POLITECNICO DI MILANO
School of Industrial and Information Engineering
Department of Electronics, Information and Bioengineering
Master of Science Degree in Computer Science and Engineering

3D Position Estimation Using Deep Learning

Internal Supervisor: Prof. Giacomo Boracchi
Politecnico di Milano

External Supervisor: Prof. Magnus Boman
Royal Institute of Technology

Master Thesis's by:
Filippo Pedrazzini, 879110

Academic Year 2017-2018

Alla mia famiglia.

Abstract

The estimation of the 3D position of an object is one of the most critical topics in the computer vision field. Where the final aim is to create automated solutions that can localize and detect objects from images, new high-performing models and algorithms are needed. Due to lack of relevant information in the single 2D images, approximating the 3D position of an object can be considered a complex problem. The single specific task of estimating the 3D position of a soccer ball has been investigated. This thesis describes a method based on two deep learning models: the *ball* net and the *temporal* net that can tackle this task. The former is a deep convolutional neural network with the intention to extract meaningful features from the images, while the latter exploits the temporal information to reach a more robust prediction. This solution reaches a better Mean Absolute Error compared to already existing computer vision methods on different conditions and configurations. A new data-driven pipeline has been created to deal with videos and extract the 3D information of an object.

Keywords: *Convolutional Neural Network, Deep Learning, Computer Vision, 3D Object Localization.*

Sommario

La stima della posizione 3D di un oggetto può essere considerata uno degli aspetti più importanti nell'ambito intelligenza artificiale. Dove l'obiettivo finale è quello di creare soluzioni automatizzate in grado di localizzare e rilevare oggetti dalle immagini, è necessario lo sviluppo di nuovi modelli e algoritmi ad alte prestazioni. A causa della mancanza di informazioni rilevanti nelle singole immagini 2D, l'approssimazione della posizione 3D di un oggetto può essere considerato un problema complesso. Il singolo studio relativamente alla stima della posizione 3D di una palla da calcio è stata investigata in modo specifico. Questa tesi descrive un metodo basato su due modelli neurali: *ball* net e *temporal* net. La prima è una rete neurale convoluzionale con l'intenzione di estrarre caratteristiche significative dalle immagini, mentre la seconda sfrutta le informazioni temporali per ottenere una stima più accurata. Questa soluzione raggiunge un errore assoluto medio migliore rispetto ai metodi di computer vision esistenti in condizioni e configurazioni diverse. Una nuova data-driven pipeline è stata creata per gestire video ed estrarre le informazioni 3D di un oggetto.

Acknowledgments

I would like to thank professor Boracchi for giving me support during the thesis. I would also like to thank the people at KTH who supported me, most notably my supervisor Magnus Boman and my examiner Henrik Boström.

Moreover, I would like to thank Matteo Presutto for giving me access to his GPU server, making the training of the models possible.

Finally, I want to express my profound gratitude to my family who helped and supported me during these years in every occasion and each decision I took. Thank you for the love that you gave me.

Milan, July 12, 2018
Filippo Pedrazzini

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Network.

BCE Binary Cross Entropy.

FPS Frames Per Second.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MSE Mean Squared Error.

ReLU Rectified Linear Unit.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

Chapter 1

Introduction

This chapter has the objective to give an introduction to the problem and purpose of the work, ending up with the description of how the thesis is structured.

1.1 Background

Since the beginning, the objective of computer vision was to build a machine able to simulate the human ability of understanding, processing and extracting high-level features from images and videos, with the final aim to create automated solutions that could imitate human tasks with a high level of accuracy. The goal is to make computers able to recognize, detect and classify objects without any human assistant. The field has been investigated for a long time due to the exciting aspects of automation and the multiple use cases in which it can be exercised. After years of development in the same direction, where the application of geometric-based solutions has been manipulated, a revolution in the field happened in 2013 [1]; the first deep learning approach came out to solve an object detection task without any assumption on the environment. From that moment on, deep learning has been used to address more and more difficult tasks, creating a worldwide competition for the fastest and most accurate algorithm.

When dealing with soccer statistics a manual solution is still used to gather real-time highlights. The aim for an automatic solution is to reconstruct real-time statistics from soccer videos matches without involving the human labor. At the moment, the market has different products to achieve a data-driven analysis of games and performances. More specifically, the primary objective of these solutions is to accurately detect each player and the ball in the field to compute some essential highlights about the match.

In the case of soccer, highlights are highly correlated with the 3D position of each object in the field. With the ball and players information, a simple analytical method

can be used to have aggregated data useful for additional improvements and analysis. A data-driven approach can be used to detect the object position in the image. The first strategies came out years ago following a geometrical computer vision solution: [2], [3], [4], [5], [6], [7] and [8]. All of them shares the same paradigm used to find a final solution:

1. Computing the projection matrix.
2. Detecting the object within the image.
3. Estimating the 3D position of the object.

On the other hand, the deep learning growth led to more performing models that can tackle, and that can be engaged to find more robust and accurate results. The estimation of the object's position relative to the court system of coordinates requires advanced and robust techniques. This work introduces a solution to accurately predict the 3D position of an object using the union of deep learning strategies.

1.2 Problem

In just a few years, a massive revolution in the computer vision industry took place, moving the research toward the deep learning field. Considering these significant advancements, estimating the object coordinates in an image can be seen as a solved problem, but when the goal is to reconstruct the 3D position from a video coming from a single camera, still much work is needed to achieve a good result. From the 2D information, we can reassemble the 3D position of an object under specific requirements; if all the objects lay on the same plane, a comprehensive mapping can be created using the camera matrix; a deep explanation regarding the geometrical problem will take place in the next chapter. In the soccer domain, the players preserve one coordinate equal to zero, which is the requirement needed to reconstruct their position in the court coordinates system. For this reason, the 2D location within the images that can be extracted using state of the art deep learning techniques are enough to build a model able to deliver statistics about the teams. While the players tracking can be easily solved, the estimation of the ball 3D position can be recognized as a more complicated issue. The object size and the physic behaviour make the task more challenging to face, both concerning detection and reconstruction. The presence of the third coordinate makes the 3D prediction more complicated, and the solution cannot just be based on the previously mentioned camera matrix. Camera matrix, projection matrix and camera calibration, they all refer to the same process of computing the model that maps the 3D coordinates system to the camera one (2D). In the following chapter, a detailed intuition about the concepts will take place.

[2] and [5] tried to solve the same problem using domain-specific information regarding the soccer field and the physical constraints of the ball. On the other hand, deep learning state of the art techniques were not available ten years ago, and the need to create a more recent and advanced procedure employing new artificial intelligence methods is required.

Following the discussion; this work addresses the problem of estimating the 3D location of an object starting from 2D videos; more specifically, having a sequence of images from a single fixed camera, a data-driven algorithm should predict the 3D position of an object relative to the environment system of coordinates. Given these assumptions, the implicit research question regards the success of the solution.

"Having a single fixed camera is it possible to extract the 3D position of an object accurately using a data-driven approach?"

1.3 Goal

The goal of this work is to investigate and create a method that can solve the problem of estimating the 3D coordinates of a soccer ball without any assumption of the court characteristics starting from a video. The combination of deep learning techniques could lead to a significant result concerning accuracy. Due to the lack of meaningful existing solutions, some metric criteria have been created to evaluate the work. A robust solution would mean to create a product or an architecture that can be adapted to other applications. Some steps are required to achieve this result:

1. Collecting data: creating a comprehensive dataset to train and test the designed models.
2. Define a model to solve the task: iterating over the assumptions and configurations a deep learning model should be designed to predict the labels accurately.
3. Tune the parameters: adjusting the hyper-parameters for better performances.
4. Analyzing the results: reflecting over the obtained scores to possible further improvements.

1.3.1 Benefits, Ethics and Sustainability

The introduction of a method able to capture the 3D position of an object having as input a sequence of images can be useful for many different applications. Where the requirement is to localize an object in the 3D space having a single camera, the employment of the same paradigm can be lead to a performing estimator. On the other

hand, when dealing with the specific case of soccer, the introduction of highly accurate performing computer-based solutions can substitute the currently manual work of gathering statistics. The need for maintenance and performance assurance is still required, but different expertise is needed to face the task of managing a smart machine.

From a sustainability perspective, many are the domains in which the estimation of the 3D position of an object can bring added value and help improving people life. An example is the Augmented Reality industry where most of the applications address the problem of helping sick people in the daily life creating products that can help them to track and localize objects in the environment. Another example is the autonomous driving cars industry; the core technology has been changed from sensor-based solutions to computer vision methods. The possibility to precisely track people and obstacles is required to have a safe machine.

1.4 Methodology

An empirical approach has been followed to achieve a robust result. Due to the limited number of related works, different implementations have been tested before ending up with a meaningful answer. A quantitative analysis was mandatory to find the most suitable model. A quantitative data collection was required to train and test the models. On the other hand, the quality of the performances is strictly related to the domain field in which the solution is applied. For this reason, a qualitative analysis associated with the effectiveness of the answer has been discussed toward the obtained results. A single case study has been investigated to validate the effectiveness of the architecture, which can not be considered enough to generalize the solution to different domains.

The complexity of the problem led to the first decision of splitting the issue into different and simpler tasks:

1. 3D trajectory extraction: the reconstruction of the 3D position of the ball while doing a trajectory (flying).
2. Flying - not flying classification: the binary classification regarding the flying:not-flying position of the ball.
3. Object detection and Size estimation: the detection of the 2D coordinates of the ball as well as the estimation of the size of the object in the 2D images.

The result is the union of two deep learning models *ball* net and *temporal* net; the former is a convolutional neural network that has the objective to extract the images information; while the latter is a shallow network with the intent of extracting the temporal

information for a better prediction. In Figure 1.1 an illustration of the final model. A final Mean Absolute Error (MAE) of 1.3 has been reached using the combination of these two networks in the prediction of the (x, y, z) position of the ball in the soccer environment.

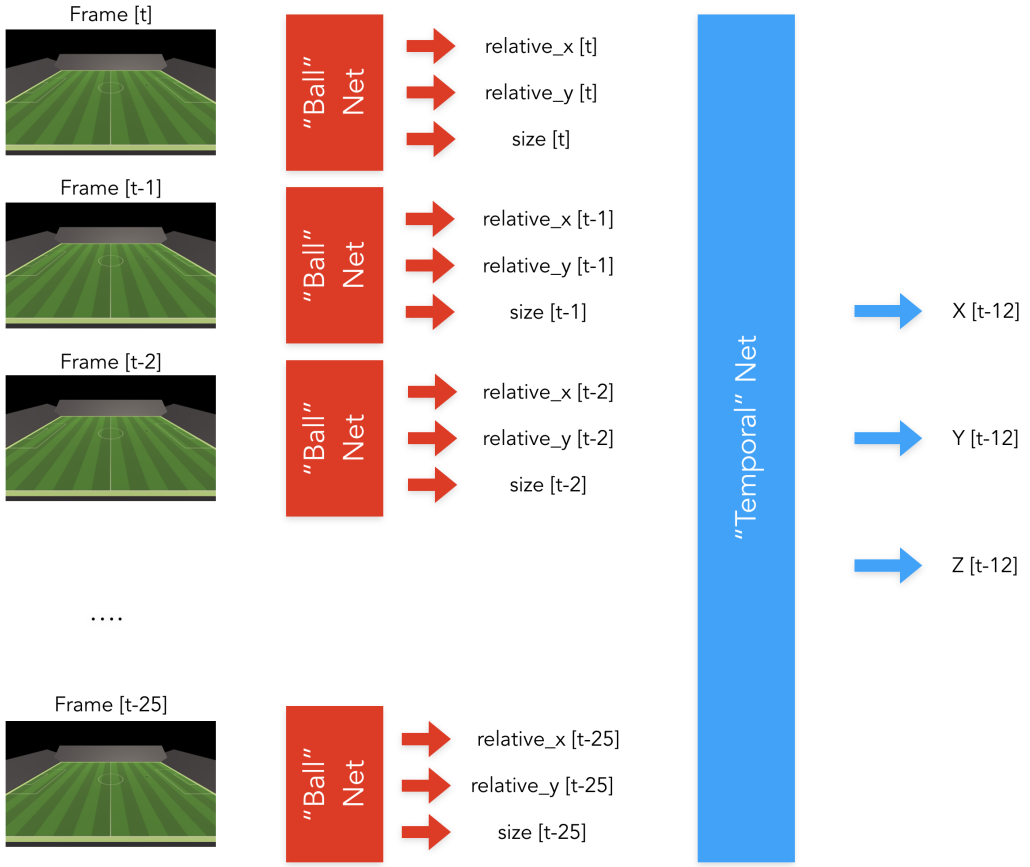


Figure 1.1. The final pipeline used for the prediction.

Due to lack of data, a synthetic dataset using Unity [9] has been created to simulate a real-case scenario and to test the methods over a meaningful simulation, providing scores relative to the given environment.

This work illustrates the followed path to achieve a performing model. Each step represents a move toward the direction of finding a suitable and robust solution for the problem mentioned above. The design choices are explained progressively in order to deliver to the reader a logical flow of decisions and improvements.

1.5 Delimitations

The main limitation regards the data employed when dealing with this specific task; a synthetic dataset has been used to validate the architecture, but in case of a real dataset the same performances could not have been reached. A secondary limitation regards the generalization of the solution. The work addresses the problem of finding a data-driven approach that could be employed in different domains, but here a single case study has been investigated. Where specific configurations and characteristics regarding the data are not satisfied, the same performances could not be obtained compared to the investigated task.

1.6 Outline

The second chapter gives a detailed intuition of the related works of the field and how the thesis can benefit from the investigation of these methods. The basic concepts are explained to give the reader the possibility to understand all the thesis technologies and phases. Moving on, the third chapter includes the explanation of the empirical method followed to create a robust implementation to solve the explained research question effectively. Next, the fourth chapter presents the results obtained using the models designed and explained in chapter three. The fifth chapter illustrates the possible applications and problems of the implemented product. Finally, the sixth chapter concludes answering the research question and possible further improvements.

