidate object is extracted from the scene represented in the image and then passed through the two previously trained networks. The regression network provides the final position. The use of binary images allows the management of objects without textures, making the network more sensitive to contours and robust to lighting variations. The method also manages symmetrical objects and occlusions implementing ad-hoc strategies. However, edges can be inaccurate and cause errors; runtime is high due to manually obtained object patches.

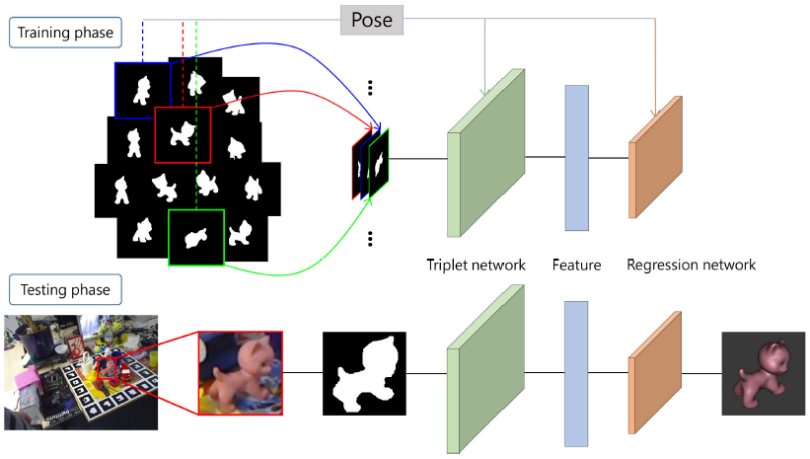


Figure 3.28. Pose estimation for texture-less objects. (*Image from* [38])

In [32] Capellen et al. introduce ConvPoseCNN, an architecture derived from PoseCNN [73], described above. Initially, a VGG16 convolutional backbone extracts the features. The system performs first pixel-wise semantic segmentation through a fully convolutional branch. Then a fully convolutional vertex branch estimates central direction and depth. The results of these two branches find the centre of the objects via a Hough transform layer, which also predicts the bounding boxes of detected objects. A fully convolutional architecture, similar to the other two branches, replaces PoseCNN quaternion estimation branch to estimates quaternions for each pixel, which are regressed directly using a linear output layer. The following figure (Figure 3.29) shows an overview of the system.

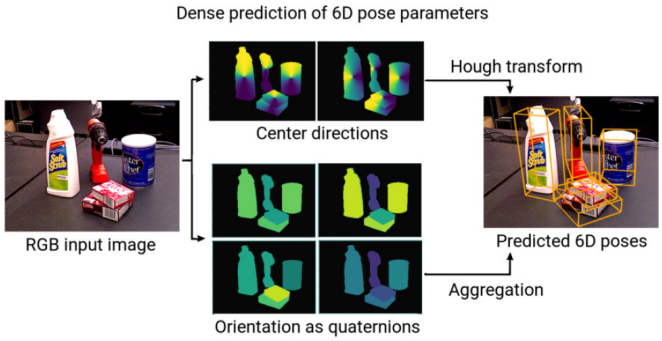


Figure 3.29. Overview of ConvPoseCNN system. (*Image from* [32])

3.5 Conclusions

We proposed methods belonging to different fields, such as robotics, autonomous driving, and entertainment. All of the methods described estimate the 6D position of an object from an image, reaching, in some cases, high levels of accuracy. But the object considered had distinctive shapes or textures in most cases. These characteristics make the process less complex to address than the one this thesis focuses on.

We worked in the medical field, and the object of interest is a little organ, the prostate, having a simple roundish shape and a texture which is easily confused with the surrounding tissues. Moreover, during surgery, the illumination conditions are not optimal, and there could be occlusions caused by surgical tools, which complicate the problem even more. For this reason, the task results complex for a machine and for a human without medical knowledge too.

The proposed method follows a specific process involving two different neural networks to handle the reported problems. The former performs the semantic segmentation to detect the region of interest, and the latter addresses the rotation issue as a classification problem, following a method similar to [67, 68]. We will explain the details of the process in the following chapter.

Chapter 4

Proposed Methodology

4.1 Deep Learning and Augmented Reality Solutions for Urologic Surgery Support

Urology was one of the first adopters of Artificial Intelligence for object detection, image classification, segmentation, skills assessment, and outcome prediction for complex urologic procedures [1]. Recently, minimally invasive laparoscopic and robotic-assisted approaches have replaced many traditional open urologic surgeries [1]. These technologies assist the surgeon in performing more complex and precise tasks improving the precision of the procedure, with consequent benefits for the patient during the post-operative period [2]. This change of paradigm led the surgeon to see the operatory scene through a console and a visor experience, reducing his spatial perception of the surgical environment [2]. This drawback can be mitigate exploiting Augmented Reality. Augmented Reality allows a real-time overlapping between computer-generated images or 3D models and the real environment. In the medical field, doctors can apply AR technology in the pre-operative phase for the operative setting, for data visualization on diagnostic to make diagnosis and prognosis, and for treatment procedures during in-vivo surgeries [1]. We will focus on Augmented Reality applied during an in-vivo urologic surgery. In this delicate phase, the main challenge for an Augmented Reality application is the registration process. Registration is the accurate alignment of the virtual model with its physical counterpart [3]. There are different strategies to achieve precise alignment. The first way is to use endoscopic markers placed on the surface of specific structures as points of reference during the surgery. Another option to determine the organ position in real-time is a marker-less approach, which is technically more challenging and time-consuming, and exploits machine learning techniques [3, 4]. The main advantages of Augmented Reality for doctors during in-vivo surgeries are [3]:

- The opportunity to view reconstructions directly on the body of the patient, reducing the number of distraction caused when the surgeon has to look away from the surgical site;
- The reduction of surgery time, exploiting AR during pre-operative planning for tailoring incisions and cutting plans;
- It improves the surgeon's spatial perception of the surgical field, avoiding unnecessary manipulations or accidental injuries to inner organs.

In this work, we involved Augmented Reality to improve the surgeon's spatial perception during an urologic surgery called Robot-Assisted Radical Prostatectomy (RARP), described in the following section.

4.2 The Robot-Assisted Radical Prostatectomy (RARP) Procedure

Radical prostatectomy removes the prostate gland and tissues surrounding it in case of prostate cancer. The precision of this procedure is fundamental for patient well-being to avoid short and long-term complications. The adoption of robotic surgery tries to reduce these risks minimizing the invasiveness of the intervention [5]. To improve the spatial perception of the surgeon, the RARP can exploit the Augmented Reality technology. AR allows the registration of the 3D virtual model of the patient's organ over its image captured by the endoscope, using different real-time tracking techniques. We can subdivide the procedure into five subsequent standardized stages [5]. According to [2], a stage is a set of surgical tasks performed by the surgeon during an interval with similar visual conditions and with the same level of benefit from a specific type of visual augmentation. The procedure stages are:

1. *Defatting and incision of the endopelvic fascia.* During this stage, identifying and locating the prostate neck could be a difficult task for the surgeon. Therefore, a correct overlay of the 3D model can enhance intra-operatory perception and potentially lead to fast and more precise surgery [2];
2. *Management of the bladder neck.* In this phase, the surgeon rarely requests the 3D overlay [2];
3. *Vase clamping and nerve-sparing.* Preserving the functionality of the nerves after the procedure is necessary for the patient's well-being. A 3D model correctly representing organ boundaries can improve the precision of this stage. Moreover, deforming the 3D mesh accordingly to tissue stretching and blending further increase the accuracy of organ border detection [2];

4. *Surgery by the prostatic apex.* In this step, 3D augmentation is not useful because reconstruction from Magnetic Resonance Imaging (MRI) does not accurately portray the organ apex [2];

5. *Targeted Biopsy.* After the organ's removal, the 3D overlay is fundamental to locate tissue samples for post-operatory biopsies. In this stage, the possibility to insert a catheter in the pelvis cave offers an artificial visual element easy to be detected and used to guide the registration process [2].

4.3 Existing Augmented Reality Solutions for the RARP Procedure

The RARP procedure can benefit from Augmented Reality mainly in three stages: the defatting and incision of the endopelvic fascia, during vase clamping and nerve-sparing, and for targeted biopsies. A modular approach exists, which is currently used during in vivo surgery, for extensive testing, by the Urology unity of the San Luigi Hospital, in Orbassano (TO) [2]. This system adopts a specific augmentation strategy for each stage of the surgical procedure exploiting different visual elements, such as anatomical features or surgery tools, to guide the virtual-over-real overlap. The system receives two input data, the endoscopic camera video stream images and the 3D patient-specific organ's mesh, and when it completes the augmentation process, it generates the output. During the augmentation process, the system mixes colours from the two sources to avoid unwanted chromatic effects. The Tile-Pro visualization system of the Da Vinci surgical console directly displays the final augmented video stream. In this way, the surgeon can visualize the output on the remote monitor where he or she is operating. The framework needs an accurate virtual reconstruction obtained from high-resolution pre-operatory medical imaging techniques, such as MRI. This virtual model reproduces the prostate of the patient undergoing surgery and its surrounding structures, and it is modelled by bio-engineers using the $HA3D^{TM}$ trademarked technique. Moreover, the surgeon can modify in real-time the transparency value of each element to visualize only a specific subset of structures and maximize his or her perception of the intra-operatory environment [2].

According to the surgeons' needs, three distinct strategies exist for registration, applied in different procedure stages [2]:

- *Feature-based*, useful for rapid localization of the prostatic apex;
- *Human-assisted*, to improve precision during nerve-sparing, preserving nerves' functionality;
- *Marker-based*, helpful to locate tissue sampling for post-operatory biopsies.

4.3.1 Starting point

The starting point of this work is the registration strategy adopted for the Targeted Biopsy stage. In the last stage of the RARP procedure, the surgeon can introduce a catheter in the operative environment after the prostate removal. The framework employs this element as a reference point for a marker-based registration strategy, which is computationally less costly. In this phase, augmentation is fundamental as it helps the localization of the tumoral lesions allowing the surgeon to collect tissue samples for biopsies [2]. This application involves three stages, shown in Figure 4.1:

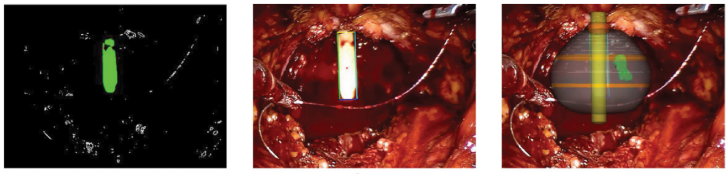


Figure 4.1. Application steps. (*Image from* [2])

a. Since the catheter's colour is a distinguishable feature, the system applies binary thresholding to the frame image for excluding those pixels not belonging to the colour range defined for the catheter. The application highlights in green the portion of the image recognized as the catheter [2].

b. After the feature detection, the system computes the bounding box and the contour of the catheter. The software considers as vertices of the bounding polygon four points of the contour that are closer to the four corners of the bounding box, filtering wrong results. The following figure (Figure 4.2) shows

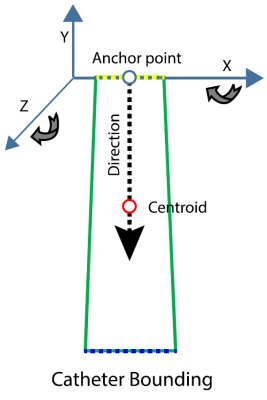


Figure 4.2. The bounding polygon of the catheter. (*Image from* [2])

the bounding polygon of the catheter where the centre point of the upper edge (in yellow) of the bounding polygon is the anchorage point of the 3D mesh, corresponding to the prostate apex. The bounding box helps the system to compute scale and rotation values. It sizes the upper edge of the bounding box according to the real diameter dimension of the catheter. The estimation of the rotation values involves different approaches according to the axis:

- *Z-axis* rotation: uses the vector from the centre point of the upper edge to the bounding box centre of mass [2];
- *Y-axis* rotation: not considered as rarely involved [2];
- *X-axis* rotation: handled comparing dimensions of the upper and the lower edges of the bounding polygon. When the lower border is greater than the upper, the software rotates the mesh toward the camera and vice-versa [2].

c. The final result shows the virtual prostate in the position of the removed one. The figure displays, in dark green, the cancer mass to guide the surgeon in sampling tissues for the biopsy [2].

This thesis tries to extend the approach implemented for the targeted biopsy stage to the first and the third phases: the defatting and incision of the endopelvic fascia and vase clamping and nerve-sparing. This task is more complex to address for these stages because we cannot consider a visible element as the catheter as a reference point, but we have to work directly on the prostate. The proposed approach, described in the following sections, tries to solve the registration problem involving neural networks.

4.4 Implementation of the proposed approach

As we said before, the focus for this work is finding a new technique for real-time automatic registration of a 3D virtual prostate model and its physical counterpart during a Robot-Assisted Radical Prostatectomy. The roundish shape of this little organ and its texture, similar to the surrounding tissues, make the task challenging. Moreover, non-optimal illumination condition and occlusions with surgical tools further complicate the situation. To address these problems, we propose a learning-based approach, described in detail in the following sections. This approach, applied to the catheter, achieves satisfactory results. Therefore, this work tries to extend the method to the prostate. The system involves two convolutional neural networks:

- *Segmentation Neural Network*: it acquires an image as input and outputs a segmentation mask, which distinguishes prostate, surgical tools and background. This network provides the position and the scale of the organ;

- *Rotation Neural Network*: it receives a synthetic image and its rotation tag and outputs X-axis and Y-axis rotation values.

We manually created both the datasets following two different strategies described in Section 4.4.1. By combining the results of both the network, we know all the essential parameters for the overlay. Figure 4.3 shows an overview of the proposed methodology:

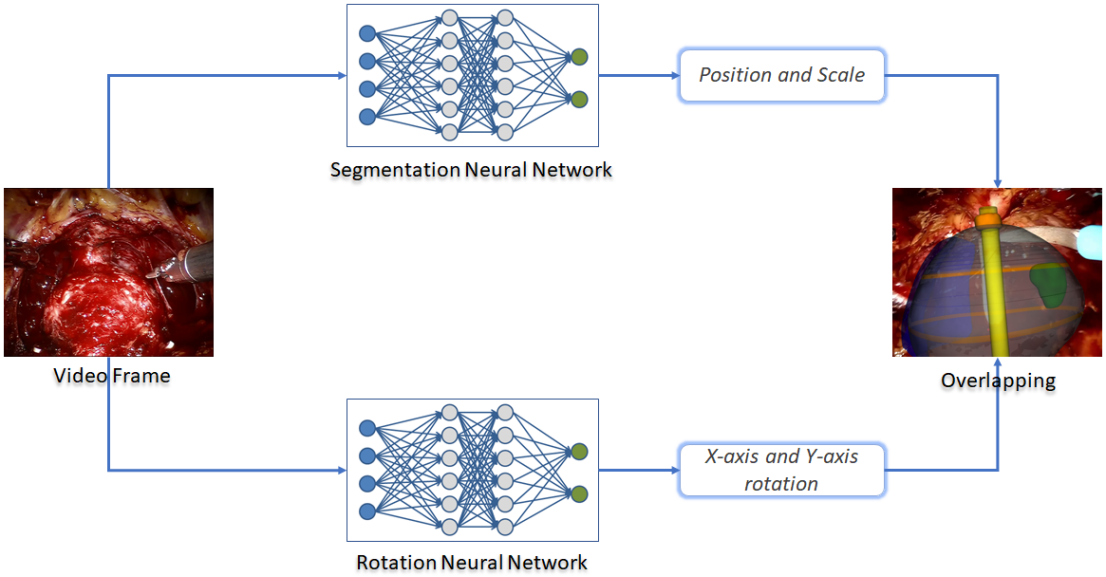


Figure 4.3. General overview of the proposed approach.

4.4.1 Datasets Creation

The datasets creation was the first step for the implementation of our method. The dataset creation strategy was different according to the specific network to address the semantic segmentation and the rotation estimation tasks. We explain both the approach in detail below.

Semantic Segmentation Dataset

To train the segmentation network, we needed the RGB images and the corresponding segmentation masks. We obtained 388 RGB images by extracting frames from three different surgical videos provided by the San Luigi Hospital of Orbassano (TO). As described in Section 2.4.1, the segmentation mask is a matrix that has the same dimensions of the image and contains the predicted class identifiers corresponding to all the pixels. For our model, we needed to distinguish

background, surgical tools and the prostate. Therefore, our segmentation masks should contain three different classes, each marked with a unique ID: "*background*", "*tool*", and "*prostate*". We manually generated the ground truth segmentation masks through a graphical image annotation tool called *labelme* [91]. This tool, given an image, allows creating polygons, as shown in Figure 4.4, and assigning each of them the corresponding segmentation class.

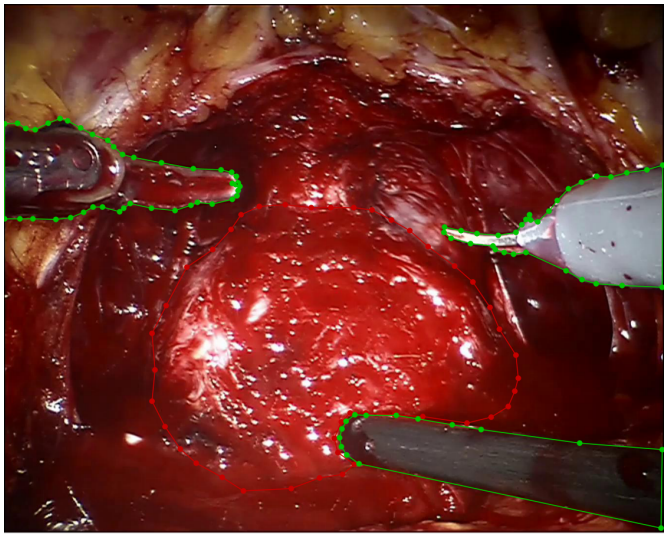


Figure 4.4. Manual Tagging with Labelme.

Labelme provides a *JSON* file for each image as output, which contains the vertices coordinates for each created polygon and the corresponding *group_id*. As an example, the following code fragment displays the *JSON* file provided in output by labelme:

```
{
  ...
  "shapes": [
    {
      "label": "prostate",
      "points": [
        [
          761.4457831325301,
          537.1325301204819
        ],
        [
          706.0240963855421,
          631.1084337349397
```

```
        ],
        [
          798.7951807228915,
          678.0963855421686
        ]
      ],
      "group_id": null,
      "shape_type": "polygon",
      "flags": {}
    },
    {
      "label": "tool",
      "points": [
        [
          574.6987951807229,
          353.99999999999994
        ],
        [
          633.7349397590361,
          308.2168674698795
        ],
        [
          674.6987951807229,
          375.6867469879518
        ]
      ],
      "group_id": null,
      "shape_type": "polygon",
      "flags": {}
    }
  ],
  ...
}
```


In the segmentation images, each pixel value should denote the class ID [15]. Therefore, before feeding this output to the segmentation network for training, we needed a preprocessing step to convert the *JSON* file into a segmentation mask. In this way, we obtained the final segmentation mask, in which the pixel values were 0 for those pixels representing the background, 1 for surgical tools, and 2 for prostate, as Figure 4.5 displays.

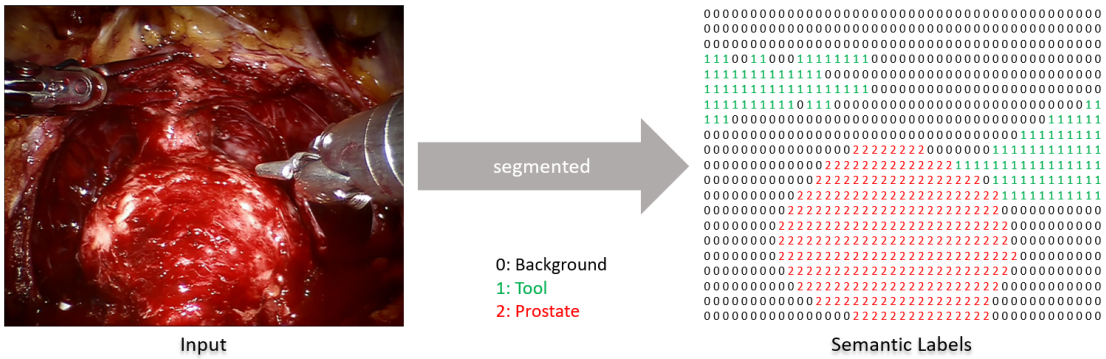


Figure 4.5. Input image (left) with its semantic labels matrix (right).

Rotation Dataset

The rotation estimation task was complex to address because it is troublesome to determine the prostate rotation values from an image, as they are minimal. For this reason, we created a synthetic dataset through Blender to make the rotation estimation easier by using 3D models of the prostate, three surgical tools and 2.688 real backgrounds as inputs. The core for this dataset creation is a Python script, which manipulates the data, and generates 20.0000 images for our dataset.
According to real rotation values of the prostate during surgeries, we considered the following ranges in degrees for the three rotation axes:

- *X-Axis*: $[-35, -15]$;
- *Y-Axis*: $[-25, 25]$;
- *Z-Axis*: $[-10, 10]$.

Figure 4.6 displays the 3D virtual prostate with the Cartesian Axes.

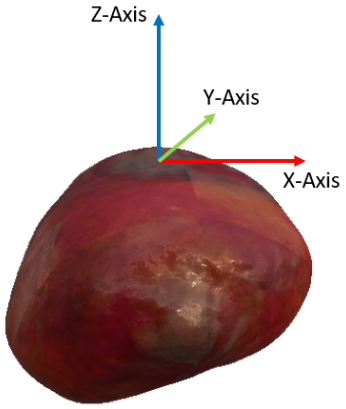


Figure 4.6. 3D prostate with Cartesian Axes.

The script generates a rendering for each combination of the previous prostate virtual model rotation values, randomly rotating its texture once every 100 renderings and randomly changing lighting conditions, surgical tools positions, the background of the scene, the view scaling, and the camera position. Completed all the iterations, we obtained a synthetic dataset of 20.000 images to train our Rotation Neural Network and a file containing the rotation values for every image. Figure 4.7 shows the Rotation Dataset creation pipeline.

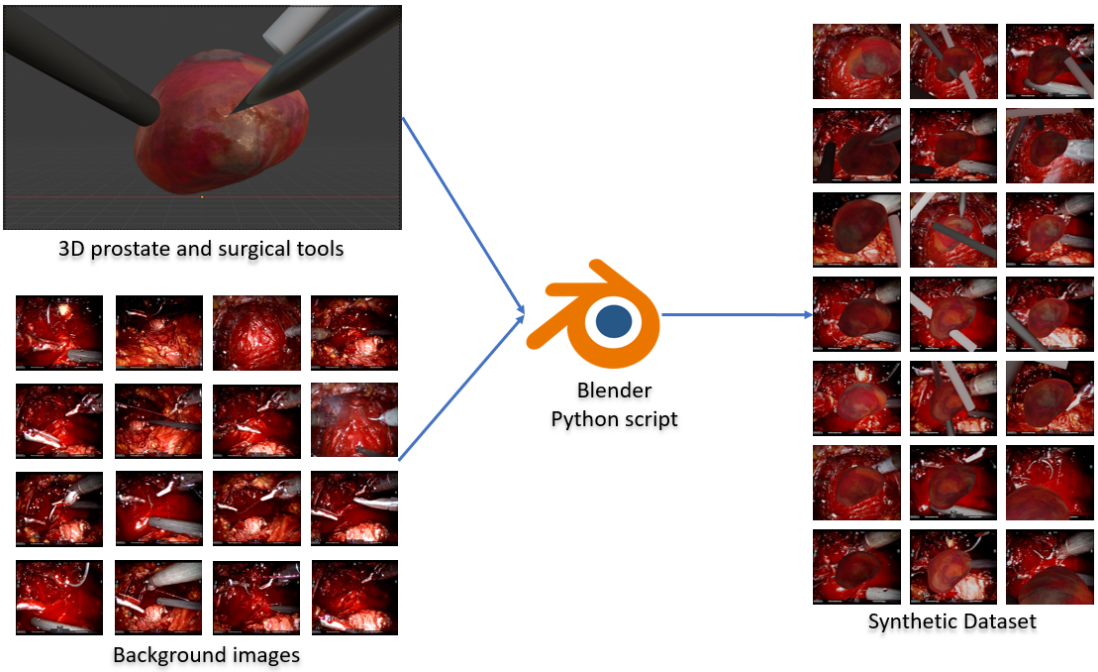


Figure 4.7. Rotation Dataset creation.