Chapter 2

Theoretic Background

The chapter contains the literature study performed to have insight regarding the state of art approaches in the computer vision field and the related papers that tried to address the same research problem. The basic concepts regarding machine learning are explained in order to let the reader understand all the thesis phases.

2.1 Geometrical Concepts

2.2 Prerequisites

2.2.1 Homography

A *homography* is a 2D projective transformation from one system of coordinates to another. A *homography* H maps 2D points in *homogeneus coordinates* according to:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.1}$$

Homogeneus Coordinates are a useful representation; if we normalize having $w = 1$ we obtain the unique image coordinates (x, y). For this reason, the *homography H (*3×3 *matrix* has eight independent degrees of freedom.

2.2.2 The Direct Linear Transformation

Homographies can be computed using the points of two system of coordinates. As mentioned above a *homography* has eight degrees of freedom. Each point correspondence

gives two equations, one each for the x and y coordinates, and for this reason four points are required to compute H.

The direct linear transformation (DLT) is an algorithm for computing H given four or more mappings. Solving the equation $Ah = 0$ where A is the matrix containing two rows for each mapping, H can be computed.

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' & x_1' \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y_1' & y_1y_1' & y_1' \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' & x_2' \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y_2' & y_2y_2' & y_2' \\ & \vdots & & \vdots & & \vdots & & \vdots & \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0 \tag{2.2}$$

2.2.3 The Camera Matrix

The camera matrix can be decomposed as:

$$P = K[R|T] \tag{2.3}$$

- R (3×3) in 2.3: is a rotation matrix that describes the orientation of the camera.
- T (3×1) in 2.3: is 3D translation vector that explains the position of the camera center.
- K (3×3) in 2.3: is the intrinsic calibration matrix that describes the projection properties of the camera.

The intrinsic matrix depends only on the camera properties and is in a general form written as:

$$K = \begin{bmatrix} af & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

- f: is *focal length* which corresponds to the distance between the image plane and the camera center.
- a: is the *aspect ration*; it is used for non-square pixel elements.

With this assumption, the matrix becomes:

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

The remaining points are $c = [c_x, c_y]$ which correspond to the coordinates of the optical centre, the image point where the optical axis intersects the image plane. These values are often assumed as half the width and height of the image.

2.2.4 From 3D to 2D - The Pin-Hole Camera Model

The pin-hole camera model is the relationship between a point in the 3D space and its projection in the image plane. In Figure 2.1 an illustration.

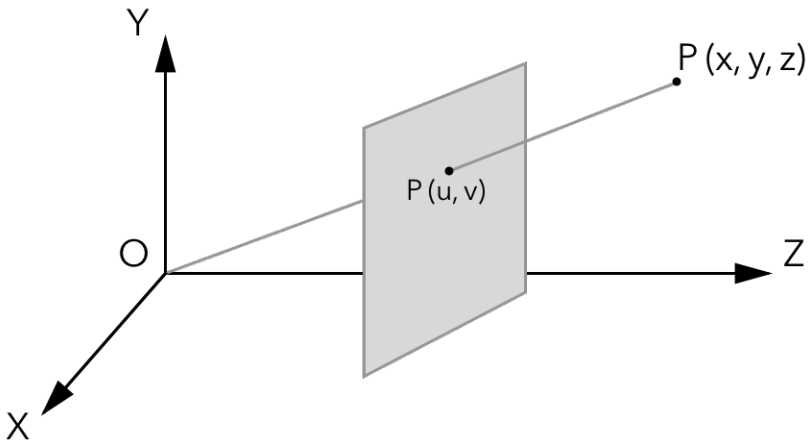


Figure 2.1. The pin-hole camera model.

The relationship can be expressed as:

$$ax = PX \tag{2.6}$$

Where the 3×4 matrix P is the camera matrix explained above. Having the information about the projection matrix, that can be easily computed using the Direct Linear Transformation and some known points (3D and 2D), we can transform each 3D point into the 2D representation from the camera perspective.

2.2.5 From 2D to 3D

While a linear mapping can be identified between the 3D to the 2D representation, the problem of transforming a point from the 2D coordinates of the image plane to the 3D

system of coordinates cannot be easily solved. If we assume the points in the 3D system of coordinates written in terms of homogeneous coordinates (X, Y, Z, K)$^{\mathrm{T}}$ and let the centre of projection be the origin (0, 0, 0, 1)$^{\mathrm{T}}$ with the image points represented in terms of homogeneous coordinates (x, y, z)$^{\mathrm{T}}$; the mapping can be represented, as previously explained, by a 3×4 projection matrix P with the linear system:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P_{3\mathrm{x}4} \begin{bmatrix} x \\ y \\ z \\ k \end{bmatrix} \tag{2.7}$$

From the 2D information, the 3D reconstruction cannot be performed accurately since each point on the projection plane could lie anywhere on a ray from the camera through the plane. In Figure 2.2 an illustration of the problem from a visual perspective.

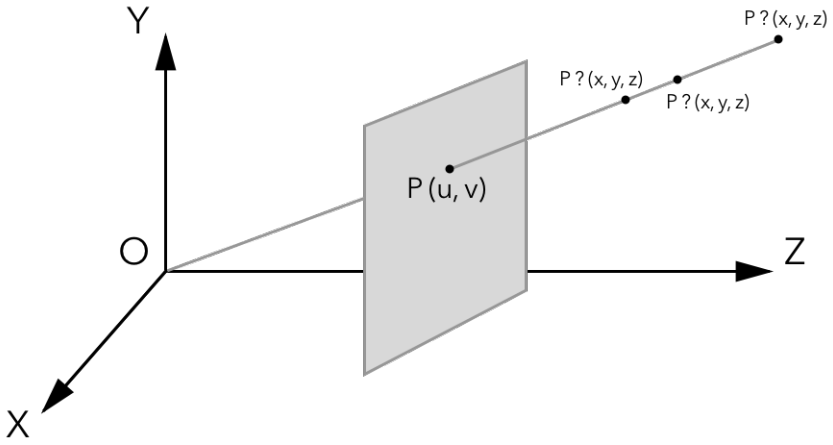


Figure 2.2. The pin-hole camera model with the intrinsic uncertainty in the reconstruction of a 3D point.

Two possible solutions are available to solve the problem:

1. Same Laying Plane: if all the objects belong to the same z plane, the projection matrix can be computed using some of those points constraining the problem to a 2D to 2D mapping in the homogeneus coordinates. Assuming, for example, $z = 0$; the projection matrix can be computed using the Direct Linear Transformation and the linear mapping will be reduced to:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P_{3\mathrm{x}3} \begin{bmatrix} x \\ y \\ t \end{bmatrix} \tag{2.8}$$

2. Distance Between Object and Camera: referring to Figure 2.2, the missing information necessary to correctly reconstruct the position (x, y, z) given (u, v) is the distance between the 3D point and the camera center 0. If for each 2D point we can gather the value w, which represents the missing information, a complete mapping can be computed without taking care of the fact that all the 3D points should lay on the same plane.

2.3 Deep Learning Prerequisites

The objective of this section is to give an insight into the most meaningful concepts that have been employed during the thesis and that actively contributed to the design of a practical solution.

2.3.1 Introduction

AI has been part of the world history for a long time. Starting from the Turing test [10], the willing to create a smart machine increased over the years. Turing predicted that the first intelligent machine able to beat his experiment would have been invented with the opening of the new century (2000). In 2018, still, a performing solution to solve the test has not been developed. Recently, a new prediction by AI worldwide experts resulted in the delay of almost 30 years compared to Turing opinion. On the other hand, the need for a general purpose machine is not always required. For this reason, the first impressive returns in the AI field came out in the last few years, resulting in an incredible extension regarding investments. The exponential growth concerning the computational power and the generated data made these first solutions available. When dealing with AI, several different fields of expertise are present; each one of them has the objective to simulate the specific human behaviour and reach a certain level of accuracy or abstraction. Here an intuition for each macro-category:

1. Reinforcement Learning: agents capable to self-learn the patterns of a specific environment. They are mostly employed to solve optimization problems when an environment is defined, and certain predefined actions can be performed. Recent advancement brought to the combination of this field to the one of deep learning to deal with complex environments creating a new field known as Deep Reinforcement Learning.

2. Deep Learning: represents the branch of AI that deals with unstructured data (e.g., audio, images, speech, text). The growing computational power gave the opportunity to develop deep models able to better perform and generalize over different complex tasks. The ANN invention was the first move toward the investigation of these new set of models.

3. Machine Learning: all the algorithms able to learn a pattern in the data minimizing a defined loss function.

In Figure 2.3 the AI landscape and how the different fields are connected and related to each other.

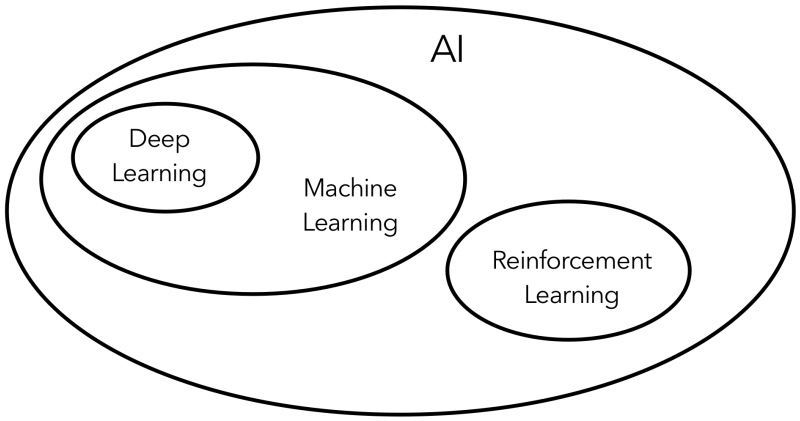


Figure 2.3. The AI landscape in terms of algorithms fields.

The next subsections will take care of explaining the basic deep learning concepts and the algorithms that the work exploited to obtain a final performing pipeline.

2.3.2 ANN

The Perceptron ANN is an AI model able to learn patterns in the data trying to emulate the human brain. In fact, the human brain is composed of many neurons interconnected between each other that process the information and send the transformed signal to the next neuron. The first statistical model has been designed by Rosenblatt with the introduction of the Perceptron [11] in 1957. Even if the model is composed of a single neuron, it represents the basic concept behind the modern neural networks. The objective of this neuron is to learn the mapping between a specific input and a particular output (target). In Figure 2.4 the Perceptron structure is shown, where $x[i]$ represents the input features and y the predicted output.
Each input features $x_1, x_2, x_3, \dots x_n$ is multiplied with a single weight w_i and the obtained values are then summed with a bias term (1) to be fed into the single neuron. The sum is then processed using an activation function resulting into the output y equals to $+1$ if $w_0 + w_1x_1 + \cdots + w_nx_n > 0$, and -1 otherwise, described as $g(\sum_i^N x_i w_i)$. The weights

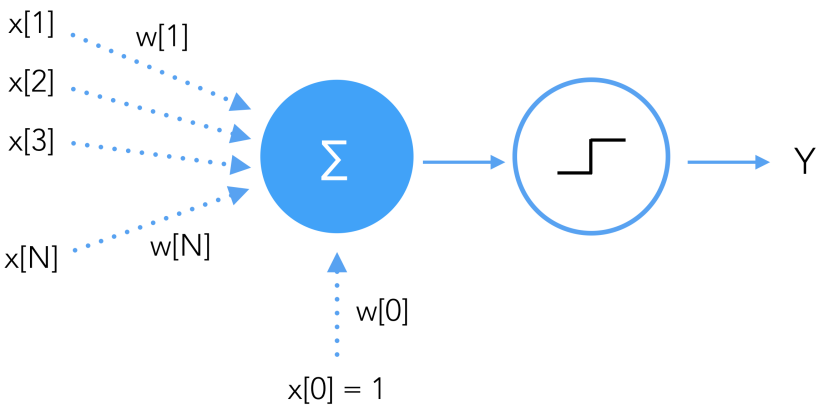


Figure 2.4. The Perceptron architecture.

$(w_0, w_1 \dots w_n)$ are randomly initialized and learned during the training procedure using the Hebbian rule [12]; at each observation, the error is computed between the model prediction and the actual value and then the weights updated accordingly to the rule:

$$w_i \leftarrow w_i + \delta w_i \tag{2.9}$$

$$E = \tilde{y} - y \tag{2.10}$$

$$\delta w_i = \alpha E x_i \tag{2.11}$$

The data pattern is detected minimizing the error function updating the corresponding weights after observing each sample. $\tilde{y}$ represents the actual value and y the Perceptron prediction computed as $y = g(\sum_i^N x_i w_i)$. α describes the learning rate value, which is the dimension of the update step. If the problem is linearly separable, the architecture is guaranteed to converge after a certain number of epochs. When dealing with more complex problems where the decision boundary is not a straight line, the need to add more layers and neurons is required. For this reason, the concept of feed forward neural network has been introduced.

Feed Forward Neural Networks A Feed-Forward Neural Network is an ANN which employs several hidden layers composed of a certain number of neurons. The model learning capabilities are increased due to the presence of a certain amount of neurons each one of them employing non-linear activation functions ($h()$ in Figure 2.5); each layer can be seen as a features transformation step that helps in the identification of

information able to reduce the defined loss function. In Figure 2.5 an example of Feed Forward Neural Network with a single hidden layer is provided.

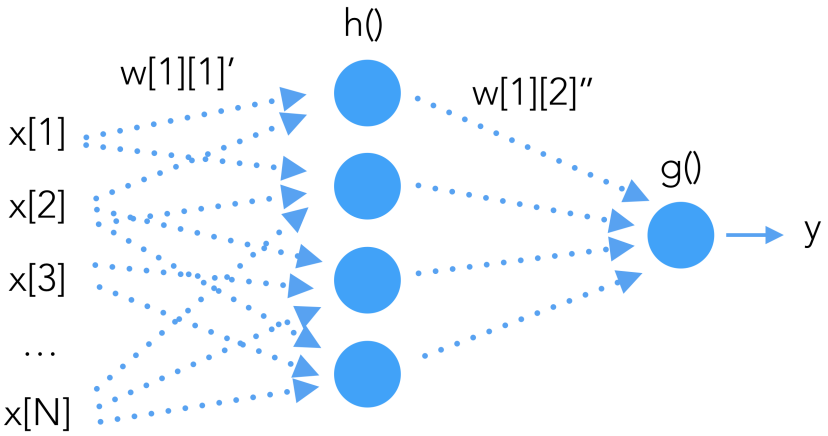


Figure 2.5. Example of Feed Forward Neural Network with a single hidden layer.

As in the Perceptron, the task of finding the weights that best solve the problem of finding a pattern between the input and the output is defined. The difference is represented by the used technique to update the weights. As a matter of fact, *backpropagation* is used to achieve this scope and propagate the error over the different layers to update the weights employing a similar rule as the one of the Perceptron. The update rule is defined as:

$$w_i \leftarrow w_i - \alpha \frac{\partial E(\theta)}{\partial w_i} \tag{2.12}$$

At each observation, the error is computed, and the derivative obtained with respect to the weights propagated through all the layers of the network. The received value is then subtracted to the current weight w_i performing a step in the direction where the loss function is minimized. Since the weights are updated accordingly to the error derivative, a requirement regarding the activation function regards the *differentiability*; the non-linear activation functions are chosen taking care of this need.

2.3.3 Error Functions

Based on the problem that the ANN has to solve a different loss function is required. The majority of the tasks can be divided into two substantial categories: regression and classification. The former represents the work of predicting a continuous target value given a specific input features, while the latter represents the estimation of the

discrete value which the input should belong to. Based on the problem a different loss function is used to deal with the defined task. In case of regression Mean Squared Error (MSE) is used to represents the distance between the prediction and the actual value; while for classification cross entropy is the most suitable to represents the probability to belong to a particular class. Both have been derived considering an assumption on the distribution of the data. In case of regression, the target is normally distributed, while for classification purposes a multinomial distribution represents the distribution of the different classes C.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{2.13}$$

$$E_{entropy} = -\sum_{i=1}^{n}\sum y_{k=1}^{C} y_i'^k \log\left(y_i^k\right) \tag{2.14}$$

2.3.4 ANN Hyperparameters

Hyperparameters are all the parameters that are not learned intrinsically during the training of the model while minimizing the defined loss function. The decision over these kinds of settings is of crucial importance in the deep learning field, on the other hand, they represent the most challenging task in the design/training of a deep learning model; the number of possible combinations and the need to train over a considerable amount of data makes the problem tough to tackle when hardware requirements are not met. Many methods have been investigated to find those values in a faster way taking care of saving concerning hardware performances. When the employment of a deep neural network is required, some design choices are necessaries to create a model able to solve the defined task. Different parameters are present concerning the architecture structure. The decision over activation functions, weights initialization methods, optimizers is a crucial point when developing the network structure. New techniques have been invented to facilitate the training convergence of the model. Indeed, the convergence of a neural network represents the most challenging task in the training procedure of a deep learning model. Often, due to the complexity of the error surface and the presence of many local optima, the difficulty of reaching the optimal solution is present. New optimizers, weights initialization methods and activation functions have been designed to overcome this problem. Following, a brief description of the most effective ways.

2.3.5 Activation Functions

Sigmoid and Tanh represents the most known activation functions; which have been employed for a long time due to non-linearity characteristics and the easiness in the

differentiability. At the same time, they cannot be engaged when many layers are involved in the network design. Taking Sigmoid as the example, the function is displayed in Figure 2.6 with its derivative.

$$S(x) = \frac{1}{1 + \varepsilon^{-x}} \tag{2.15}$$

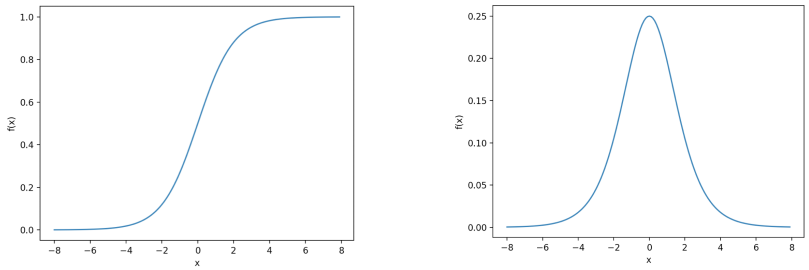


Figure 2.6. On the left the Sigmoid Activation function and on the right the respective derivative.

As above mentioned, the weights are updated using back propagation, which means that each layer propagates the error to the previous layer accordingly to the derivative computed with respect to a specific weight. On the other hand, the maximum value of Sigmoid derivative (in Figure 2.6) is obtained when the input is 0 and is represented by a low number (0.25).
Multiplying a low number recursively at each backward step will result in a gradient amount close to zero in the first hidden layers of the network. This behaviour is called *gradient vanishing*. The same pattern can be seen when employing Tanh as activation function; even if the maximum gradient value is 1, most of the times the input will lay in a range of values that are not close to zero. A new activation function has been designed to solve the task: ReLU.

ReLU described as $y = max(x, 0)$, does not suffer from the gradient vanishing issue since the derivative, if the input is greater than zero, is always 1. The gradient can be propagated over the different layers without taking care of the problem of low values in the first hidden layers. In Figure 2.7 ReLU activation function and its derivative.

On the other hand, ReLU suffers from another problem known as *dying ReLU* which is encountered when the activation function outputs a value that is always zero. Due to the flat characteristic of the activation function, when the input is lower than zero, the activated output will always be zero leading to the *death* of the neuron. At the moment that a neuron is *dead* a difficulty in the recovering phase is present since the gradient will

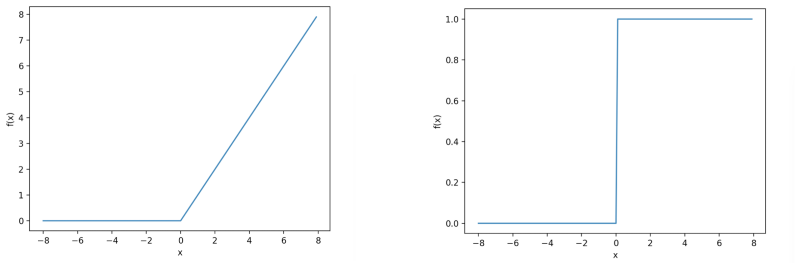


Figure 2.7. On the right ReLU activation function and on the left its derivative.

always be zero and the optimizer will not alter the weights correlated to that neuron. To solve the problem a bias term initialization should be used or another similar activation function employed; Leaky ReLU described as $y = max(x, 0.001)$ is applied to solve the task. The only difference regards the behaviour of the function for values lower than zero. A small amount (0.001) is multiplied by the input to avoid to stack into a constant zero value causing the death of the neuron. In Figure 2.8 the activation function and its derivative are displayed.

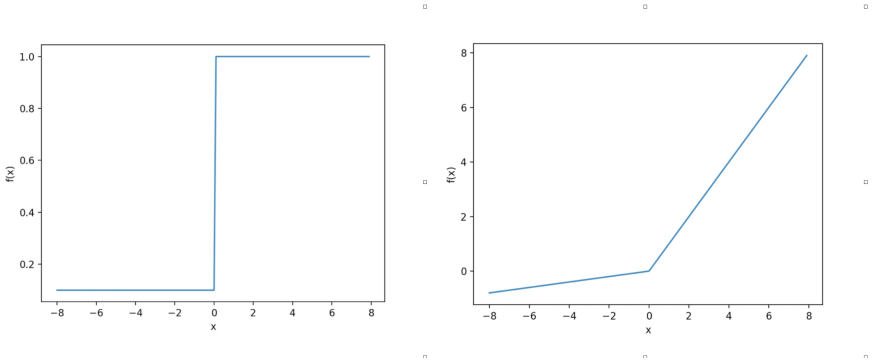


Figure 2.8. The Leaky ReLU activation function and its derivative.

Other activation functions have been designed to improve the convergence of deep learning models (Elu [13], PReLU [14], Swish [15]), but a similar pattern has been exploited compared to ReLU.

2.3.6 Weights Initialization

Weights initialization represents the method used to initialize the weights when the algorithm has not started yet to learn the pattern from the provided data. The starting point is of crucial importance to converge faster and better. Many ways are present to

deal with this scope. Both simple and more complicated techniques are available; here a brief introduction of the possible most known methods is given.

All zero initialization Initializing all the weights to zero can be considered a mistake because during backpropagation all the weights will compute the same gradients resulting in the same updates while different weights should learn different patterns to minimize the loss function.

Small Random Numbers A simple solution is to initialize the weights with a random value close to zero with a Gaussian distribution. Even if the method can be useful in a lot of problems, it does not take into account a crucial aspect: uncalibrated variance. The variance of the randomly initialized weights output grows with the number of inputs (features). Considering $z = \sum_i^N x_i w_i$ with weights w and inputs x the variance of z is:

$$Var(z) = Var(\sum_i^N x_i w_i) \tag{2.16}$$

$$= \sum_i^N Var(x_i w_i) \tag{2.17}$$

$$= \sum_i^N [E(w_i)]^2 Var(x_i) + E[(x_i)]^2 Var(w_i) + = Var(x_i)Var(w_i) \tag{2.18}$$

$$= \sum_i^N Var(x_i)Var(w_i) \tag{2.19}$$

$$= (nVar(w))Var(x) \tag{2.20}$$

Calibrating the Variance To solve the problem mentioned above, we can normalize the assigned value with the number of inputs (fan-in) as following (where $\mathcal{N}$ represents the standard normal distribution):

$$W \sim \mathcal{N}\frac{1}{\sqrt{n}} \tag{2.21}$$

If the objective is to have the variance of z equals to the variance of all its inputs x, the initizalization of each weights w should be $1/n$. With this simple technique better convergence can be obtained because all the neurons in the network initially have approximately the same output distribution.

Xavier Initializer Glorot and Bengio in [16] proposed a new initialization method that takes into account both the number of input features (n_j) and the number of neurons

in the next layer (n_{j+1}), empirically proving a better convergence over complex tasks.

$$Var(W) = \frac{2}{n_j + n_{j+1}} \tag{2.22}$$

2.3.7 Optimizers

The optimizer represents the rule that we use to update the network weights with respect to the obtained derivative. Many different optimizers are present, and a brief intuition for the most famous ones will be lead in the current Subsection.

Gradient Descent Gradient Descent represents the most used optimizer when dealing with deep learning models; the adopted rule is defined in Paragraph 2.3.2 where the explanation of back propagation has been taken. Three different variants of Gradient Descent are available; the main difference regards the amount of data used to compute a single weights update.

1. Batch Gradient Descent: the entire train set is used to compute the gradient and update the weights.
2. Stochastic Gradient Descent: a single example is employed to compute the gradient.
3. Mini-Batch Gradient Descent: a set of samples is considered to update the weights.

In terms of convergence, a difference is present among the different methods. When Stochastic Gradient Descent is employed the convergence could be reached faster due to the more frequent updates (at each seen observation), but a noisy update can be computed; e.g., if the dataset on which we are training the model contains noisy samples when the optimizer will see one of those samples the weights will be updated in a direction that is not the correct one. On the other hand, if a batch is used (Mini-Batch Gradient Descent), a more accurate update is performed due to the lower variance. Even if Mini-Batch Gradient Descent is the most performing solution among the basic methods, it does not perform well when dealing with local minima in more than one dimension.

Momentum Based Optimization Momentum [17] solves the problem mentioned above pushing the optimizer toward the relevant direction, lowering the number of oscillations. The result is achieved by adding a value β of the past time step update vector.

$$w_i^{(t)} = w_i^{(t-1)} + \delta w_i^{(t)} \tag{2.23}$$

$$\delta w_i^{(t)} = -\alpha \frac{\partial E(\theta)}{\partial w_i} + \beta \delta w_i^{(t-1)} \tag{2.24}$$

Another improved version of Momentum Based Optimization is Nesterov Accelerated Gradient; the velocity of the updates are changed more responsevely showing improvements when dealing with Mini-Batch Gradient Descent where the batch is not a small number.

Adaptive Learning Rate One of the problem when dealing with a deep learning model regards the dimension of the step that is used to updates the weights. If the learning rate is too small the convergence could be too slow; at the same time if the learning rate is too large, the model could never get out from a local minimum. In the last years, some advancements in this direction have been taken designing new outperforming optimizers that change the learning rate accordingly during the training. Adam [18], AdaGrad [19], AdaDelta [20] and RMSProp belong to this category of optimizers.

2.3.8 Overfitting and Underfitting

When dealing with machine learning algorithms two possible obstacles that could be faced are *overfitting* and *underfitting*. The former represents the issue of having a model able to learn each pattern in the train data perfectly, but with a low capability to generalize over unseen data, while the latter represents the issue of having a model that is not able to learn the pattern in the data due to the lower feature space represented by the algorithm. In Figure 2.9 an example of underfitting/overfitting models.

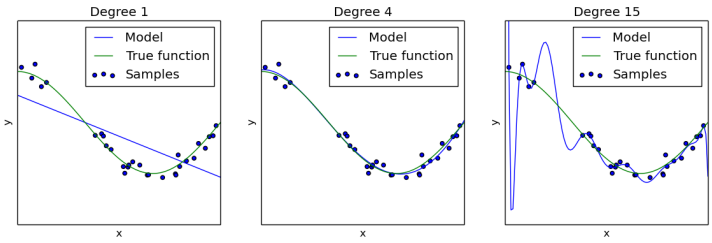


Figure 2.9. Starting from left an example of underfitting, well fitting and overfitting curves.