4.4.2 Segmentation Neural Network

After the dataset creation, we randomly split the 388 images and their corresponding labels into three classes: 310 for training, 39 for testing, and 39 for validation. To do so, we employed *Scikit-learn*, an open-source machine learning library for the Python programming language [92]. We used a random split because consecutive frames had a high correlation as they belonged to the same video. Therefore, the model would have been not very efficient and generalized. Once splitting the dataset, we prepared the data for the training phase of our Segmentation Neural Network. We needed to create two folders for every group of images [93]: an images

folder for all the training images, and an annotations folder for the corresponding ground truth segmentation images. The labels filenames had to be the same as the RGB images ones, and both the elements had to have the same size [93]. To implement our model, we involved several open-source APIs: *OpenCV* [94], a library for computer vision; *Tensorflow* [95] and *Keras* [96], libraries for machine learning and neural networks. In particular, for the semantic segmentation task, we exploited the *Keras-segmentation* module [93]. The task the neural network had to learn was the semantic segmentation of the photogram into three classes: prostate, surgical tools, and background. During the training phase, the neural network was fed with every RGB images and the corresponding label and, at the end of the training process, outputs the trained model, able to distinguish the three classes. Figure 4.8 shows an example of the training process:

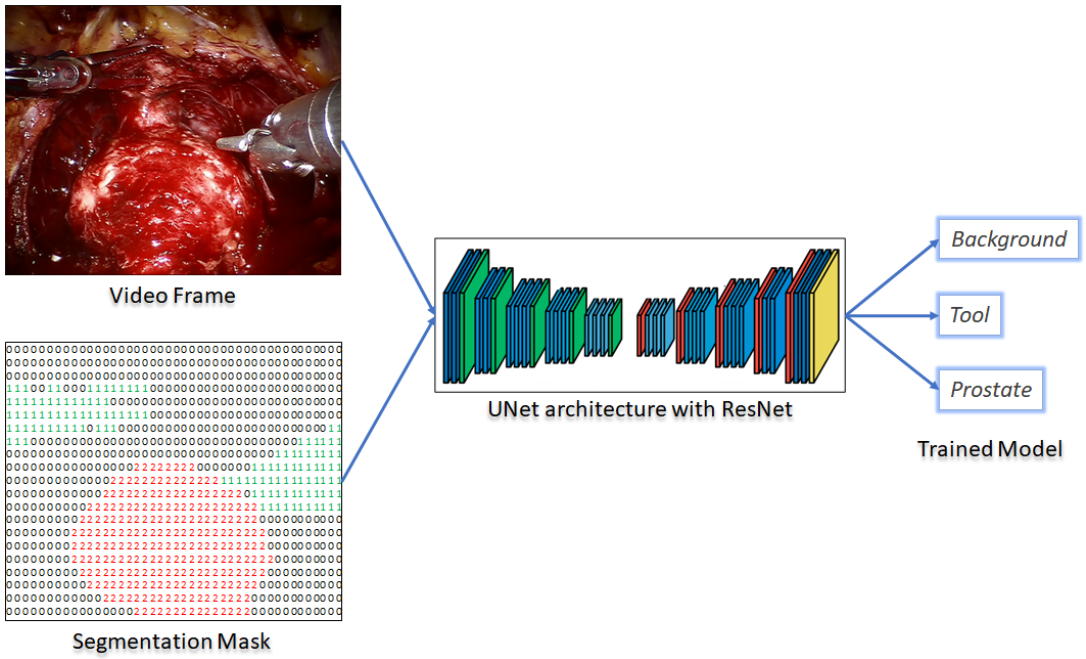


Figure 4.8. Segmentation Neural Network Training.

We trained the model with all the three segmentation models described in Section 2.4.1, SegNet, UNet, and PSPNet, with ResNet, VGG-16, and MobileNet architectures. By combining architectures and models, we obtained nine different models. The Keras-segmentation module allowed us to import the models already compiled and ready for training. We ran every model for 50 epochs using a batch size of 8, except for UNet-ResNet and Segnet-ResNet models, which used a batch size of 4, and an Adam optimizer with a learning rate of 0.0001. We saved the model for each epoch to allow choosing a previous version in case of overfitting.

Then, we tested all the trained model to choose the best one. During testing, the trained neural network received the RGB test images and produced the segmented image. For better visualization, we then overlapped the result over the input image with a certain level of transparency, as Figure 4.9 displays. Among the different combinations between architecture and methods, as usual for medical images, the UNet architecture with ResNet gave the best test accuracy.

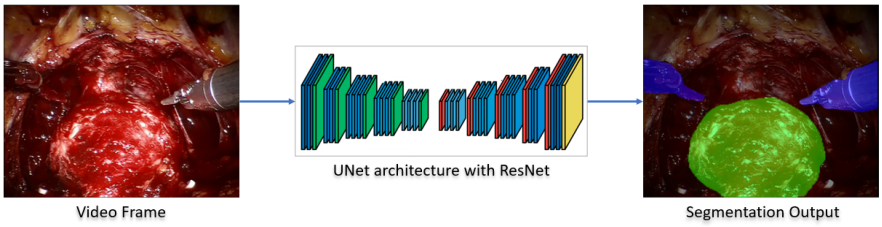


Figure 4.9. Segmentation Neural Network Testing.

4.4.3 Rotation Neural Network

For the Rotation Neural Network, we used a *ResNet50* model, shown in Figure 4.10:

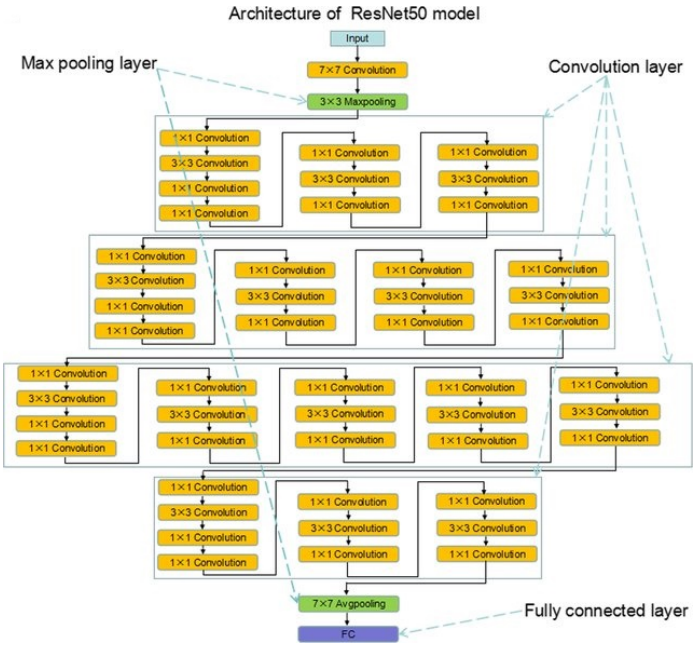


Figure 4.10. The architecture of ResNet50. (*Image from* [97])

Starting from this base model, we deleted the fully connected top-layer creating two different branches for *X-Axis* and *Y-Axis* rotation, respectively (Figure 4.6). We did not consider *Z-Axis* rotation since it can assume only minimal and not relevant rotation values. We chose this configuration as it was the best performing after several attempts. Every branch has the same structure, which contains:

- A *Dense Layer*, which is a regular densely-connected Neural Network layer, with 4096 neurons and a *ReLU* activation function [98]. We introduced this layer because it experimentally improved the performance;
- A *Batch Normalization Layer*, which normalizes its inputs by applying a transformation that maintains the mean output close to 0 and the output standard deviation close to 1 [16];
- A *Dropout Layer*, which applies *dropout* to the input, by randomly setting input unit to 0 with a frequency of 0.5 at each step during training time to prevent overfitting [99];
- Another *Dense Layer*, with several neurons equal in number to the current axis classes [98];
- An *Activation Layer*, which applies the *Softmax* activation function to the output [100].

We solved the estimation of the *X-Axis* and *Y-Axis* rotation value as a classification problem. We subdivided the set of possible rotation values along an axis according to the range used for the dataset creation. We considered 20 classes for *X-Axis* Rotation (from −15 to −35) and 50 for *Y-Axis* Rotation (from −25 to 25). Therefore, the neuron with the highest probability according to the *Softmax* activation function will fire and produce the corresponding rotation value as output. Figure 4.11 displays an overall scheme of the Rotation Neural Network:

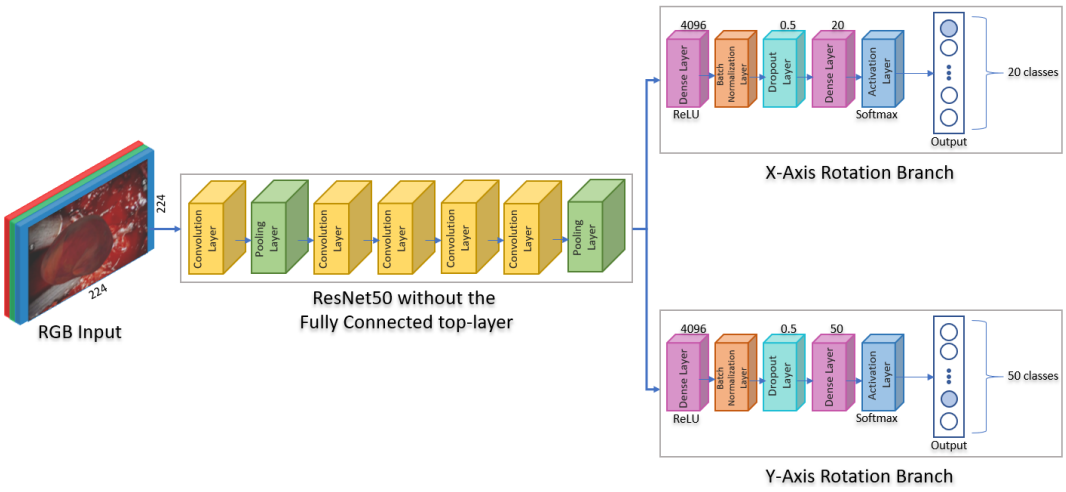


Figure 4.11. Pipeline of the Rotation Neural Network.

We randomly split the 20.000 synthetic images and their corresponding rotation labels into three groups: a training set of 18.050 images, a test set of 1.000 images, and a validation set of 950 images. We resized the original images to 224×224 pixels, as it is the standard size for *ResNet50* input. We ran our model for 13 epochs, using a batch size of 32 and an *Adam* optimizer with a learning rate of 0.00001. We also implemented three callbacks:

- *EarlyStopping*, which stops training when the monitored metric has stopped improving since five epochs (patience value) [101];
- *ModelCheckpoint*, which allows saving the Keras model or model weights at some frequency according to a monitored quantity. We decided to maintain the model at the end of every epoch regardless of performance [102];
- *TensorBoard*, a visualization tool provided with *TensorFlow* that allows plotting graphs related to accuracy and loss [103, 104].

To calculate the loss, we used *Categorical Crossentropy* because we had more than two label classes, provided with a one-hot representation [105]. During the training phase, we fed the model with the training images and the corresponding rotation values, as shown in Figure 4.12. Moreover, we validated the network after each epoch on the validation set. We tested the model both on the synthetic test set and real images, considering a prediction wrong when the deviation between the predicted and the actual values was greater than 5 degrees.