Underfitting can be easily solved incrementing the model complexity (number of layers/neurons), while the concept of overfitting can be more challenging to tackle and in the next Subsection, some techniques will be explained.

2.3.9 Regularization Techniques

When the problem is not difficult to solve, the design of a machine learning model could quickly lead to the creation of a higher feature space that maps correctly the input with the target variable, but it does not generalize well on unseen data. For this reason, there are few techniques to tackle the problem of overfitting.

Weight Decay A first technique is known as *weight decay*. Indeed, the problem of overfitting is highly correlated to the values of the weights in the network; higher the weights higher the probability of overfitting. The method consists in adding of a penalty term γ to the loss function $E()$, constraining the weights to have a lower value:

$$\tilde{E()} = E() + \lambda\gamma \tag{2.25}$$

lambda represents the regularization term and determines the amount of penalty to put into consideration when computing the error function. There are two forms of regularization: *L2 and L1 regularization*. The former represents the sum of squares of the network parameters in each layer of the network:

$$\gamma = \frac{1}{2}\sum_{i}^{N} w_i^2 \tag{2.26}$$

The latter penalizes the weights using the sum of the absolute values:

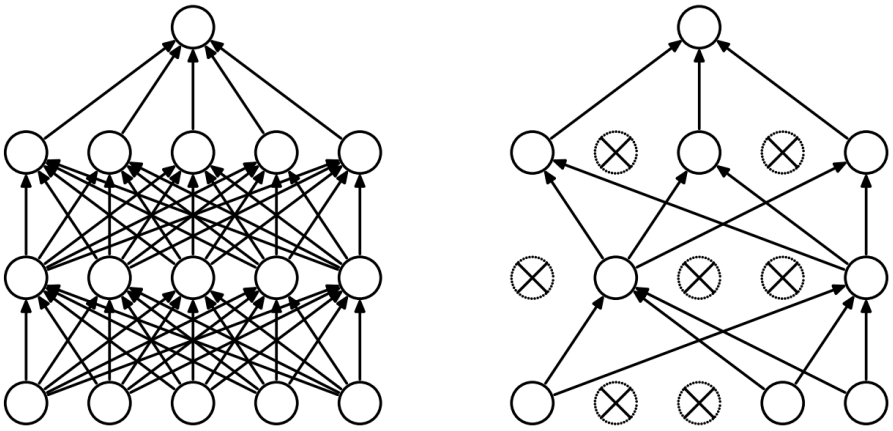
$$\gamma = \sum_{i}^{N} |w_i| \tag{2.27}$$

It constrains the weights to be more sparse with respect to the previous method, limiting the model to perform a feature selection step implicitly.

Dropout Another most recent technique to avoid overfitting when dealing with deep learning models is *dropout* [21], which forces the network to generalize by dropping some neurons and the corresponding weights during the training procedure. At each forward step, a certain probability of neurons is dropped pushing the algorithm to extract just the most essential features to solve the problem. In Figure 2.10 the visual explanation of the algorithm is provided.
At the testing time, all neurons are used, but the weights are reduced to a factor of p to take into consideration the missing information during training.

2.3.10 2D Convolutional Layers

The networks mentioned above are based on a single type of layer; the dense layer, which is a simple fully connected layer in which each input is connected to all the



(a) Standard Neural Net (b) After applying dropout.

Figure 2.10. Example of Dropout during training.

neurons of the next layer. On the other hand, based on the task to solve, different techniques and layers are available. The 2D Convolutional layer is a particular layer that works well when local patterns in the data are present (e.g., images). Instead of having a set of neurons, the Convolutional layer has a set of filters of arbitrary dimension ($width \times height \times input - channels$) that convolve the input to extract spatial features moving with a certain stride from left to right. At each set of features, the product between each element of the filter (kernel) and the input elements is computed, and the results are summed to obtain the value of the current convolution; the linear operation of each convolution is described in equation 2.28 where h represents the filter, y the generated feature map, and x the input. The spatial dimensions of the output feature map are equal to the number of steps made, plus one, accounting for the initial position of the kernel. The depth, instead, is equal to the number of different filters applied in every location. Another possibility in order to keep the dimension of the input over the layers is present. If padding is used, the image is enlarged with zero values in order to have the same number of features as output. In Figure 2.11 an example of Convolutional operation is displayed.

$$y[k,m] = \sum_j \sum_i x[i,j]h[k-i,m-j] \tag{2.28}$$

When dealing with images the spatial information is of crucial importance and the convolutional operation can effectively extract the necessary features to create a performing learning model. Object classification and detection are two of the most felt topics in the computer vision field and the need to extract pattern in the images is required when dealing with this kind of objectives. As an example, let's consider the aim of finding

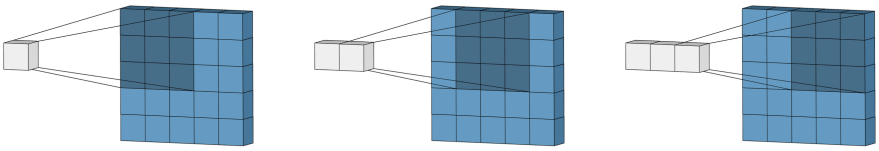


Figure 2.11. Example of Convolutional Operation.

the features to recognize a human face. To tackle the same an example is to create some hand-crafted features (filters) that emphasize the facial characteristics and tries to extract the nose, hears mouth ... features using a specific operation over the images (filters). On the other hand, convolutional layers apply a similar automatic technique. Randomly initialized filters are used to learn different features in the pictures trying to detect the values that best extract the required information to minimize the loss function. Filters are randomly initialized to learn different patterns and reduce the error faster; while minimizing the error the architecture intrinsically learns the necessary and most important features (filters) to solve the task. In Figure 2.12 an example of a filter (hand crafted) used to extract the edges in an image is provided.

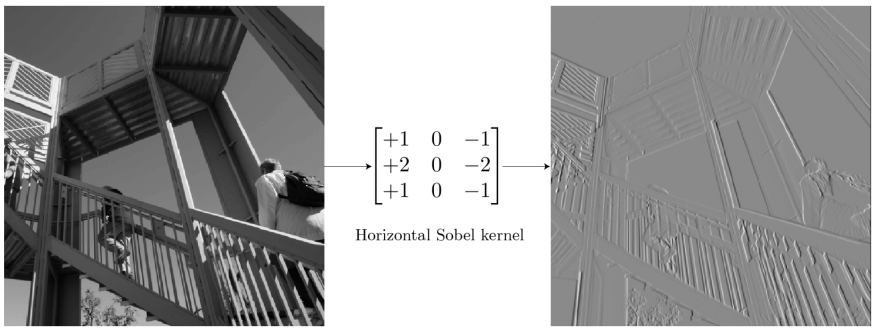


Figure 2.12. Example of filter used to detect edges in images.

2.3.11 Pooling Layers

The pooling layer is another layer used when dealing with spatial information to reduce the dimension of the data after convolving the input and maximizing the data gathered using the convolution. The pooling operation aggregates the spatial data applying a precise procedure and moving into the data with a particular stride and kernel size. In Figure 2.13 an example of Max Pooling Layer is presented.

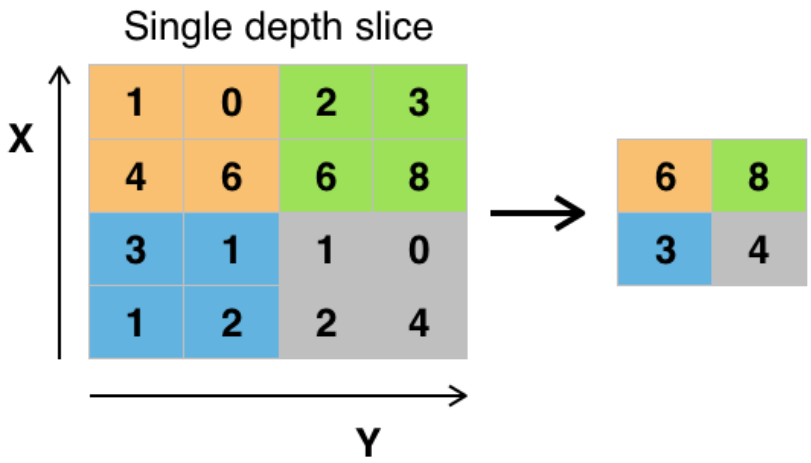


Figure 2.13. Example of Max Pooling Operation.

2.3.12 Convolutional Neural Networks

Both Convolutional and Pooling layers have been widely used when dealing with image processing. As mentioned-above the opportunity to automatically extract meaningful features from the images without involving the human assistance created a more flexible solution to the problem of classifying and detecting objects in images. Nowadays different deep networks are available to face accurately the problem of classifying and detecting objects. The employment of many layers has been the most significant discovery of the past years; the invention of AlexNet [22] in 2012 can be considered the first achievement. A more substantial number of layers helps in the definition of more hierarchical features; starting from simple filters going deeply to low-level features. In Fugure 2.14 an example of features extracted using a Convolutional Neural network is provided.

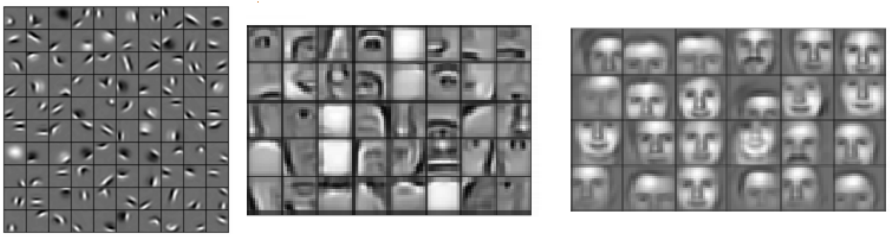


Figure 2.14. From left (low) to right (high) an example of features extracted using a Convolutional Neural Networks.

Designing a network with many layers rises a model that can better learn and generalize over the faced task; on the other hand, some problems concerning propagating the information can be present due to a large number of forwarding passes. For this

reason, some more recent networks as ResNet [23] overcame this problem by introducing the concept of the residual block, in which each layer is forwarded to the next block of convolutions with a residual sum. The introduction of this technique led to the opportunity to create deeper networks reaching better performances concerning accuracy. In Figure 2.15 an example of ResNet architecture from 2015 is provided.

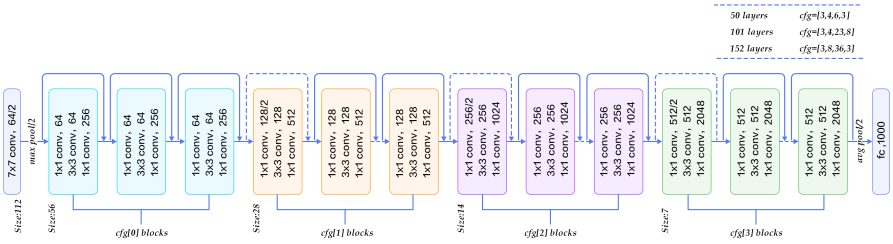


Figure 2.15. The ResNet architecture.

2.3.13 1D Convolutional Layers

Another type of convolution is the 1D Convolutional layer, which follows the same pattern as the 2D one with the difference that the input is a 2D matrix and the applied filters are in a single dimension (the difference concerning formula is described in Equation 2.29). The convolutional operation follows the same principle as in the previously explained case, but usually, the applications are different. While for images 2D convolutions represent the best approach to extract features when dealing with text and audio the input is represented by a 2D matrix and to extract spatial information 1D convolutions represent the most suitable technique; e.g., in the case of text each word is described by a certain number of features called embeddings; for this reason, the input is drawn as a 2D matrix. Convolving a set of words features can solve the defined task effectively. In Figure 2.16 an example of 1D convolution layer applied on a text sequence.

$$y[k] = \sum_i x[i]h[k-i] \tag{2.29}$$

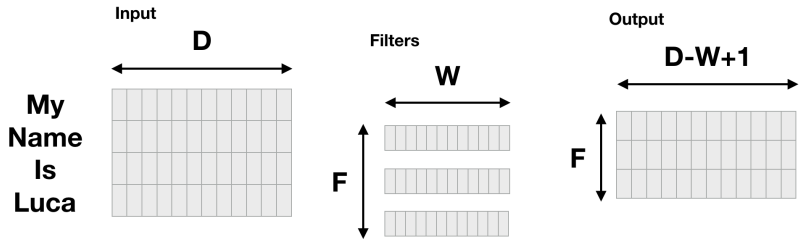


Figure 2.16. Example 1D convolutions; where D is the embedding dimension, W the filter width, and F the number of filters.

2.3.14 RNN

As seen so far, feed-forward neural networks and convolutional neural networks represent two subsets of possible deep learning models. Based on the application a network can outperform the other, but still, the temporal information has not been taken into account while describing the possible uses of both networks. In fact, another type of network has been invented to extract temporal dependencies: RNN. The output of each hidden layer is passed as input with the next sample instead of being lost in the forward pass. The basic intuition behind RNN outperformed state of the art algorithms when dealing with sequences (e.g., time series, machine translation). The need to extract temporal dependencies in these kinds of applications is required to have high performing models. In Figure 2.17 an unrolled recurrent neural network is displayed, where a single neuron behaviour is shown through the time, and the activated output is forwarded to the same neuron with the input at time $t + 1$.

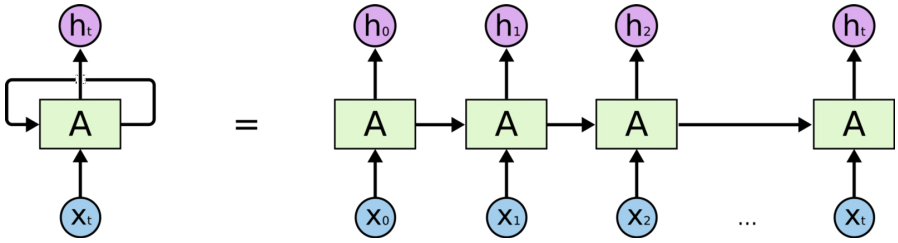


Figure 2.17. Example of RNN unrolled.

At the time of writing RNN can be used to learn long-term dependencies applying ReLU activation functions and initializing the weights in a proper way [24]. On the other hand, when RNN has been invented, the *gradient vanishing problem* was evident when dealing with long-term dependencies due to the fact that Sigmoid and Tanh were mainly used to transform the input (due to the lack of Relu presence in 1997), and *backpropagation* algorithm was strictly dependent on the input sequence length. As explained in Subsection 2.3.5, with this kind of settings, the propagation of the gradient through all the layers of the network resulted in a low gradient value when reaching the first layers of the network. In this case, the time dependency results in a similar format as having several hidden layers. For this reason, to solve the problem mentioned above a new temporal network has been invented to deal with the *vanishing of the gradient problem* and deal with long-term dependencies. The new network, known as LSTM [25], represents a model with cells instead of neurons capable of keeping the information during the sequence using specified gates which employs internal activation functions and weights. This particular configuration allows LSTM to maintain the error trough long periods without occurring in the *gradient vanishing problem*. In Figure 2.18 an illustration of a LSTM cell is provided.

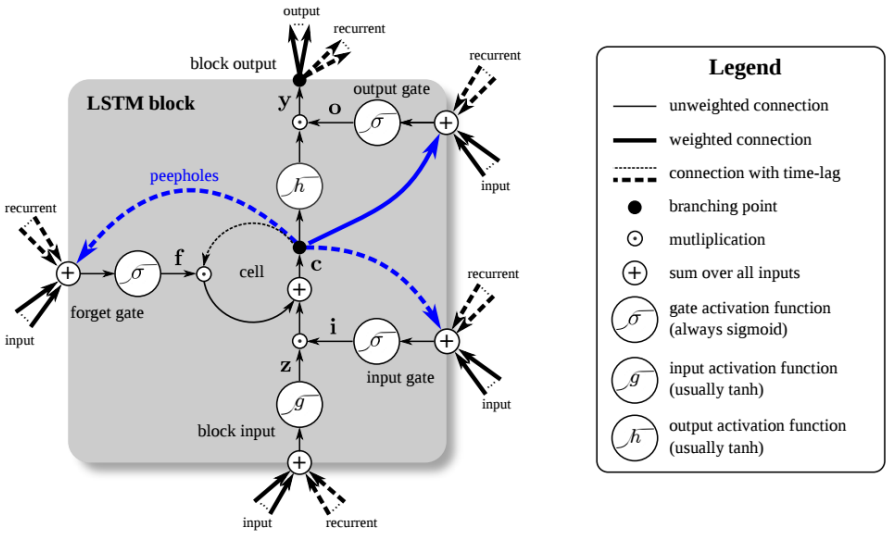


Figure 2.18. Example of LSTM cell.

Even if the problem of *vanishing gradient problem* has been solved using ReLU activation functions, still, LSTM cells are preferred by the research industry when dealing with temporal data due to the capabilities of deciding which information is essential to keep/forget thanks to the presence of the unit gates. Following, a brief explanation of the LSTM cell concerning gates and states.

Forget Gate Here the equation of the first gate of the cell, where x_t represents the input and h_{t-1} the cell output at the previous time step.

$$f_t = \sigma(W_f x_t + W_f h_{t-1} + b_f) \tag{2.30}$$

The forget gate has the objective to decide which information should be kept and which instead should be updated taking care of the new input using Sigmoid (σ) activation function.

Internal State The second step is to decide which information we are going to store in the cell state. The procedure is composed of two phases: first, the input gate layer determines which value will be updated, second, a *tanh* activation function creates a new candidate that could be added to the state.

$$i_t = \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i) \tag{2.31}$$

$$\tilde{C}_t = tanh(W_C x_t + W_C h_{t-1} + b_C) \tag{2.32}$$

The two outputs $(i_t, \tilde{C}_t)$ are then combined and the new state cell updated accordingly to the formula:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.33}$$

Output Gate The final step represents the decision of what the cell is going to output. The output is based on the cell state C_t filtered using the processed input o_t which has the objective to decide which information of the cell it is essential to output.

$$o_t = \sigma(W_o x_t + W_o h_{t-1} + b_o) \tag{2.34}$$

$$i_t = o_t * tanh(c_t) \tag{2.35}$$

Here the basic LSTM has been explained, but to overcome some limitations, new architectures have been designed during the last few years due to speed and convergence problems of LSTM under certain configurations.

2.3.15 Machine Learning Workflow

Each machine learning problem must be faced with a specific pattern. In this Subsection, the explanation of the general workflow of a machine learning project will take place. This prerequisite is essential to understand the taken decisions during the work and follow the used paradigm to obtain a performing result.

1. Problem Definition and Data Collection: the need to collect the right data and define the problem which a data-driven approach should solve is required in the first step.

2. Metric Definition: a formula is represented by the metric used to evaluate the model based on the problem that it has to be solved (e.g., in case of sales predictions MAE can be used to estimate the error of the model).

3. Evaluation Method: after defining the metric to use, the necessity to validate the model is required. Cross-validation is usually used for this task, but when not enough resources are available, and a deep model has been designed, hold-out validation is used.

4. Baseline Definition: a basic method or a state of the art approach should be defined as the baseline to compare and validate the results in case of a positive outcome.

5. Network Design and Hypothesis: based on the problem and metric defined a first design step is necessary to start the implementation. When a model has created

a hypothesis over the solution is intrinsically made (e.g., taking the same previous example, in case of sales prediction, exploiting the temporal information could increase the performances; a recurrent layer should learn the sequence pattern and outperform a simple linear regression).

6. Model that Overfits: after the definition of the architecture we need to test the learning capabilities of our model on the train set. We have to find the hyperparameters that make the convergence of the algorithm possible. Increasing the model complexity (features space), adding layers, decreasing the data, they are all steps that can help with this intent.

7. Regularizing the Model: when low train error has been reached the final step is to increase the generalization capabilities of the model enlarging the number of samples, decreasing the model complexity or applying regularization techniques (e.g., weight decay, dropout).

The list represents the step to follow when dealing with a machine learning project. Each of the steps is crucial to find a final performing model.

2.3.16 Model Selection and Hyperparameters Tuning

When dealing with machine learning the necessity to design the most suitable model to tackle the defined problem is required. Each machine learning algorithm has a certain number of hyperparameters which must be defined a priori by the user. These hyperparameters are not learned from the model itself, and they must be carefully decided during the implementation. In the case of machine learning the amount of these hyperparameters is not a relevant number; while when a deep neural network is used many are the possible combinations. The number of layers, number of neurons, optimizer, learning rate, dropout and activation function are all parameters that must be defined during the network design. Good hyperparameters could lead the model to converge faster and having better performances. At the same time trying all the possible combinations of parameters is not scalable concerning time, and a faster solution is often required to decide these values. Moreover, an evaluation scheme is required to prove the reliability of those parameters. Most of the time Cross-validation is used for this task since validating the performances over a set of folds results in a more robust value. Computing mean and variance of the obtained scores can give an idea about the general performances of that particular model with a certain set of hyperparameters.

To investigate and find the best parameters some methods are available:

1. Grid Search: define a range for each parameter and try all the possible combinations.

2. Random Search: a similar range of parameters is defined as in the Grid Search method, but here a random selection is performed instead of trying them all.

3. Bayesian Optimization: a probabilistic model is built between the hyperparameters and the validation fold. The model is updated based on the obtained results at each interaction to find the optimal parameters configuration. It uses a trade-off between exploration and exploitation to avoid local optima.

4. Evolutionary Strategy: using genetic algorithms optimization technique tries to find the best hyperparameters space minimizing the defined fitness function related to the obtained performances during validation.

All the search methods listed above try to find the best parameters that minimize the validation error on a specific set of folds, but even if more advanced approaches as Bayesian optimization and Evolutionary Strategy should converge faster to the optimal solution, the need to iterate and validate on many possible combinations is required, resulting in a super computational expensive procedure in terms of time. Due to hardware limitation and network dimension is not always possible to perform one of these methods to find the required parameters.

Moreover, none of those methods guarantees the success of the searching procedure. Cross-validation is a robust metric to evaluate the performances of a model and therefore finding the best hyperparameters, but it is not guaranteed that the same settings could perform equally well on the test set. When searching for the most suitable hyperparameters cross-validating on the train data can lead to a decision of specific values, while the best ones are not among the selected ones due to some differences in the data distribution between train and test or other specific characteristics that the data could have. Train and test may come from a different stream of data, and more generalization is not guaranteed when tuning with Cross-validation on the train set. For this reason, when dealing with substantial deep learning models, a different approach is used. As mentioned above in Subsection 2.3.15 the need to overfit is the first phase after the definition of the network architecture. Training a deep learning model is not a simple task [26], as mentioned above the number of hyperparameters and the complexity in the error surface increase the probability to get stuck in a local optimum. Without the right optimizer and learning rate value, the converge to the global optima is not guaranteed even with a deep network composed of many layers and neurons. For this reason, a ranking among the hyperparameters can be defined. Optimizer and learning rate represents the first two parameters that must be found to overfit when having a certain complexity in the model architecture (assuming a meaningful network design phase based on the task). After the definition of an optimizer, the need to adjust the learning rate is mandatory after each run ((100-500) epochs). Starting with a higher

learning rate and observing the behaviour of the train loss error, weights are saved, and the learning rate value is then adjusted manually decreasing it every-time a fixed error is obtained. Nowadays, adaptive learning rate optimizers are available to deal with this kind of task (e.g., Adam [18] is the most recent and used optimizer); on the other hand, a fixed starting point is present, and the definition of this value can profoundly affect the convergence. The process of manually change and iterate the learning rate saving the weights at each run is called "babysitting" the model. After the definition of the optimizer and the most suitable learning rate the need to find the other parameters is required. A first big step has been already taken toward the definition of the model hyperparameters; a search approach could be used to find the remaining parameters or going further in the "babysitting" can be still the best option to follow.

The "babysitting" approach is highly scalable concerning time compared to other methods as mentioned earlier and monitoring the results can help in the problem understanding as well as the definition of a more accurate architecture without the necessity to iterate over many configurations which in most cases are a lousy set of hyperparameters.

2.4 Related Work

Many authors have faced the task of predicting the 3D position of a soccer ball during the past years, but even if much effort has been put in the research field no optimal solution is still available. Due to the complexity of the task, all the available answers are biased toward the author's assumptions. Even if the possible solutions are a vast number, a pattern can be detected among these models. As mentioned in Section 1.1, the task of finding the 3D ball position can be split into few different steps:

1. Camera calibration: computing the projection matrix using some known points of the field.
2. Ball detection: determining the position of the ball within the image.
3. 3D position estimation: estimating the 3D coordinates of the ball in the field with the previously gathered information.

For this reason, the section goes through the explanation of related works that tried to solve the problem using a computer vision approach, moving on to the description of each separate step and how can be faced as a single different deep learning task.

2.4.1 Camera Calibration

Camera calibration is the procedure of finding the projection matrix 2.36 from the 3D to the 2D representation. The projection matrix P can be decomposed in the three central matrices: the rotation matrix R (3×3), the translation matrix T (3×1) and the camera matrix K (3×3). The results is a 3×4 matrix that represents the mapping between the 3D environment and the 2D representation. Rotation and translation represent the extrinsic camera parameters while the camera matrix corresponds to the intrinsic camera parameters (e.g., focal length, image dimension). With this data, the mapping from the 3D to the 2D environment can be estimated.

$$P = K[R|T] \tag{2.36}$$

Knowing some known points (both 2D and 3D), the computation of the mapping matrix can be solved using different methods. The projection matrix can actively help in the definition of a solution since the 2D position and the camera calibration constraint the problem to one dimension.

The calibration can be computed with different algorithms which use linear or not linear optimization. The most known is [27] that became the standard OpenCV [28] method for computing camera calibration having as input some observed points. Another most

recent method [29] uses Neural Networks to find the camera intrinsic and extrinsic parameters. A better camera calibration can produce better results, but a reconstruction limit will always lead to a certain amount of error in the prediction.

The importance of this step is explained in the Subsection 3.3.4 where the baseline method of this work is defined. A brief introduction was required for better understanding.

2.4.2 2D Object Detection

The task of detecting objects in images has been studied for a long time using different techniques and approaches. Since the birth of computer vision, this problem has been one of the most studied topics, with consistent use of various new methods. The growth of deep learning correlated with an increased computational capability enabled the first object detector to come out in 2013 [1]. The approach started a revolution concerning experiments and researchers toward this new direction. Among the different methods we can spot two streams: the methods focused on accuracy as RCNN [30], FasterRCNN [31], MaskRCNN [32], and the ones that try to decrease the complexity and the performances to have a faster solution; YOLO [33], SSD [34] and YOLO9000 [35] belong to this category.
The same paradigm is used in all the most recent solutions; a deep base network tries to extract the image features while the last layers are readapted and fine-tuned to compute the bounding boxes and the object's classification. In Figure 2.19 a sample in the ImageNet Dataset [36] is displayed. The introduction of a single network for both tasks [31], raised a better solution regarding computational requirements and results.