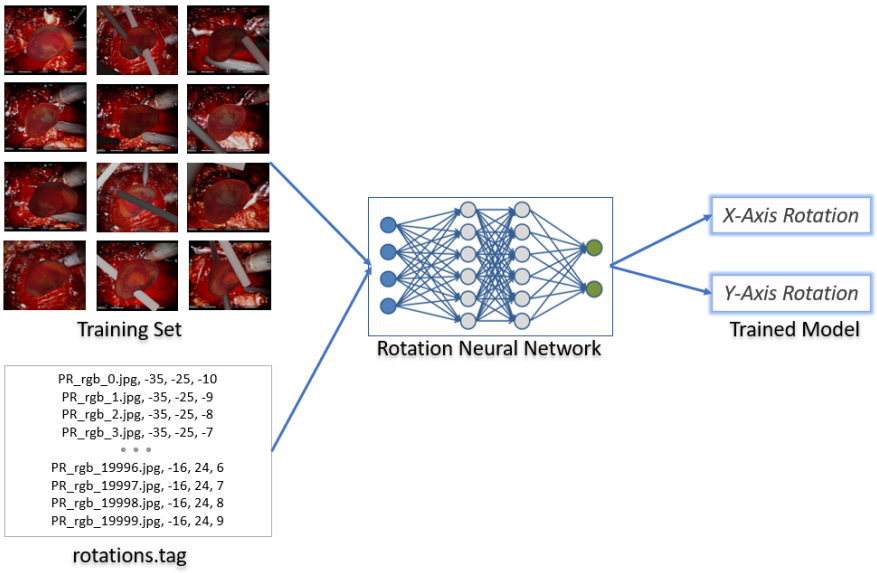


Figure 4.12. Rotation Neural Network Training.

Chapter 5

Testing Results

We ran both the Semantic Segmentation and the Rotation neural networks using an NVIDIA Quadro P4000 as GPU. We involved different strategies and metrics to test the two neural networks and produce the testing results.

5.1 Segmentation Neural Network

As already said in the previous Chapter, we ran 9 different models obtained by combining three Semantic Segmentation architectures (UNet, SegNet and PSPNet) and three neural network models (MobileNet, Vgg, and ResNet). The training phase of these models was fast enough. Although we trained our models for 50 epochs, the GPU required an average of 23 minutes for each configuration. For a first evaluation, during the training phase, we monitored if the model was starting to overfit by testing the validation accuracy on the validation set after each epoch. Completed the training phase, we plotted the accuracy progression for both the training and the validation sets for a first comparison between the different models. As Figure 5.1 displays, almost all the models reached a satisfactory level of accuracy.
The training accuracy progression presents an increasing trend in the first 20 epochs and a more flat one in the following part. This behaviour means that, in general, our semantic segmentation models learn rapidly for about 20 iterations, then they continue to learn, but the learning rate becomes much slower. The validation accuracy progression has a slightly more irregular trend, probably due to the small validation set of 39 images. Despite this, the validation accuracy presents minimal oscillations between really high values, 0.8 and 0.9, for almost all the networks except for PSPNetMobilenet and SegnetMobilenet architectures, which present peaks at low levels of accuracy in the first phase. This behaviour is probably due to the Mobilenet neural network architecture, which, having a small

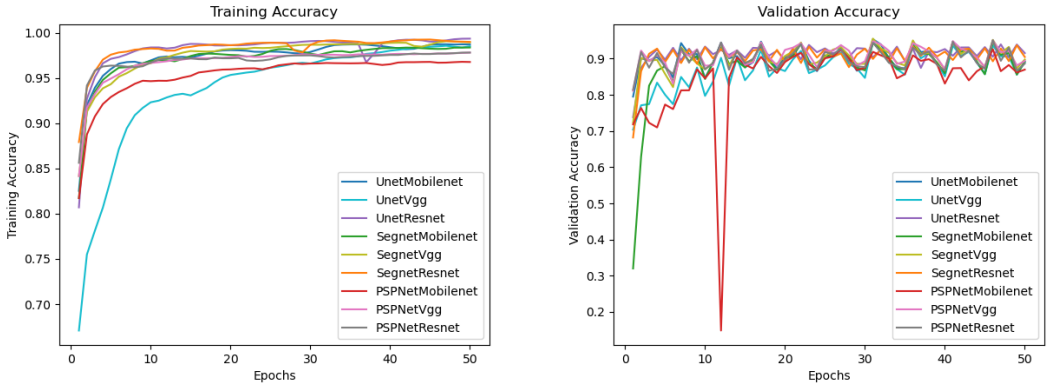


Figure 5.1. Training and Validation Accuracy.

model size and faster inference time, could have low accuracy, as already said in Section 2.4.1. Graphically, all the trained models present accuracy values close to each other at the end of the training phase.
To choose the best segmentation model, we tested the trained architectures on the test set, made of 39 images, by analyzing the models' performances analytically. Therefore, to evaluate the testing accuracy, we exploited a diffuse metric for semantic segmentation, known as *Intersection-over-Union* (IoU) or *Jaccard Index*. This method quantifies the percent overlap between the target mask and the prediction input. To evaluate our models through IoU metric, we first calculated true positives, false positives, true negatives, and false negatives, and then, we computed the IoU, which is defined as [106]:

$$IOU = \frac{true\_positive}{(true\_positive + false\_positive + false\_negative)} \tag{5.1}$$

In other words, the IoU measures the number of pixels common between the target and prediction masks divided by the total number of pixels present across both the segmentation masks [107, 108]. We determined the IoU score for each category separately (background, tool, and prostate), then averaged over the three classes to provide a global mean IoU score for every semantic segmentation model [107]. According to the definition, the IoU score is a value between 0 and 1. If the prediction is perfectly correct, $IoU = 1$. The closer the IoU is to 0, the worse will be the prediction result [108].
Table 5.1 contains the IoU scores for background, tool and prostate classes and the mean IoU for every last epoch of the semantic segmentation architectures we trained.

Architecture	Background IoU Score	Tool IoU Score	Prostate IoU Score	Mean IoU Score
UNet-MobileNet	0.9286	0.7182	0.6969	0.7812
UNet-Vgg16	0.9356	0.7434	0.7119	0.7970
UNet-ResNet	0.9427	0.7479	0.7296	0.8067
SegNet-MobileNet	0.9274	0.6948	0.6856	0.7693
SegNet-Vgg16	0.9378	0.7449	0.7109	0.7979
SegNet-ResNet	0.9392	0.7441	0.7062	0.7965
PSPNet-MobileNet	0.8862	0.6337	0.6073	0.7091
PSPNet-Vgg	0.9357	0.7120	0.6965	0.7814
PSPNet-ResNet	0.9285	0.7133	0.6853	0.7757

Table 5.1. IoU Scores for Semantic Segmentation Architectures.

As we had already foreseen from the accuracy trend in Figure 5.1, the mean IoU values demonstrate that all the architectures have an adequate test accuracy, which is always higher than 0.7 for the mean. Tool and prostate classes seem to assume lower values, between 0.6 and 0.7. These values are due to the IoU metric, which works pixel by pixel, but the visual results are optimum for almost all the architectures. Figure 5.2 shows examples of outputs obtained by testing our trained models on a specific video frame:

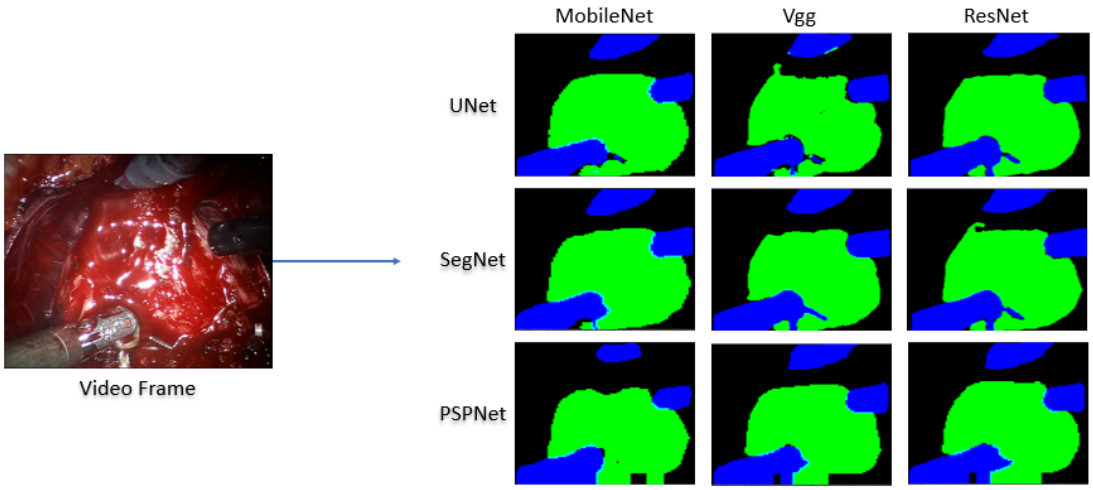


Figure 5.2. Segmentation Neural Networks testing outputs for every combination between semantic segmentation architectures (rows) and neural network models (columns).

Among the nine combinations between semantic segmentation and neural networks architectures, we considered the UNetResnet architecture as the best semantic segmentation model, according to its IoU score. It had the highest IOU score for the prostate class, which is the most relevant class to correctly identify for our research. This result was one of the most presumable because, as already said in Section 2.4.1, the UNet architecture often works better for medical images. The UNet architecture presents, indeed, skip connections which allow this model to capture also tiny details.
Figure 5.3 shows a comparison between UNetResnet training and validation accuracy progressions:

Figure 5.3. UnetResnet Training and Validation Accuracy.

Both the training and the validation accuracy assume optimal results. The former is always greater than 0.95 except for the first epochs, while the latter oscillates between 0.87 and 0.94.
As already said in the previous Chapter, to check if an epoch different from the last could give a better result, we saved all the 50 iterations. To further improve the model performance, we searched if a precedent epoch could provide better accuracy. To do so, we selected the epochs which have the highest validation accuracy, with a value greater than 0.93, and we tested the corresponding models.
Table 5.2 shows the validation accuracy and the IoU scores for the iterations examined, compared with the last one. As we can see from Table 5.2, the best IoU score for the prostate class is given by epoch number 47, so we assumed that with a lower number of iterations, the model could not learn enough, with a higher number of iteration, it probably starts to overfit. Therefore, basing on the results

obtained from our testing images, we chose the $47th$ epoch for our final Semantic Segmentation model.

Epoch	Background IoU Score	Tool IoU Score	Prostate IoU Score	Mean IoU Score
10	0.9393	0.7455	0.7132	0.7993
23	0.9433	0.7525	0.7258	0.8072
34	0.9385	0.7594	0.7164	0.8048
42	0.9424	0.7530	0.7339	0.8097
43	0.9413	0.7503	0.7274	0.8064
45	0.9434	0.7531	0.7331	0.8098
47	0.9437	0.7623	0.7344	0.8135
49	0.9429	0.7554	0.7327	0.8103
50	0.9427	0.7479	0.7296	0.8067

Table 5.2. IoU Scores for the best UNet-MobileNet training epochs.

5.2 Rotation Neural Network

To address the rotation estimation task, we tried different configurations before reaching a satisfactory accuracy level. This task was troublesome because the prostate has a roundish shape, and detecting the exact rotation is complex also for a human. For this reason, to make the task easier to understand for the machine, we decided to first train a model with a manually created synthetic dataset, described in Section 4.4.1. This custom dataset allowed obtaining images where the organ had a more discernible shape and orientation than the video frames available. Furthermore, the Blender script also provided the actual rotation of the object, arduous to tag manually. In the pictures extracted from real videos, the organ has almost the same texture as the surrounding tissues, so determining the rotation would have become troublesome. We generated the synthetic renderings through *Blender*, which allowed us to change parameters and obtain the rotation values directly. As already said in the previous Chapter, we choose a classification-based model to address the rotation problem. We ran different combinations of branches to determine the final architecture described in Section 4.4.3. We maintained the *ResNet50* base model (Figure 4.10) in the first part of the network for all the configurations, and we modified only the last part of the network. The training phases of the Rotation Neural Network models were slower than the Segmentation Neural Network ones. It required 6 minutes for each epoch on average. For a first evaluation, we trained for 15 iterations a model with three branches

that addressed the rotation issues for the three axes (Figure 4.6). The training and validation accuracy exposed different behaviour according to the rotation axis:

- *X-Axis* branch achieved an optimal level of accuracy, around 0.9 both on training and validation set;
- *Y-Axis* and *Z-Axis* reached an accuracy close to 1 during the training phase, but the validation accuracy grew slowly, and, at the end of the training process, it was less than 0.2 and 0.1, respectively.

This first neural network configuration demonstrated that the model learned much more slowly for the *Y* and *Z-Axis*. For this reason, we decided to train three different models for the three axes to check if the performance improved. The following figures (Figure 5.4, Figure 5.5, Figure 5.6) present a branches comparison between the *XYZ_Classification* model and the models trained only for a single axis classification:

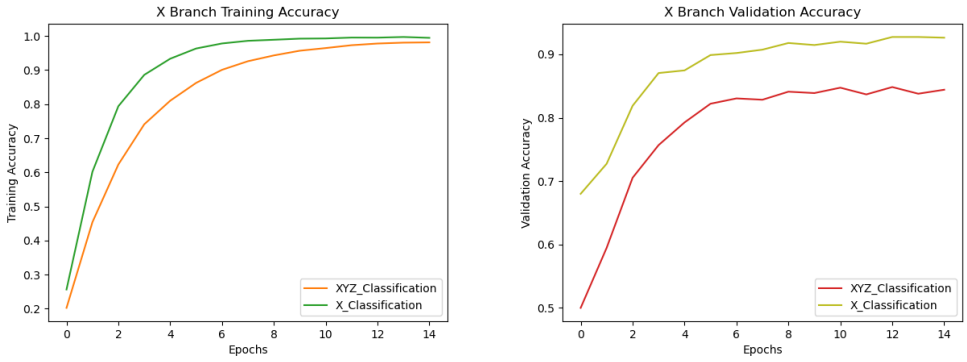


Figure 5.4. X Branch with 15 epochs: Training and Validation Accuracy.

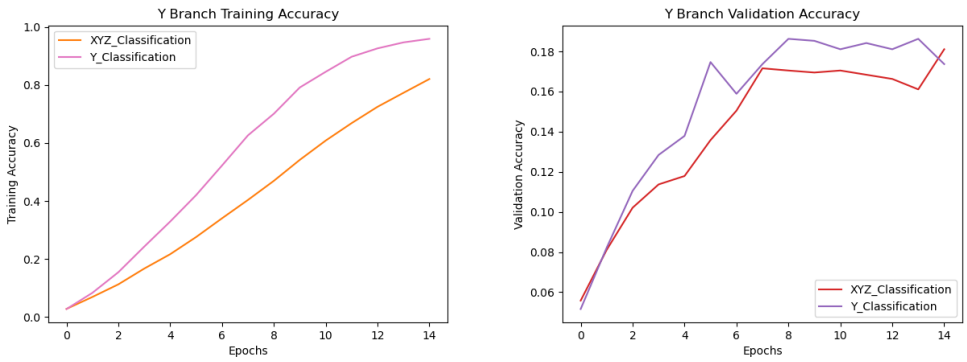


Figure 5.5. Y Branch with 15 epochs: Training and Validation Accuracy.

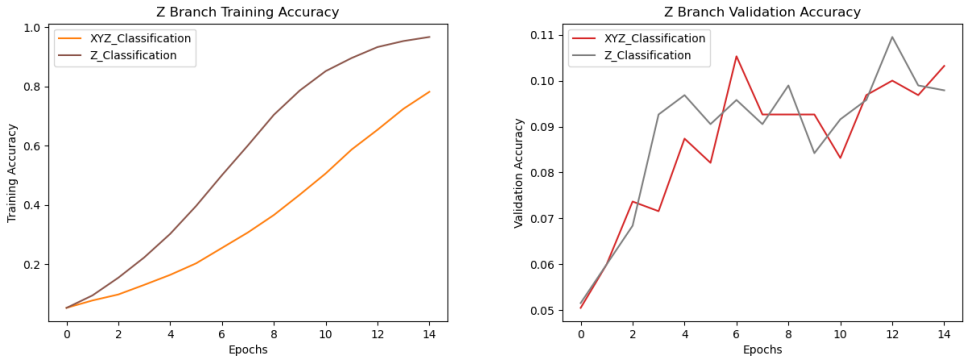


Figure 5.6. Z Branch with 15 epochs: Training and Validation Accuracy.

With this new configuration of the network, nothing changed about the accuracy. Therefore, we tried to improve the number of epochs to 30. The X-Axis model ran for all the 30 iterations, but the accuracy saturated after the first 15, as Figure 5.7 displays.

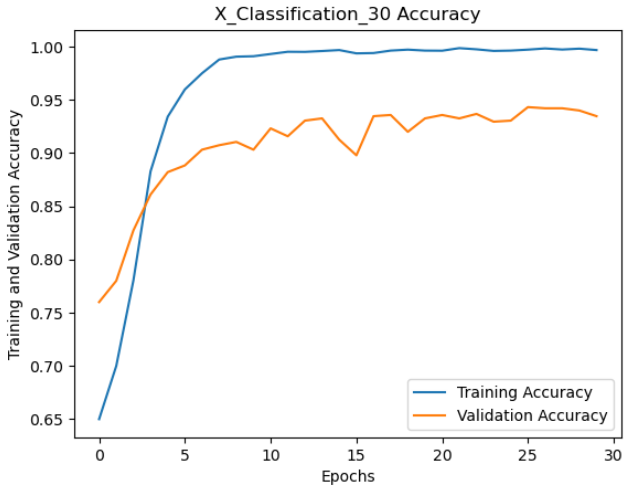


Figure 5.7. X Branch with 30 epochs: Training and Validation Accuracy.

The simulation of the Y and Z axis early stopped after 24 epochs because of overfitting. The training accuracies for both the models grew and were close to 1. The validation accuracy progressions remained almost flat with a higher number of epochs, too, as shown in Figure 5.8.

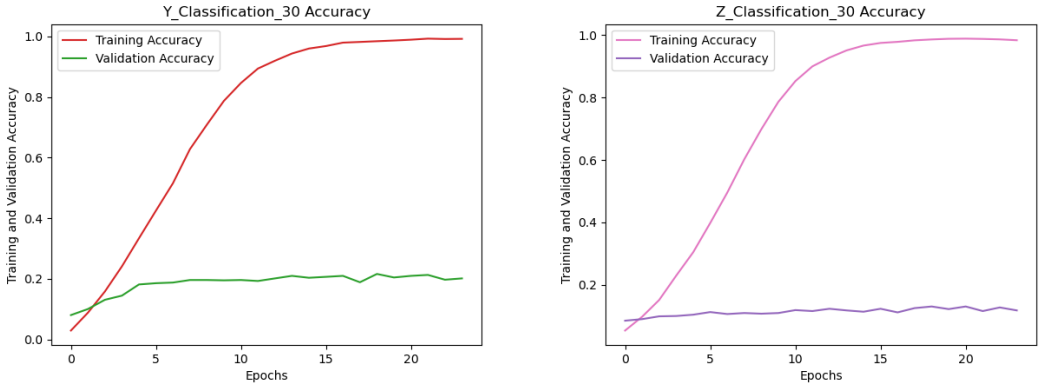


Figure 5.8. Y and Z Branches with 30 epochs: Training and Validation Accuracy.

At this point, since the *Z-Axis* was the worst performance and its rotation estimation was not relevant, we tried to train a model only with *X-Axis* and *Y-Axis* branches. Moreover, from the accuracy progression showed in Figure 5.4 and Figure 5.5, we noticed that the accuracy progression saturated around 0.9 between epoch 10 and epoch 15 for both the axes, so we decided to train this new model for 13 epochs. Figure 5.9 shows the training accuracy progressions of *X-Axis* and *Y-Axis* for the *XY_Classification* model.

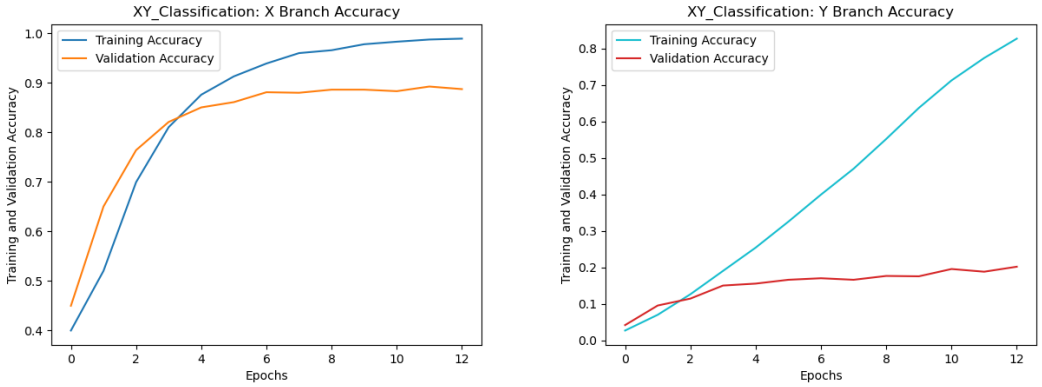


Figure 5.9. XY_Classification model: Training and Validation Accuracy.

The graphics comparison did not show an effective improvement for this model because the graphs are generated according to the exact rotation value. The improvement is visible, instead, from the testing results. To evaluate all the models with the same conditions, we tested each model on 13 iterations. We performed

this first evaluation by testing 1000 synthetic images. As already said, we considered a prediction wrong when the deviation between the predicted and the actual values was greater than 5 degrees. Table 5.3 displays, for each model, the prediction errors for the three branches:

Model	X-Axis Prediction Errors	Y-Axis Prediction Errors	Z-Axis Prediction Errors
XYZ_Classification	18	142	229
XY_Classification	11	100	
X_Classification	11		
Y_Classification		144	
Z_Classification			251

Table 5.3. Rotation models testing results on synthetic images.

The testing results showed that the *XY_Classification* configuration was effectively the best compromise. Once chosen the best performing configuration, we tested the *XY_Classification* model on different epochs to check if a previous one performed better. Table 5.4 shows the testing errors on 1000 synthetic images, separately for each branch, from epoch 9 to epoch 13:

Epoch	X-Axis Predictions Errors	Y-Axis Predictions Errors
9	9	104
10	10	108
11	12	104
12	12	95
13	11	100

Table 5.4. Rotation Neural Network testing results on synthetic images from epoch 9 to epoch 13.

As we can observe from the table, epoch 12 achieved the best testing results, so we chose this iteration as the final model for the Rotation Neural network.

At this point, despite our model was trained on synthetic images, we decided to examine its behaviour on surgical pictures. To do so, we created a testing set of 36 items by extracting frames from the available surgical videos. The first step was to assign the rotation values to the prostate depicted on the testing frames,

to have a criterion of comparison with the network output. We manually tagged the images through an ad-hoc graphical tool, shown in Figure 5.10.

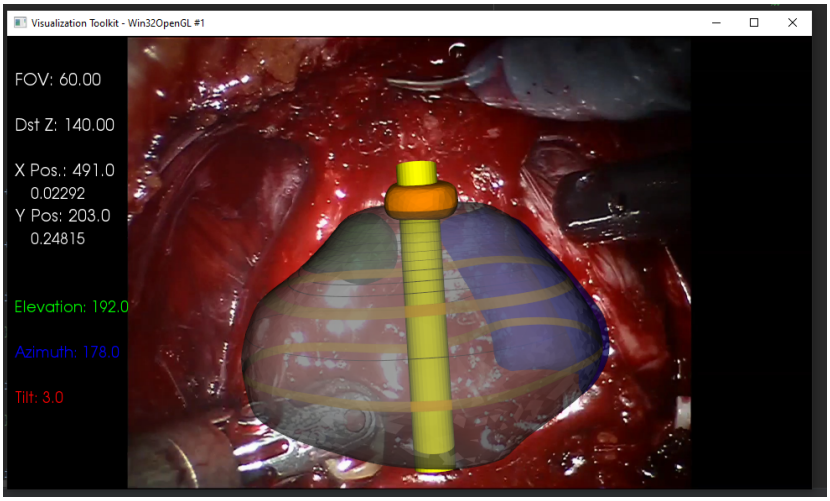


Figure 5.10. Rotation Tagger.