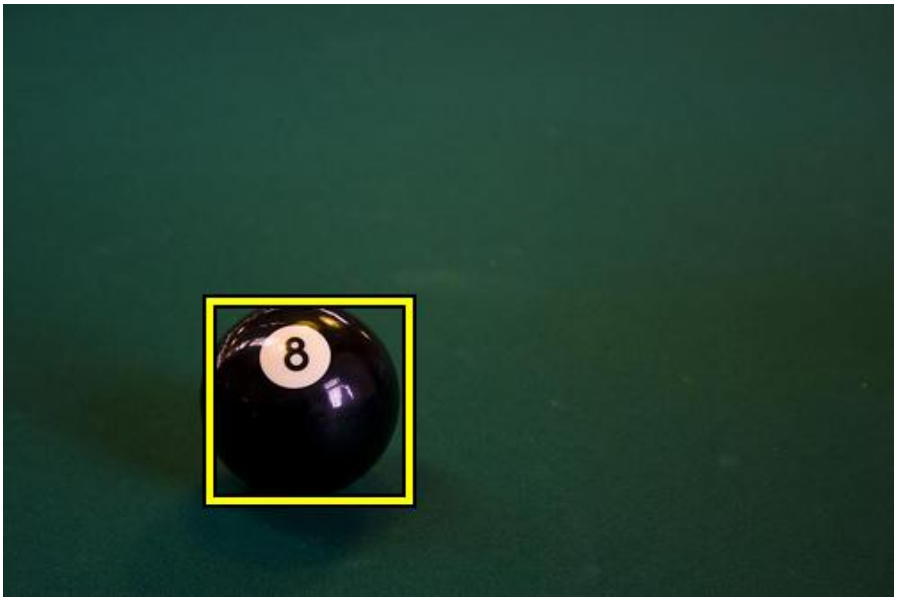


Figure 2.19. Example of sample in the ImageNet Dataset

The base networks are usually trained over large datasets for a considerable amount of time (in the order of weeks), resulting in impressive scores concerning generalization. For this reason, the same model weights are used in different domains giving a first baseline result concerning accuracy. On the other hand, when a specific solution has required, these performing models cannot be exploited.

2.4.3 3D Trajectory Reconstruction

Estimating the 3D coordinates of a ball during a (tennis, basketball, soccer) match has been faced with various approaches. Given the assumption that when the ball is laying on the ground, the perfect position can be computed just using the projection matrix and the 2D coordinates of the ball in the image, the authors concentrated on the problem of understanding the trajectory when the ball was not on the field. Most of them have exploited the physical constraints that the ball has to follow within a real environment.

The first approach related to soccer is [2], where the physical limitation on the laying plane of the ball, the trajectory equations and the height of the ball have been adopted for the prediction. The height of the ball is computed according to the assumption of having the height of the players. This work tries to overcome the limitations of [6], which exploits only the virtual plane constraint where the ball can lay.

From the point of view of ball trajectory estimation, [3] fitted the trajectory equations to

compute the ball coordinates. Similarly, [4] estimates the projection matrix using some meaningful points of the fields (6 no-coplanar points). Using the computed matrix for each sample (25 frames) the parabola parameters are computed $(x_0, y_0, z_0, v_{x0}, v_{y0}, v_{z0})$; the (x, y, z) position can then be determined using the t variable as the input of the equations 2.37.

$$\begin{cases} X = x_0 + V_x t \\ Y = y_0 + V_y t + g t^2/2 \\ Z = z_0 + V_z t \end{cases} \tag{2.37}$$

In [8], a solution is proposed similar to [4]. The objective was to decrease the problem of noise in the prediction. When computing the $(x_0, y_0, z_0, v_{x0}, v_{y0}, v_{z0})$ parameters each sample has equal weight; instead of being weighted on the time step of each frame.

Another innovative approach [5] tries to reconstruct the trajectory using a new geometrical projection. The use of shadows gives the possibility to obtain similar results without assuming the physical motion of the ball, that, most of the time, is not satisfied. The most recent approach related to soccer is [7], where an optimization approach has been used to find the best virtual plane where the ball lies. A comparison with the first plane-based method [2] has been done for motivating the decisions. Most of the papers cited in this subsection have been published before deep learning methods came out as the new state of the art algorithms in the object detection field. For this reason, the task of computing the 3D ball trajectory has been combined with the design of the object detector to find an ad hoc solution. A considerable advantage is now present compared to past years.

2.4.4 3D Position Estimation using Deep Learning

The problem of estimating the 3D coordinates of an object can be directly linked to the self-driving car industry. Deep learning successes moved the research from sensor-based approaches to computer vision image-based models. When dealing with autonomous cars, a network should be able to detect and correctly map each object in the images. For this reason, a massive investigation into the capability of these models for estimating the depth of an object from a single camera has been performed.

A model able to extract the depth information can perfectly map the object in the 3D environment with no need of exploiting the temporal dependencies. On the other hand, even if state of the art methods ([37], [38], [39] and [40]) seem accurate on big objects and static environments, the limitation on the small size of the ball makes the problem more difficult from a depth perspective. Moreover, a gradient-based dataset is

needed to estimate the depth-value of each pixel. RGBD images are usually needed to train the models as provided in Figure [41].

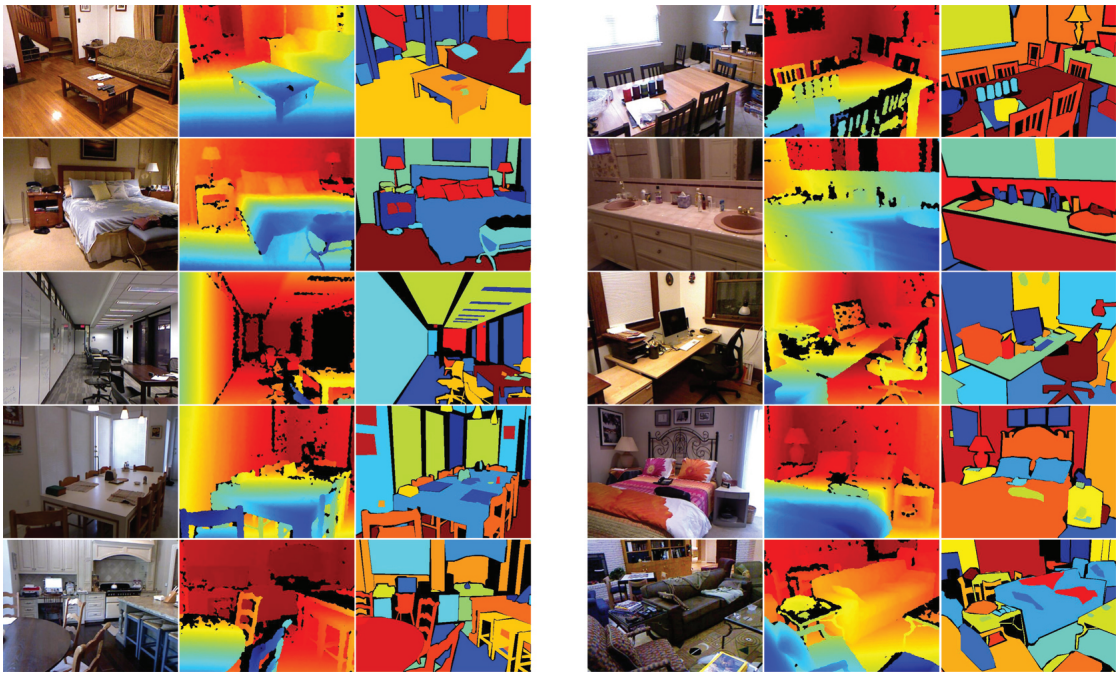


Figure 2.20. Example of images in the NYU Depth Dataset

The two above mentioned limitations, brought to the decision of giving up with the depth-pattern, even if a similar problem has been already solved using the combination of an object detector and a depth-estimator [42].

Chapter 3

Method

Despite the long research history, the estimation of the 3D coordinates of an object starting from a single fixed camera is still a hard problem. For this reason, an empirical approach should be followed to achieve a robust result. The employment of quantitative analysis was required to prove the effectiveness of the implemented method; on the other hand, the quality requirements are strictly related to the domain in which the pipeline is employed. In this chapter, a brief description of each performed step will be explained with the motivation behind the choices.

3.1 Empirical Approach

As mentioned above, the need to experiment and iterate over the obtained results was mandatory to achieve a suitable solution. The lack of related works and the opportunity to employ new advanced data-driven techniques brought to the decision of creating different models based on the intuitions gathered while analyzing the problem from a mathematical perspective. At the same time, domain-specific requirements are present when the 3D position of an object has to be estimated. A good result depends on the field in which the architecture is applied; the work addressed a single case study in which an iteration over the performances have been used to outperform existing methods and create a general solution for this specific task.

3.2 Splitting the Problem

Due to the complexity of the work, the need for splitting the problem into different sub-tasks was required. Breaking the problem into different functions can facilitate the implementation and the debugging. For this reason, the problem of estimating the 3D position of an object has been split into four different tasks:

1. 2D object detection: the localization of the object in the 2D image with the relative coordinates.
2. Flying - not flying classification: estimating if the object is flying or not flying. The reason for this task regards the fact that when the ball is on the ground the y coordinate is equal to zero and the 2D information of the ball is enough to reconstruct the 3D position.
3. 3D trajectory estimation: predicting the 3D position of the ball when it is flying.
4. Plugging the models together: using the predictions of the detector as input features of the other models to have a final 3D estimation.

Some design choices have been taken to speed up the implementation and the validation in case of an additional iteration. For each task, a brief description will motivate the project steps.

The problem of detecting an object in a 2D image can be considered solved. In the past few years, the deep learning community came out with performing methods able to accurately predict the object position within an image, as mentioned in Subsection 2.4.2. For this reason, the principal task of the work is not related to the 2D estimation but is more connected with the next issues.

The classification task is of crucial importance since when the ball is on the ground, perfect accuracy could be obtained just employing the 2D information. At the same time, the classification problem could be assumed as a sub-problem of the 3D trajectory reconstruction. If an algorithm is perfectly able to estimate the 3D position of an object, the flying - not flying classification is intrinsically solved. When the object is at $y = 0$, it means that the object is on the ground, else, it is flying.

These assumptions brought to the decision of facing the 3D trajectory extraction as the first principal problem to tackle during the work.

3.3 3D Trajectory Estimation

As mentioned above, the bottleneck consists in the estimation of the 3D position while the object is flying. As explained in Subsection 2.4.3, the topic has already been addressed in the computer vision community.
This work proposes a solution that employees most recent techniques in order to have a flexible and general implementation with superior accuracy and less domain specific assumptions.

The need for a working and performing algorithm led to the following steps:

1. First trajectories dataset creation: a dataset composed of trajectories has been created using Unity.
2. Data preprocessing: the raw data is processed to feed the models.
3. Metric and loss definition: a decision regarding the metric and the loss function is needed.
4. Baseline definition: implementation of a basic algorithm.
5. Pushing the baseline: different camera calibration for a slight improvement.
6. First deep approach: implementation of a deep learning method to face the problem.
7. Noise necessity: noise added to the data to simulate a real environment.
8. Problem discussion: brief discussion regarding the poor results.
9. Second trajectories dataset creation: a new dataset has been created to iterate the solution.
10. A performing solution: first problem solution with excellent performances.
11. Robust to noise: the implemented model can deal with the added noise.
12. A faster performing solution: a new, faster solution has been achieved.

This Section will go through all the listed steps in order to achieve an appealing algorithm that could be employed to solve the issue and move on to the next phase.

3.3.1 First Trajectories Dataset Creation

Each machine learning algorithm needs a dataset where to learn patterns from. The objective of the problem leads to the ground truth decision and what to consider as *label*. In this case, the solution has to predict the 3D coordinates of the ball at each frame, which means that the label will correspond to a 2D matrix where at each timestamp the (x, y, z) position of the ball in the real world coordinates is present. Due to lack of existing data a Unity-based dataset has been built for this task. Unity [9] is a cross-platform game engine which is mainly used for 3D/2D game development. It has flexibility in terms of fast implementation but also methods for exporting the environment information during a game instance. The objective was to simulate a real case scenario in which the ball

moves with a specific pattern in the court. In the Figure 3.1 an example of extracted screenshot is provided. The court and the ball have been designed trying to keep the same dimensions as in case of a real-world scenario. The FIFA [43] rules have been applied to the size of the court. A range of values can be used in the designing of a soccer court; the maximum values, in this case, have been used to make the task more difficult, resulting in a $90m \times 120m$ field. The same pattern has been followed for the ball, $70cm$ in diameter has been used. One Unity unit represents one meter in the real world. The result is an environment that can correctly simulate a soccer stadium.

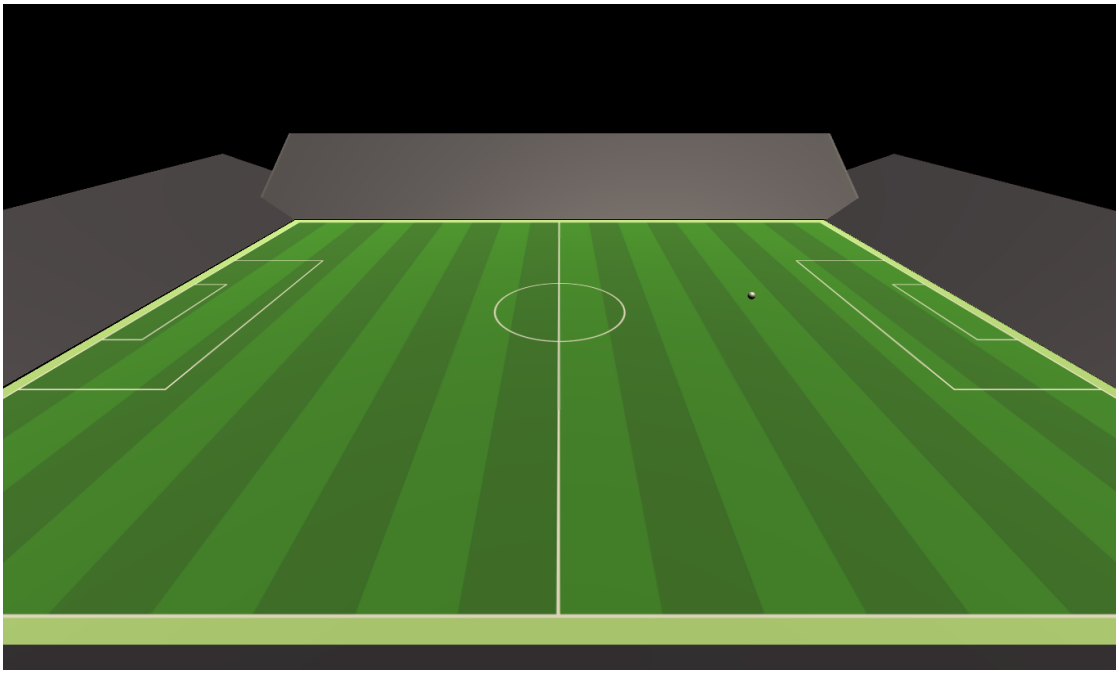


Figure 3.1. Example of an image extracted using Unity engine.