The tool interface displayed a 3D virtual prostate model over its physical counterpart. It has several functionalities: we could translate the model, rotate it around the three axes, modify the opacity of its components (such as cancer masses, catheter, and the prostate itself) according to our needs. Once tagged all the images, the tool produced a file containing axes rotation as output. We involved these values to compare the predicted value and the actual one. As for the synthetic images, we repeated the test on the same epochs, from 9 to 13. Table 5.5 displays the testing results on a total of 36 images:

Epoch	X-Axis Predictions Errors	Y-Axis Predictions Errors
9	15	24
10	14	28
11	11	28
12	11	23
13	15	25

Table 5.5. Rotation Neural Network testing results on real images from epoch 9 to epoch 13.

In this second test situation, epoch 12 was the best performing too. Therefore, we

obtained a second proof to elect this epoch as the final model.

Observing Table 5.4 and Table 5.5, we remark a higher percentage of errors for the test on surgical images. This result is due to the Rotation Neural Network was trained with synthetic images. Moreover, the test set made of video frames was too close, and not all the items clearly showed the organ because of occlusions by the surgical tools and the blood, which made the prostate less identifiable. Despite this, the results are acceptable, and the error is lower than 10 degrees for almost all cases. This rotation angle is troublesome to detect for a human, too. Furthermore, the network failed with a cluttered scene or non-optimal lighting condition; when the prostate was occluded or its texture was easily confused with the surrounding tissues. This result was relevant for our research because the goal is to detect and perform the automatic registration on a frame in which the organ is visible and recognizable. The following figure (Figure 5.11) shows the Rotation Neural Network results for three different situations:

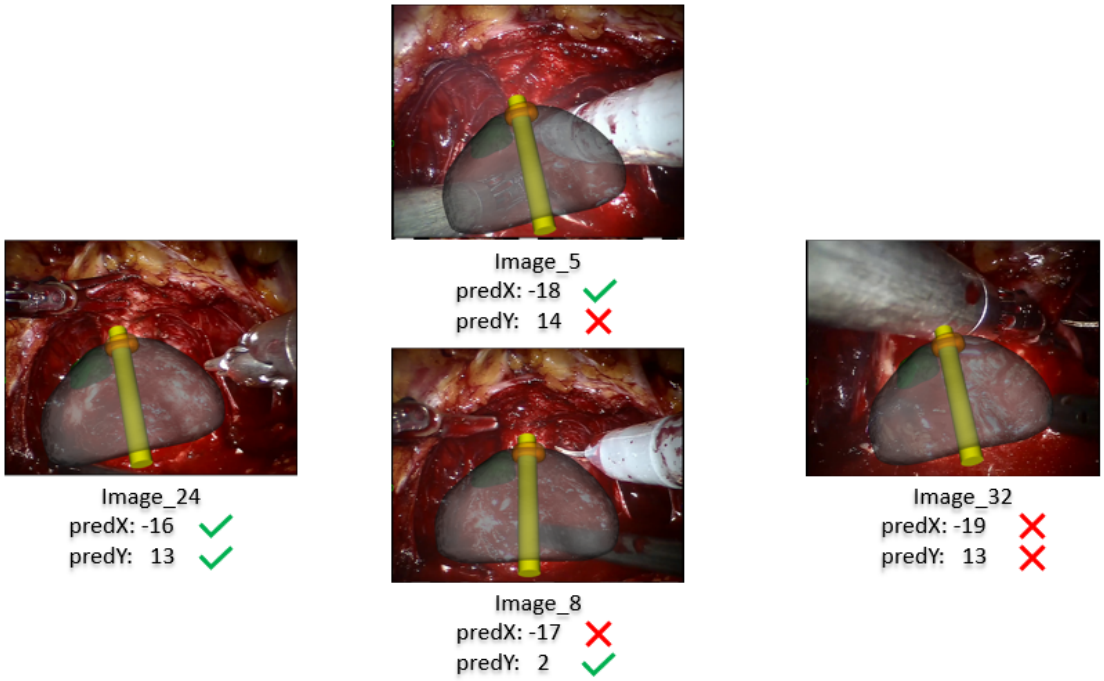


Figure 5.11. Perfect Rotation Detection (left); the prediction failed one axis (centre); the prediction failed both the axes (right).

Table 5.6 displays the epoch 12 testing results for each image.

Test Image	X-Axis Actual Value	X-Axis Predicted Value	X-Axis Prediction Result	Y-Axis Actual Value	YAxis Predicted Value	Y-Axis Prediction Result
0	-15	-17	✓	-3	13	✗
1	-15	-17	✓	12	13	✓
2	-15	-17	✓	7	13	✗
3	-15	-33	✗	7	19	✗
4	-15	-32	✗	8	13	✓
5	-15	-18	✓	8	14	✗
6	-15	-19	✓	2	14	✗
7	-19	-18	✓	-8	13	✗
8	-10	-17	✗	4	2	✓
9	-13	-17	✓	7	14	✗
10	-15	-17	✓	5	19	✗
11	-25	-16	✗	5	13	✗
12	-20	-19	✓	8	13	✓
13	-17	-16	✓	21	13	✗
14	-15	-18	✓	2	13	✗
15	-18	-18	✓	2	13	✗
16	-18	-17	✓	5	13	✗
17	-15	-16	✓	-1	2	✓
18	-18	-19	✓	12	13	✓
19	-10	-17	✗	-1	13	✗
20	-13	-17	✓	-1	13	✗
21	-10	-16	✗	2	13	✗
22	-12	-17	✓	16	13	✓
23	-10	-18	✗	2	13	✗
24	-16	-16	✓	13	13	✓
25	-15	-17	✓	13	16	✓
26	-12	-35	✗	4	14	✗
27	-17	-17	✓	3	19	✗
28	-18	-17	✓	11	14	✓
29	-18	-32	✗	11	19	✗
30	-10	-18	✗	3	13	✗
31	-17	-19	✓	10	13	✓
32	-10	-19	✗	0	13	✗
33	-12	-17	✓	-11	-12	✓
34	-12	-17	✓	-15	19	✗
35	-17	-17	✓	15	19	✓

Table 5.6. Rotation Neural Network testing results on real images.

With this work, we tried to extend the previous similar model focused on the catheter to the defatting and incision of the endopelvic fascia and vase clamping and nerve-sparing stages. Doctors have already tested the model applied to the targeted biopsy stage, and it gave support during the operations. This new model has not been tested by doctors yet. The task was more arduous to address because we had to work directly on the prostate without any visual element as a reference point. However, we hope this new model may improve the existing augmentation strategies used during the other two RARP procedure steps.

5.3 Conclusions and Future Works

Recently, Augmented Reality and Deep Learning have been actively used in the medical environment. These technologies help doctors in diagnosis and prognosis, during pre-operative and post-operative phases, and in-vivo surgery. In the urologic field, these technologies are used together with the robotic-assisted and minimally invasive procedure to help doctors in operations planning and during delicate surgeries. In this work, we focused on the Robot-Assisted Radical Prostatectomy, and the goal was to introduce a new registration strategy that involved neural networks. The first step was to analyze the existing methods for real-time 6D pose estimation from monocular images. We reviewed techniques belonging to different fields of research. Then we classified them into three main categories: Template-based methods, Feature-based methods, and Learning-Based methods. Our proposed method divided the problem into two phases. We first involved a Semantic Segmentation Network to detect the region of interest, which allows understanding the position and the scale the 3D virtual model should have for a correct superimposition. The second step was finding the solution for the rotation issue. We involved a Rotation Neural Network, which solved a classification problem to detect the X-Axis and the Y-Axis rotation. The former neural network achieved optimal results. The latter worked well with synthetic images, but it had some limits to generalize on real ones due to the complexity of the problem and the lack of clearly visible input data. However, this is just the beginning: to achieve optimal performance and overcome the limits of this approach, as, for example, the limited number of images, future works could train the Rotation Neural Network directly with surgical data, with greater availability of surgical videos provided by doctors. This improvement could exploit the new visualization tool to manually tag the input data and create a custom dataset of surgical images to train the network. Hopefully, these changes will improve the performance of our work. In this way, we could obtain an increasingly efficient tool to help doctors during the different phases of the Robot-Assisted Radical Prostatectomy procedure.

Bibliography

[1] T. C. Chang, C. Seufert, O. Eminaga, E. Shkolyar, J. C. Hu, and J. C. Liao, "Current trends in artificial intelligence application for endourology and robotic surgery," *Urologic Clinics*, vol. 48, no. 1, pp. 151–160, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0094014320300690.

[2] M. Gribaudo, P. Piazzolla, F. Porpiglia, E. Vezzetti, and M. G. Violante, "3d augmentation of the surgical video stream: Toward a modular approach," *Computer methods and programs in biomedicine*, vol. 191, p. 105505, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0169260719322801.

[3] P. Vávra, J. Roman, P. Zonča, P. Ihnát, M. Němec, J. Kumar, N. Habib, and A. El-Gendi, "Recent development of augmented reality in surgery: a review," *Journal of healthcare engineering*, vol. 2017, 2017. [Online]. Available: https://www.hindawi.com/journals/jhe/2017/4574172/.

[4] E. Checcucci, D. Amparore, C. Fiori, M. Manfredi, M. Ivano, M. Di Dio, G. Niculescu, F. Piramide, G. Cattaneo, P. Piazzolla *et al.*, "3d imaging applications for robotic urologic surgery: an esut yauwp review," *World journal of urology*, vol. 38, no. 4, pp. 869–881, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s00345-019-02922-4/.

[5] L. M. Huynh and T. E. Ahlering, "Robot-assisted radical prostatectomy: a step-by-step guide," *Journal of endourology*, vol. 32, no. S1, pp. S–28, 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6071518/.

[6] F. Porpiglia, E. Checcucci, D. Amparore, M. Manfredi, F. Massa, P. Piazzolla, D. Manfrin, A. Piana, D. Tota, E. Bollito *et al.*, "Three-dimensional elastic augmented-reality robot-assisted radical prostatectomy using hyperaccuracy three-dimensional reconstruction technology: a step further in the identification of capsular involvement," *European*

urology, vol. 76, no. 4, pp. 505–514, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0302283819302702.

[7] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. [Online]. Available: http://aima.cs.berkeley.edu/index.html

[8] S. S. Islam, S. Rahman, M. M. Rahman, E. K. Dey, and M. Shoyaib, "Application of deep learning to computer vision: A comprehensive study," in *2016 5th international conference on informatics, electronics and vision (ICIEV)*. IEEE, 2016, pp. 592–597. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7760071.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10166/.

[11] M. Nielsen, *Neural Networks and Deep Learning*, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/.

[12] A. Amini, A. Soleimany, S. Karaman, and D. Rus, "Spatial uncertainty sampling for end-to-end control," *arXiv preprint arXiv:1805.04829*, 2018. [Online]. Available: https://arxiv.org/abs/1805.04829/.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986. [Online]. Available: https://www.nature.com/articles/323533a0

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/726791

[15] D. Gupta. A beginner's guide to deep learning based semantic segmentation using keras. [Online]. Available: https://divamgupta.com/image-segmentation/2019/06/06/deep-learning-semantic-segmentation-keras.html

[16] Keras. Batch normalization layer. [Online]. Available: https://keras.io/api/layers/normalization_layers/batch_normalization/.

[17] T. Huang, "Computer vision: Evolution and promise," 1996. [Online]. Available: https://cds.cern.ch/record/400313/files/p21.pdf

[18] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Science and Information Conference*. Springer, 2019, pp. 128–144. [Online]. Available: https://arxiv.org/abs/1910.13796/.