For the first explorative approaches, a structured dataset has been created. The x and y centre position of the ball in the 2D image have been exported respectively with the identifiers *relative x* and *relative y*. In Table 3.1 the dataset columns are provided.

Features		Labels		
relative x	*relative y*	x	y	z

Table 3.1. The first structured dataset columns used for training.

Unity gives the possibility to extract all the information capturing the frames and the coordinates at a specific predefined sample rate. At each frame features and labels were obtained to create enough comprehensive dataset. For this task 1224 random trajectories have been generated. Each trajectory with different applied forces in the three directions. For the model, to work in a real-time scenario, 25 Frames Per Second (FPS) have been exported. The result was a dataset composed of 1224 trajectories with an average of 379 frames each. In Figure 3.2 an example of a random trajectory is provided.

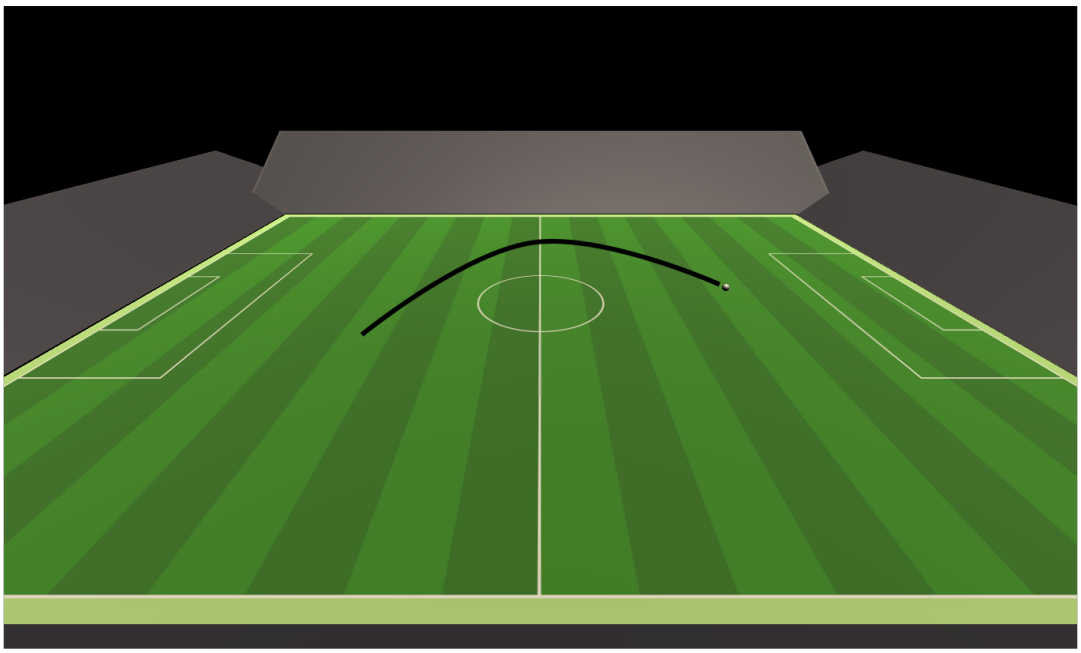


Figure 3.2. Example of a a random trajectory generated using Unity.

3.3.2 Data Preprocessing

Due to the fact that the dataset has been built specifically with certain characteristics a detailed data exploration was not needed. On the other hand, a meaningful preprocessing

was necessary to feed the data into a parametric model keeping a certain consistency to measure the algorithm performances. To have the same data distribution for the train and the test set, under the assumption of exploiting the temporal information for a more robust solution, a sequence generation step has been applied. For this reason, ten random sequences of 25 records have been sampled from every single trajectory (379 frames) with the respective label (x, y, z), corresponding to the position of the ball in the middle of the sequence (12th frame). This data generation leads to a dataset composed of 10*1224 samples that can be used to train and test. Due to some inconsistency in the Unity exporting procedure a data cleaning step has been followed, resulting in 9550 final samples. In Figure 3.3 an illustration of the sampling procedure for the data generation step is provided. The 9550 samples were then split randomly into three different sets: the training set, validation set and test set. When dealing with deep learning algorithms, instead of splitting the data into different percentages a numerical value is assigned in order to demonstrate the effectiveness of the model. In such a case, 200 samples have been used to test and 100 to validate.

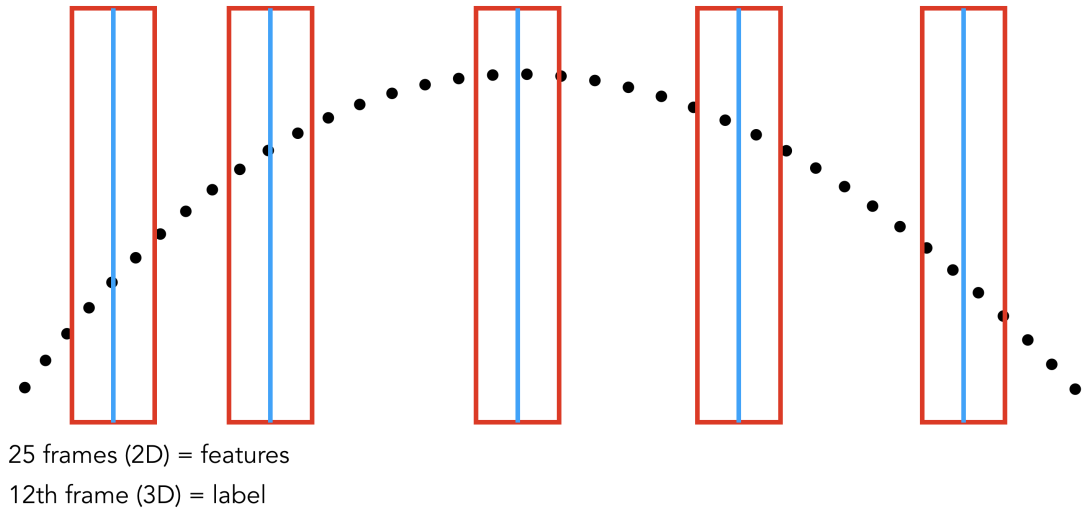


Figure 3.3. An illustration of the data generation step

Another important thing to mention regards the court system of coordinates. The Unity centre is reflected in Figure 3.4, which is not located in an ideal point of the court. Having both positive and negative values along the (x, z) coordinates doesn't help the user in the debugging process. For this reason, the system of coordinates has been moved to the bottom-left corner of the court summing each value to the max value possible in the (x, z) directions. In Figure 3.5 the new origin is illustrated.

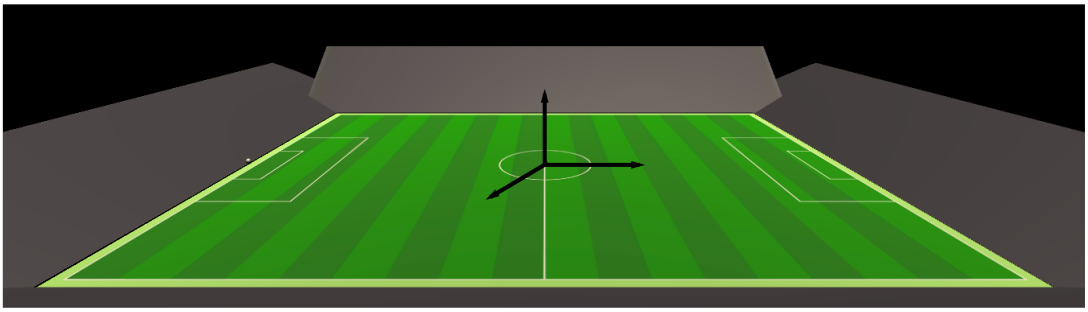


Figure 3.4. The coordinates system origin

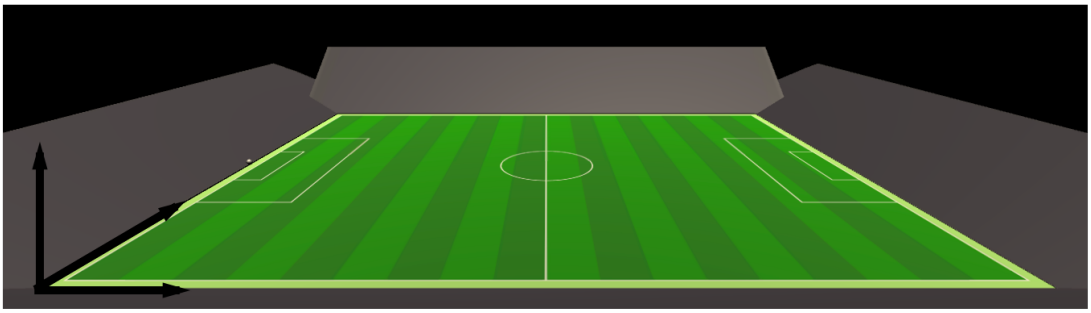


Figure 3.5. The new origin of the coordinates system

At last, to facilitate the learning procedure, a features standardization using the simple Scikit-learn [44] Standard Scaler [45] has been applied before accessing the parametric models.

3.3.3 Error and Loss Definition

When dealing with machine learning algorithms, the definition of an error metric is needed to compare different methods and models. In this case, MAE has been chosen due to the closeness to reality in terms of distance between the prediction and the real value. It represents the summed error in the three directions, which can be perceived as

the distance in meters between the ball and the prediction towards the sum of the three coordinates.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{3.1}$$

Similarly, for the training process, Root Mean Squared Error (RMSE) has been used as loss function. A machine learning good practice is to avoid to use the same formula for both error and loss, in order to have a more general idea of the model improvements. The same loss and error have been used in all the models when dealing with continuous values. The reason for this preference regards the debugging facility. Both RMSE and MAE can supply a meaningful idea about the error in the 3D coordinates.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2} \tag{3.2}$$

3.3.4 Baseline Definition

The starting point of each artificial intelligence project is to define a baseline that the work should outperform using a new solution. As mentioned in Subsection 2.4.3, the problem of estimating the 3D trajectory of a ball has already been faced by several authors. For this reason, a right baseline is to employ one of these methods and to compare the results using the defined error. Starting from the assumption that both [2], [3], [4] and [5] should have the same range of performances due to a consistent similarity, a decision was lead by the easiness of the implementation; as a matter of fact [4] has been chosen as basic solution. The authors adopt the parabola model equations with the camera calibration parameters to reconstruct the 3D trajectory of the ball. The technique is not parametric, which means that there is not an explicit training step. However, the necessity of computing the camera parameters is required. The camera calibration is calculated using some known points in the 3D and 2D environment. In [4], the line intersections are gathered using a line pixel detector method [46]. After this step, the camera matrix is computed by solving a linear system [47]. The same pattern of the paper has been followed. Some relevant points have been obtained using Unity, and the linear system applied to have the camera parameters. In Figure 3.6, the extracted points are shown.

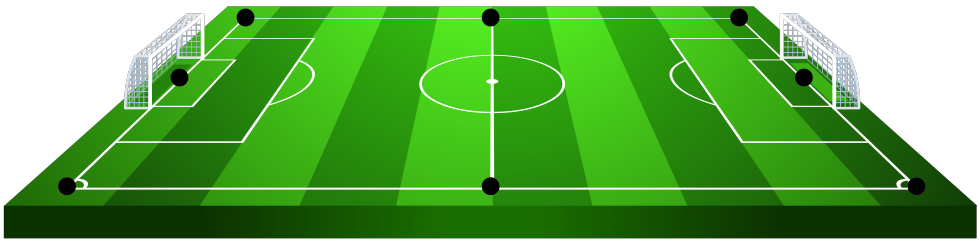


Figure 3.6. The co-planar points extracted with Unity for the Camera Calibration procedure.

With these 12 features (camera parameters), the model is now able to predict the 3D coordinates. For each sample (25 frames) the parabola parameters are computed $(x_0, y_0, z_0, v_{x0}, v_{y0}, v_{z0})$; the (x, y, z) position can then be determined using the t variable as the input of the equations 3.3.

$$\begin{cases} X = x_0 + V_x t \\ Y = y_0 + V_y t + g t^2/2 \\ Z = z_0 + V_z t \end{cases} \tag{3.3}$$

3.3.5 Pushing the Baseline

In order to improve the baseline, few methods have been investigated. The most obvious and straightforward regards the number of samples considered in the computation of the trajectory equations parameters. Better accuracy can be reached increasing the frames. Improved estimation of the 3D position can be computed having more useful points for the calibration.

Another consideration can follow after better analyzing [4]; the authors, in the camera calibration step, use six no-coplanar points of the field. For a 2D to 3D mapping the need of having no-coplanar points is essential. For this reason, the crossbar corners have been extracted to achieve a better result. In Figure 3.7, the new obtained points.