[19] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, "Deep learning for computer vision: A brief review," *Computational intelligence and neuroscience*, vol. 2018, 2018. [Online]. Available: https://www.hindawi.com/journals/cin/2018/7068349/.

[20] Mathworks. Semantic segmentation. [Online]. Available: https://ch.mathworks.com/solutions/image-video-processing/semantic-segmentation.html

[21] J. Jordan. An overview of semantic image segmentation. [Online]. Available: https://www.jeremyjordan.me/semantic-segmentation/.

[22] A. Syed and B. T. Morris, "Sseg-lstm: semantic scene segmentation for trajectory prediction," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 2504–2509. [Online]. Available: https://ieeexplore.ieee.org/document/8813801.

[23] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018. [Online]. Available: https://arxiv.org/abs/1811.03378/.

[24] J. Lieman-Sifry, M. Le, F. Lau, S. Sall, and D. Golden, "Fastventricle: cardiac segmentation with enet," in *International Conference on Functional Imaging and Modeling of the Heart*. Springer, 2017, pp. 127–138. [Online]. Available: https://arxiv.org/abs/1704.04296v1/.

[25] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890. [Online]. Available: https://arxiv.org/abs/1612.01105/.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241. [Online]. Available: https://arxiv.org/abs/1505.04597/.

[27] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7803544.

[28] E. Muñoz, Y. Konishi, C. Beltran, V. Murino, and A. Del Bue, "Fast 6d pose from a single rgb image using cascaded forests templates," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4062–4069. [Online]. Available: https://ieeexplore.ieee.org/document/7759598.

[29] F. Liu, P. Fang, Z. Yao, R. Fan, Z. Pan, W. Sheng, and H. Yang, "Recovering 6d object pose from rgb indoor image based on two-stage detection network with multi-task loss," *Neurocomputing*, vol. 337, pp. 15–23, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925231218315236.

[30] X. Li, Y. Cai, S. Wang, and T. Lu, "Learning category-level implicit 3d rotation representations for 6d pose estimation from rgb images," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2019, pp. 2310–2315. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8961408.

[31] G. Zuo, C. Zhang, H. Liu, and D. Gong, "Low-quality rendering-driven 6d object pose estimation from single rgb image," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9207286.

[32] C. Capellen, M. Schwarz, and S. Behnke, "Convposecnn: Dense convolutional 6d object pose estimation," *arXiv preprint arXiv:1912.07333*, 2019. [Online]. Available: https://arxiv.org/abs/1912.07333/.

[33] W. Zhao, S. Zhang, Z. Guan, H. Luo, L. Tang, J. Peng, and J. Fan, "6d object pose estimation via viewpoint relation reasoning," *Neurocomputing*, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0925231220300333.

[34] C. Sahin, G. Garcia-Hernando, J. Sock, and T.-K. Kim, "A review on object pose recovery: from 3d bounding box detectors to full 6d pose estimators," *Image and Vision Computing*, p. 103898, 2020. [Online]. Available: https://arxiv.org/abs/2001.10609/.

[35] S. Zakharov, I. Shugurov, and S. Ilic, "Dpod: 6d pose object detector and refiner," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1941–1950. [Online]. Available: https://ieeexplore.ieee.org/document/9010850.

[36] R. Habib, M. Saii *et al.*, "Object pose estimation in monocular image using modified fdcm," *Computer Science*, vol. 21, no. 1, 2020. [Online]. Available: https://www.researchgate.net/publication/338846350_Object_Pose_Estimation_in_Monocular_Image_Using_Modified_FDCM/.

[37] T.-T. Do, M. Cai, T. Pham, and I. Reid, "Deep-6dpose: Recovering 6d object pose from a single rgb image," *arXiv preprint arXiv:1802.10367*, 2018. [Online]. Available: https://arxiv.org/abs/1802.10367/.

[38] Y. Liu, L. Zhou, H. Zong, X. Gong, Q. Wu, Q. Liang, and J. Wang, "Regression-based three-dimensional pose estimation for texture-less objects," *IEEE Transactions on Multimedia*, vol. 21, no. 11, pp. 2776–2789, 2019. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8698890.

[39] M. Ulrich, C. Wiedemann, and C. Steger, "Combining scale-space and similarity-based aspect graphs for fast 3d object recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 10, pp. 1902–1914, 2011. [Online]. Available: https://ieeexplore.ieee.org/document/6112769.

[40] Y. Konishi, Y. Hanzawa, M. Kawade, and M. Hashimoto, "Fast 6d pose estimation from a monocular image using hierarchical pose trees," in *European Conference on Computer Vision*. Springer, 2016, pp. 398–413. [Online]. Available: https://link.springer.com/chapter/10.1007%2F978-3-319-46448-0_24/.

[41] E. Muñoz, Y. Konishi, V. Murino, and A. Del Bue, "Fast 6d pose estimation for texture-less objects from a single rgb image," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 5623–5630. [Online]. Available: https://ieeexplore.ieee.org/document/7487781.

[42] H. Tjaden, U. Schwanecke, and E. Schomer, "Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 124–132. [Online]. Available: https://ieeexplore.ieee.org/document/8237285.

[43] Z. Cao, Y. Sheikh, and N. K. Banerjee, "Real-time scalable 6dof pose estimation for textureless objects," in *2016 IEEE International conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 2441–2448. [Online]. Available: https://ieeexplore.ieee.org/document/7487396.

[44] N. Payet and S. Todorovic, "From contours to 3d object detection and pose estimation," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 983–990. [Online]. Available: https://ieeexplore.ieee.org/document/6126342.

[45] E. Corona, K. Kundu, and S. Fidler, "Pose estimation for objects with rotational symmetry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7215–7222. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8594282.

[46] F. Massa, B. C. Russell, and M. Aubry, "Deep exemplar 2d-3d detection by adapting from real to rendered views," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 6024–6033. [Online]. Available: https://arxiv.org/abs/1512.02497/.

[47] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-dof object pose from semantic keypoints," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2011–2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7989233.

[48] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "Pvnet: Pixel-wise voting network for 6dof pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4561–4570. [Online]. Available: https://ieeexplore.ieee.org/document/8954204.

[49] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2147–2156. [Online]. Available: https://ieeexplore.ieee.org/document/7780605.

[50] Z. Zhao, G. Peng, H. Wang, H.-S. Fang, C. Li, and C. Lu, "Estimating 6d pose from localizing designated surface keypoints," *arXiv preprint arXiv:1812.01387*, 2018. [Online]. Available: https://arxiv.org/abs/1812.01387/.

[51] W. Zhao, S. Zhang, Z. Guan, W. Zhao, J. Peng, and J. Fan, "Learning deep network for detecting 3d object keypoints and 6d poses," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 134–14 142. [Online]. Available: https://ieeexplore.ieee.org/document/9157824.

[52] J. Nath Kundu, A. Ganeshan, and R. Venkatesh Babu, "Object pose estimation from monocular image using multi-view keypoint correspondence," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0–0. [Online]. Available: https://arxiv.org/abs/1809.00553/.

[53] C. Chen, X. Jiang, W. Zhou, and Y.-H. Liu, "Pose estimation for texture-less shiny objects in a single rgb image using synthetic training data," *arXiv preprint arXiv:1909.10270*, 2019. [Online]. Available: https://arxiv.org/abs/1909.10270/.

[54] Z. Li, Y. Hu, M. Salzmann, and X. Ji, "Robust rgb-based 6-dof pose estimation without real pose annotations," *arXiv preprint arXiv:2008.08391*, 2020. [Online]. Available: https://arxiv.org/abs/2008.08391/.

[55] M. Rad and V. Lepetit, "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3828–3836. [Online]. Available: https://ieeexplore.ieee.org/document/8237675.

[56] M. Oberweger, M. Rad, and V. Lepetit, "Making deep heatmaps robust to partial occlusions for 3d object pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 119–134. [Online]. Available: https://arxiv.org/abs/1804.03959/.

[57] J. Liu and S. He, "6d object pose estimation without pnp," *arXiv preprint arXiv:1902.01728*, 2019. [Online]. Available: https://arxiv.org/abs/1902.01728/.

[58] ——, "6d object pose estimation based on 2d bounding box," *arXiv preprint arXiv:1901.09366*, 2019. [Online]. Available: https://arxiv.org/abs/1901.09366/.

[59] Y. Hu, J. Hugonot, P. Fua, and M. Salzmann, "Segmentation-driven 6d object pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3385–3394. [Online]. Available: https://ieeexplore.ieee.org/document/8953567.

[60] Z. Li, G. Wang, and X. Ji, "Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7678–7687. [Online]. Available: https://ieeexplore.ieee.org/document/9009519.

[61] K. Park, T. Patten, and M. Vincze, "Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 7668–7677. [Online]. Available: https://ieeexplore.ieee.org/document/9008819.

[62] X. Zhang, Z. Jiang, and H. Zhang, "Real-time 6d pose estimation from a single rgb image," *Image and Vision Computing*, vol. 89, pp. 1 – 11, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885619300964.

[63] L. Kästner, D. Dimitrov, and J. Lambrecht, "A markerless deep learning-based 6 degrees of freedom poseestimation for with mobile robots using rgb data," *arXiv preprint arXiv:2001.05703*, 2020. [Online]. Available: https://arxiv.org/abs/2001.05703/.

[64] J. Liu, S. He, Y. Tao, and D. Liu, "Realtime rgb-based 3d object pose detection using convolutional neural networks," *IEEE Sensors Journal*, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8868108.

[65] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1521–1529. [Online]. Available: https://ieeexplore.ieee.org/document/8237431.

[66] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37. [Online]. Available: https://arxiv.org/abs/1512.02325/.

[67] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2686–2694. [Online]. Available: https://arxiv.org/abs/1505.05641/.

[68] J. Josifovski, M. Kerzel, C. Pregizer, L. Posniak, and S. Wermter, "Object detection and pose estimation based on convolutional neural networks trained with synthetic data," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6269–6276. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8594379.

[69] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang, "Gs3d: An efficient 3d object detection framework for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1019–1028. [Online]. Available: https://ieeexplore.ieee.org/document/8954005.

[70] Y. Lu, S. Kourian, C. Salvaggio, C. Xu, and G. Lu, "Single image 3d vehicle pose estimation for augmented reality," in *2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2019, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8969201.

[71] S. He, G. Liang, F. Chen, X. Wu, and W. Feng, "Object recognition and 3d pose estimation using improved vgg16 deep neural network in cluttered scenes," in *Proceedings of the International Conference on Information Technology and Electrical Engineering 2018*, 2018, pp. 1–7. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3148453.3306266/.

[72] A. V. Patil and P. Rabha, "A survey on joint object detection and pose estimation using monocular vision," *arXiv preprint arXiv:1811.10216*, 2018. [Online]. Available: https://arxiv.org/abs/1811.10216/.

[73] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *arXiv preprint arXiv:1711.00199*, 2017. [Online]. Available: https://arxiv.org/abs/1711.00199/.

[74] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6d object pose prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 292–301. [Online]. Available: https://arxiv.org/abs/1711.08848/.

[75] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788. [Online]. Available: https://ieeexplore.ieee.org/document/7780460.

[76] P. Poirson, P. Ammirato, C.-Y. Fu, W. Liu, J. Kosecka, and A. C. Berg, "Fast single shot detection and pose estimation," in *2016 Fourth International Conference on 3D Vision (3DV)*. IEEE, 2016, pp. 676–684. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7785144.

[77] S. Mahendran, H. Ali, and R. Vidal, "3d pose regression using convolutional neural networks," in *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2017, pp. 2174–2182. [Online]. Available: https://arxiv.org/abs/1708.05628/.

[78] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082. [Online]. Available: https://ieeexplore.ieee.org/document/8100080.

[79] M. Sundermeyer, Z.-C. Marton, M. Durner, and R. Triebel, "Augmented autoencoders: Implicit 3d orientation learning for 6d object detection," *International Journal of Computer Vision*, vol. 128, no. 3, pp. 714–729, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11263-019-01243-8/.

[80] K. Hara, R. Vemulapalli, and R. Chellappa, "Designing deep convolutional neural networks for continuous object orientation estimation," *arXiv preprint arXiv:1702.01499*, 2017. [Online]. Available: https://arxiv.org/abs/1702.01499/.

[81] J. Rambach, C. Deng, A. Pagani, and D. Stricker, "Learning 6dof object poses from synthetic single channel images," in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE, 2018, pp. 164–169. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8699254.

[82] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946. [Online]. Available: https://arxiv.org/abs/1505.07427/.

[83] B. Xu and Z. Chen, "Multi-level fusion based 3d object detection from monocular images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2345–2353. [Online]. Available: https://ieeexplore.ieee.org/document/8578347.

[84] J. Wu, B. Zhou, R. Russell, V. Kee, S. Wagner, M. Hebert, A. Torralba, and D. M. Johnson, "Real-time object pose estimation with pose interpreter networks," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6798–6805. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8593662.

[85] J. Ku, A. D. Pon, and S. L. Waslander, "Monocular 3d object detection leveraging accurate proposals and shape reconstruction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 867–11 876. [Online]. Available: https://ieeexplore.ieee.org/document/8954143.

[86] D. M. Montserrat, J. Chen, Q. Lin, J. P. Allebach, and E. J. Delp, "Multi-view matching network for 6d pose estimation," *arXiv preprint arXiv:1911.12330*, 2019. [Online]. Available: https://arxiv.org/abs/1911.12330/.

[87] Y. Hu, P. Fua, W. Wang, and M. Salzmann, "Single-stage 6d object pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2930–2939. [Online]. Available: https://ieeexplore.ieee.org/document/9156435.

[88] Y. Wang, S. Jin, and Y. Ou, "A multi-task learning convolutional neural network for object pose estimation," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2019, pp. 284–289. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8961594.

[89] C. Koetsier, T. Peters, and M. Sester, "Learning the 3d pose of vehicles from 2d vehicle patches," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 43, pp. 683–688, 2020. [Online]. Available: https://www.researchgate.net/publication/343623538_LEARNING_THE_3D_POSE_OF_VEHICLES_FROM_2D_VEHICLE_PATCHES/.

[90] M. E. Banani, J. J. Corso, and D. F. Fouhey, "Novel object viewpoint estimation through reconstruction alignment," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3113–3122. [Online]. Available: https://arxiv.org/abs/2006.03586/.

[91] K. Wada. [Online]. Available: https://github.com/wkentaro/labelme

[92] Scikit-learn. [Online]. Available: https://scikit-learn.org/stable/.

[93] D. Gupta. Keras-segmentation. [Online]. Available: https://github.com/divamgupta/image-segmentation-keras/.

[94] OpenCV. [Online]. Available: https://opencv.org/.

[95] TensorFlow. [Online]. Available: https://www.tensorflow.org/.

[96] Keras. [Online]. Available: https://keras.io/.

[97] J. Peng, S. Kang, Z. Ning, H. Deng, J. Shen, Y. Xu, J. Zhang, W. Zhao, X. Li, W. Gong *et al.*, "Residual convolutional neural network for predicting response of transarterial chemoembolization in hepatocellular carcinoma from ct imaging," *European radiology*, vol. 30, no. 1, pp. 413–424, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s00330-019-06318-1/.

[98] Keras. Dense layer. [Online]. Available: https://keras.io/api/layers/core_layers/dense/.

[99] ——. Dropout layer. [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/.

[100] ——. Activation layer. [Online]. Available: https://keras.io/api/layers/core_layers/activation/.

[101] ——. Earlystopping callback. [Online]. Available: https://keras.io/api/callbacks/early_stopping/.

[102] ——. Modelcheckpoint callback. [Online]. Available: https://keras.io/api/callbacks/model_checkpoint/.

[103] ——. Tensorboard callback. [Online]. Available: https://keras.io/api/callbacks/tensorboard/.

[104] TensorFlow. Tensorboard. [Online]. Available: https://www.tensorflow.org/tensorboard/.

[105] Keras. Categorical crossentropy. [Online]. Available: https://keras.io/api/losses/probabilistic_losses/#categoricalcrossentropy-class

[106] ——. Intersection-over-union. [Online]. Available: https://keras.io/api/metrics/segmentation_metrics/.

[107] J. Jordan. Evaluating image segmentation models. [Online]. Available: https://www.jeremyjordan.me/evaluating-image-segmentation-models/.

[108] T. data science. Intersection over union (iou) calculation for evaluating an image segmentation model. [Online]. Available: https://towardsdatascience.com/intersection-over-union-iou-calculation-\for-evaluating-an-image-segmentation-model-8b22e2e84686/.

[109] H.-C. Chen, W. Jia, Z. Li, Y.-N. Sun, and M. Sun, "3d/2d model-to-image registration for quantitative dietary assessment," in *2012 38th Annual Northeast Bioengineering Conference (NEBEC)*. IEEE, 2012, pp. 95–96. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6206979.

[110] F. Casado, D. P. Losada, A. Santana-Alonso *et al.*, "Pose estimation and object tracking using 2d images," *Procedia Manufacturing*, vol. 11, pp. 63–71, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978917303384.

[111] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold *et al.*, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *Proceedings of the IEEE conference on computer*

vision and pattern recognition, 2016, pp. 3364–3372. [Online]. Available: https://ieeexplore.ieee.org/document/7780735.

[112] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau, "Accurate 3d car pose estimation," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 3807–3811. [Online]. Available: https://ieeexplore.ieee.org/document/7533072.

[113] Y. Zhang, C. Zhang, M. Rosenberger, and G. Notni, "6d object pose estimation algorithm using preprocessing of segmentation and keypoint extraction," in *2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, 2020, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9128980.

[114] S. Joung, S. Kim, H. Kim, M. Kim, I.-J. Kim, J. Cho, and K. Sohn, "Cylindrical convolutional networks for joint object detection and viewpoint estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 163–14 172. [Online]. Available: https://ieeexplore.ieee.org/document/9156810.

[115] L. Zhang, C. Gu, C. Gu, K. Wu, and X. Guan, "Semantic translation with convolutional encoder-decoder networks for viewpoint estimation," in *2017 11th Asian Control Conference (ASCC)*. IEEE, 2017, pp. 1660–1665. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8287423.

[116] J. Wang and W. Yan, "Fast pose estimation for texture-less objects based on b-rep model," *EURASIP Journal on Image and Video Processing*, vol. 2018, no. 1, p. 117, 2018. [Online]. Available: https://link.springer.com/article/10.1186/s13640-018-0359-6/.

[117] J. Yang, S. Wang, and G. Liu, "Viewpoint estimation in images by a key-point based deep neural network," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 2551–2555. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8803273.

[118] X. Yang and X. Jia, "6d pose estimation with two-stream net," in *ACM SIGGRAPH 2020 Posters*, 2020, pp. 1–2. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3388770.3407423/.

[119] D. Rodriguez, F. Huber, and S. Behnke, "Category-level 3d non-rigid registration from single-view rgb images," *arXiv preprint arXiv:2008.07203*, 2020. [Online]. Available: https://arxiv.org/abs/2008.07203/.

[120] R. P. Singh, I. Kumagai, A. Gabas, M. Benallegue, Y. Yoshiyasu, and F. Kanehiro, "Instance-specific 6-dof object pose estimation from minimal annotations," in *2020 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2020, pp. 109–114. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9026239.

[121] A. A. Novikov, D. Lenis, D. Major, J. Hladůvka, M. Wimmer, and K. Bühler, "Fully convolutional architectures for multiclass segmentation in chest radiographs," *IEEE transactions on medical imaging*, vol. 37, no. 8, pp. 1865–1876, 2018. [Online]. Available: https://arxiv.org/abs/1701.08816/.

[122] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/358669.358692/.

Acknowledgements

Alla fine di questo percorso il minimo che io possa fare è dedicare delle parole a tutte le persone che mi hanno supportata e sopportata durante questi anni.

In primo luogo, ringrazio di cuore il prof. Pietro Piazzolla e Leonardo per la loro gentilezza e disponibilità, per avermi messa sin dall'inizio a mio agio e avermi permesso di dare il mio piccolo contributo al loro interessante lavoro di ricerca. Non avrei mai immaginato di dedicarmi a qualcosa che riguardasse l'ambito medico ma, come sempre accade, ogni cosa che a prima vista mi spaventa finisce per intrigarmi e affascinarmi incredibilmente e così è stato anche in questo caso!

Grazie a mamma e papà, che mi hanno insegnato l'impegno e il sacrifico e mi hanno messa nelle condizioni di fare la scelta giusta dandomi sempre la possibilità di farlo da sola. Questo traguardo è il minimo che potessi fare per ripagare i loro sacrifici, grazie ai quali ho avuto la possibilità di studiare così tanto lontano da casa. La distanza non è semplice da gestire e sappiamo che una parte del mio cuore rimane sempre lì e una parte del loro parte con me. Grazie per aver sopportato con tanta pazienza i momenti di sconforto ed aver assecondato tutte le mie pazzie, compreso un salto nel vuoto da 4000 metri.

Grazie a mio fratello, nonostante i nostri due caratteri forti facciano spesso scintille, noi sappiamo benissimo che ci siamo sempre l'una per l'altro e che tutti i litigi sono solo particolari dimostrazioni d'affetto.

Grazie ai miei nonni, quelli che ho la fortuna di avere con me e quelli che non ci sono più, perchè ovunque si trovino so benissimo che mi sono stati sempre accanto. Grazie ai miei zii, cugini e a quella pazza comitiva di amici che, conoscendomi da sempre, ormai sono la mia seconda famiglia, anche se confessano che avrebbero preferito conoscermi dopo i primi tre anni di vita.

Grazie alle mie amiche di sempre, a Fosca, Gabriella, Greta e Martina, nonostante come in tutti i legami più forti ci siano alti e bassi, sappiamo sempre come ritrovarci e volerci bene. Grazie per esserci state nei periodi più impegnativi e per aver compreso tutti i miei "no, devo studiare".

Grazie a Dario, per la sua capacità di trovarsi al posto giusto nel momento giusto anche a più di mille chilometri di distanza, grazie per avermi regalato un sorriso anche quando non ne avevo voglia e per le foto del mare quando ne sentivo la mancanza.

Grazie al Collegio Einaudi e ovviamente al Quarto Piano, la mia grande famiglia di Torino. Sono arrivata in collegio con due valigie piene di insicurezze e tanta voglia di tornare a casa, ma grazie al gruppo che si è formato ho realizzato che vivere a Torino non era poi così male. Grazie soprattutto a coloro che non si sono limitati ad essere dei semplici coinquilini, ma sono diventati un pezzo di cuore e sono parte dei miei amici più cari; grazie a chi si è dimostrata la sorella che non ho mai avuto, a chi mi ha ascoltata e dato consigli di ogni tipo, alle "passeggiate" in terrazza, agli allenamenti di gruppo, ai concerti in cucina, a chi ha condiviso con me i giorni e le notti per i lavori di gruppo e a chi mi ha fatto scoprire ed apprezzare la cucina etnica in piena pandemia!

Grazie davvero a tutti voi, per aver creduto in me anche quando non ci credevo nemmeno io! E grazie anche alle grandi delusioni, perchè mi hanno resa più forte!

All'inizio di questo percorso, il primo approccio con questa disciplina era stato talmente burrascoso da spingermi quasi a mollare tutto. Eppure, qualcuno mi aveva detto che sarebbe stata proprio questa la mia strada. Dopo cinque anni, grazie a questa strana "maledizione" che puntualmente mi fa inesorabilmente apprezzare ciò che a prima vista odio, posso dire che aveva ragione! Ecco, spero che il mio "non so cosa voglio, ma voglio arrivarci", il mio non accontentarmi mai, il non essere mai soddisfatta di me stessa e tentare di andare oltre i miei limiti continuino ad accompagnarmi sempre, dandomi la possibilità di continuare imparare e migliorarmi. Sono consapevole che questo non è un traguardo, ma un punto di partenza per nuove sfide che spero di essere all'altezza di affrontare!