3.3.6 First Deep Approach

After the definition of a first baseline, a new method has been designed using deep learning. To exploit the temporal information a deep recurrent neural network with LSTM cells [25] has been implemented. The reason behind this choice regards the impressive results that LSTM networks have obtained when dealing with temporal sequences, becoming state of the art in different applications where a sequence-based pattern must be found in the data. Compared to vanilla recurrent neural network [48], they do not suffer

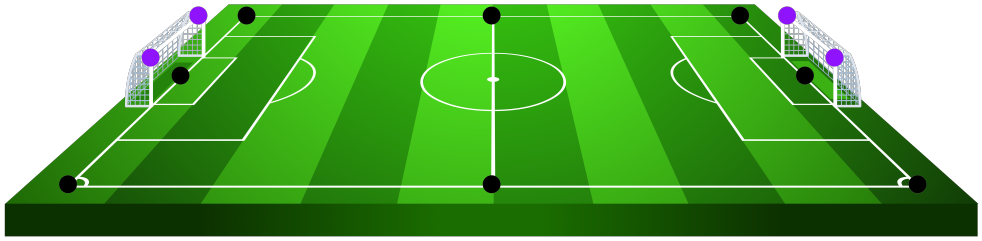


Figure 3.7. The no-coplanar points extracted with Unity for the Camera Calibration procedure.

from the vanishing/exploding of the gradient problem, taking care of learning longer sequences without having convergence issues.

The input features *(x, y)* position of the ball were given to the model in a 25 frames sequence. The output renders the (x, y, z) position of the ball in the court system of coordinates for the $12th$ frame among the sequence. In Figure 3.8 the adopted architecture is provided. The model resulted in poor performances even increasing the number of samples to consider in the sequence learning algorithm.

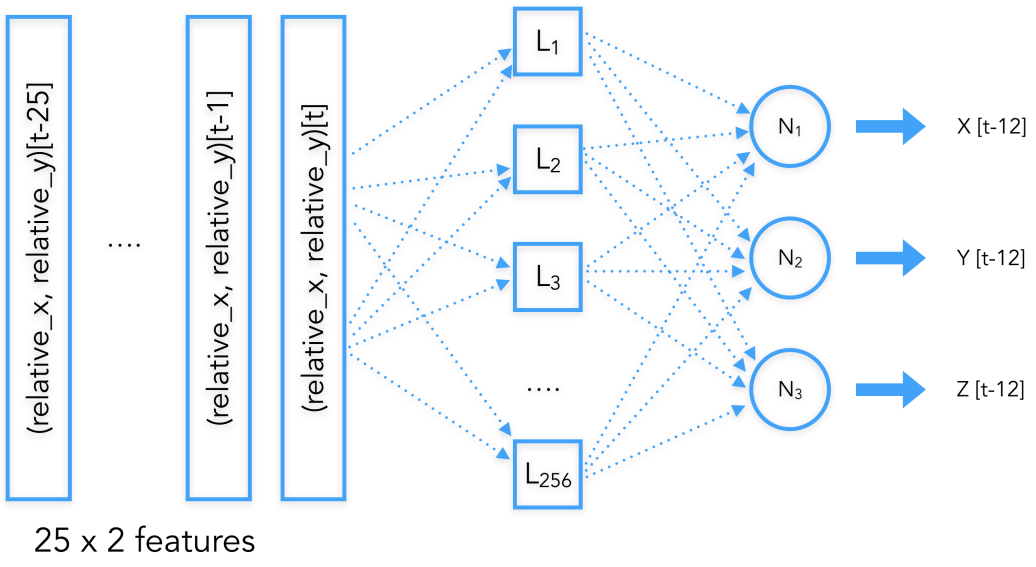


Figure 3.8. The first recurrent architecture used for training.

The model was composed of a single hidden LSTM layer with 256 neurons (cells) and a three neurons fully connected layer for the final prediction. No activation function has been applied to the activated output due to the already existing non-linearity into the LSTM cells. As optimizer Adam [18] with the default parameters has been used. Similar results were obtained using different optimizers or learning rate values. A hyperparameters search have not been applied due to the poor scores got from best practices. A problem in the architecture/input features was evident since the beginning.

The primary motivation behind this poor score concerns the fact that the 2D information creates ambiguity in the 3D reconstruction. It represents an undetermined problem where we have four variables and three equations. The missing information is the distance between the object and the camera, which is the feature required to reconstruct the 3D position correctly starting from the camera viewpoint; see Appendix **??** for more details. Even exploiting the temporal dependency, an accurate solution is not possible to obtain. For this reason, a design iteration was needed in order to find a more suitable solution and create a model capable of converging.

3.3.7 Noise

In this first experiments, the features came from Unity extraction methods. As mentioned in the previous subsections, *relative x* and *relative y* are the ball centre position with respect to the Unity camera. In a real case scenario, the object detector has probably a particular error in the prediction of the ball position; for this reason, a small amount of noise has been added to the *relative x*, *relative y* features to compare the results in a real simulation. The added noise was uniformly distributed with mean 0.0 and 3.0 of standard deviation, which means that the object detector prediction average error varies in a range of (-3, 3) pixels.

3.3.8 Problem Discussion

After investigating and experimenting with these models, an architecture reflection has been held. As mentioned in the previous subsections, the reconstruction from the 2D features is not enough to generate a consistent result in the 3D system of coordinates. Due to the lack of an entire direction, the model cannot learn a good representation. The missing information regarding the distance between the object and the camera is the essential information required to reconstruct the 3D position. At this point, the decision of adding more learnable features was mandatory. Another information could be extracted having the 2D images: the relative size of the ball in the pictures, which can be considered a crucial feature to estimate the distance between the ball and the camera. Having the precise $(x, y, size)$ of the ball in the 2D representation, a complete

reconstruction of the 3D position can be derived. After this critical assumption, an iteration of the architecture and the dataset has been applied. The next subsections will present an in-depth explanation of the new solution.

3.3.9 Second Trajectories Dataset Creation

A new dataset has been created considering the new features space. Using the same format of the previous dataset described in Subsection 3.3.1, at each frame the features have been extracted. The 2D size of the ball has been computed considering the distance between the left and the right corner of the object in the 2D images. In Figure 3.9 how the size feature has been derived.

Figure 3.9. An example of extracted size feature.

No other differences compared to the method in Subsection 3.3.1 took place. In Table 3.2, the representation of the new structured dataset is displayed.

Features			Labels		
relative x	*relative y*	*size*	x	y	z

Table 3.2. The new structured dataset columns.

3.3.10 A Performing Solution

The same deep architecture defined in Subsection 3.3.6 has been used to learn the dataset pattern and estimate the 3D position of the ball at the $12th$ frame in the sequence. The only difference is the input features, which now have another relevant information: the

relative size of the ball in the 2D images. With this new information, the deep recurrent neural network can learn an almost perfect mapping between the input and the output. In Figure 3.10, the new architecture used for training is provided.

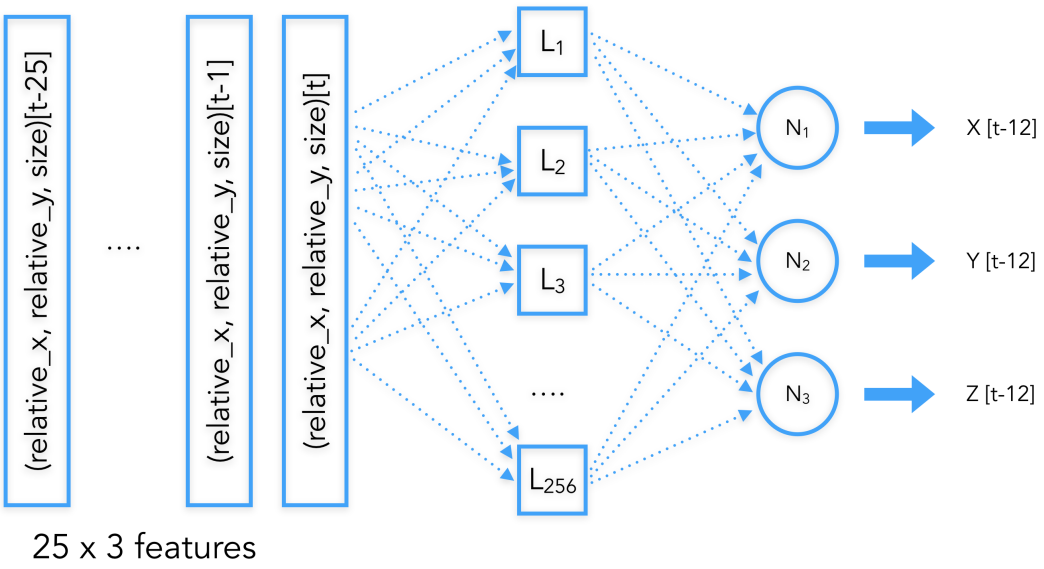


Figure 3.10. The new recurrent architecture used for training.

3.3.11 Robust to Noise

This last solution can be considered a performing approach to solve the problem, but in order to test the robustness of the algorithm, it is necessary to attach some noise also to the size of the ball feature. In this case, a normally distributed noise with 0 mean and 3 as standard deviation is not relevant for this specific feature. The prediction of the size is a much more difficult task considering the small dimension of the object in the entire image. For this reason, 25% of noise has been added to the feature. Which corresponds to a noise normally distributed with 0 mean and 0.25 as standard deviation. The final feature value has been clipped in case of negative values. A lower than zero prediction of the ball size is not possible in case of a well-trained model.

3.3.12 A Faster Performing Solution

After reaching a robust solution, a computation analysis has been taken into account. An adjustment to the previously explained architecture has been applied to speed up the training and testing procedure. In [49] the authors show a strong capability of *1*

Dimensional Convolutional Layers to extract the temporal information from sequences. An experiment with this type of layer has been held in order to compare the performances. Similar performances can be obtained using one single hidden layer with 25 of kernel-size and 256 filters. ReLU [50] activation function has been applied to the convolutional output filters. The kernel-size has been chosen considering the dimension of the sequence, to take into consideration the overall temporal dependency of the sequential samples. In each layer, Xavier initializer [51] has been adopted for weights initialization purpose. In terms of speed, this architecture is more efficient and can be used in a real-time scenario as explained in [52]. Moreover, another consistent advantage of convolutional layers compared to recurrent layers is the possibility to parallelize them over different machines. In Figure 3.11 the new model employed for training.

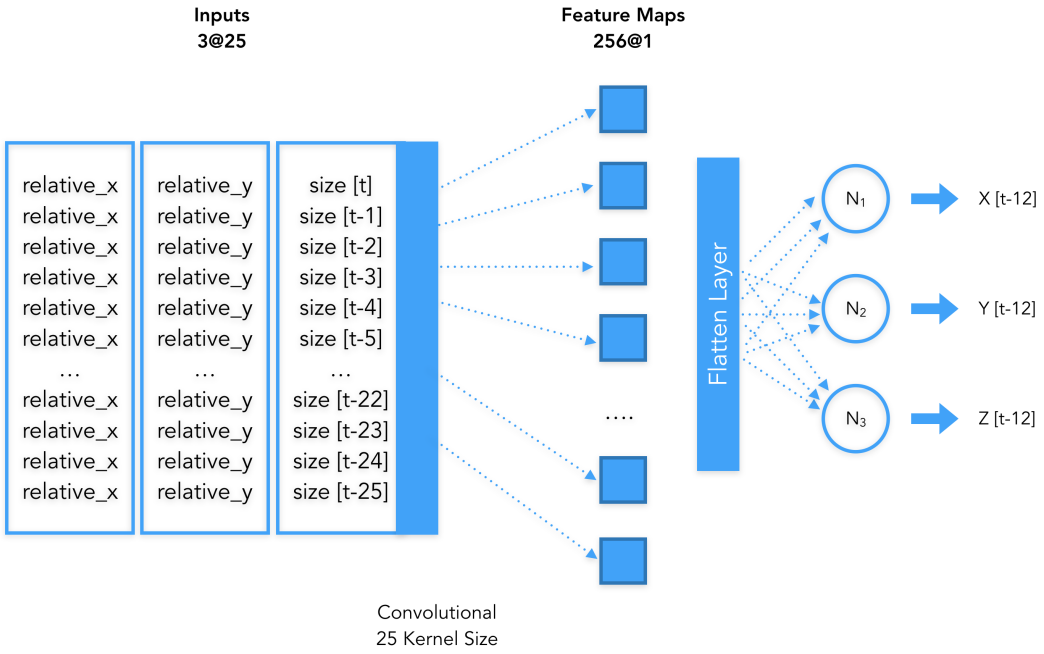


Figure 3.11. The 1D Convolution architecture.

3.4 Flying - Not Flying Classification

After solving the 3D trajectory task, the next step of identifying when the ball is on the ground or flying can take place. The definition of a suitable solution is highly correlated with the previous architecture and to reach high performing results the steps were:

1. Dataset Creation: a dataset for the new specific task has been built.
2. Data Preprocessing: the same pipeline has been applied as described in Subsection 3.3.2
3. Different task similar architecture: the same 3D trajectory network has been employed to handle a different problem with a different output layer.

In the next subsections, a brief explanation of each function will take place.

3.4.1 Dataset Creation

As in the previous task, the necessity of a suitable dataset is mandatory. In this case, the trajectories dataset defined in Subsection 3.3.1 is not general enough to test the classification performances of an algorithm, due to the fact that just in the first (when the ball takes off) and the last frame (when the ball lands) the object is on the ground. It represents a highly imbalanced dataset that cannot be used to learn this different problem. A new dataset has been created trying to simulate a real soccer match. For ~ 3 minutes the ball moves in the court doing trajectories and changing direction with different forces. The result is $\sim 3min * 25FPS = 3 * 60 * 25 = 4513$ samples as the dataset for the new task. The same features and labels have been extracted; with the difference that now the label is 0, when $y = 0$ and 1 when $y! = 0$.

3.4.2 Data Preprocessing

From the assumption of using the same network architecture used for the trajectory task, a similar data preprocessing, as explained in Subsection 3.3.2, followed the dataset creation.

3.4.3 Different Task similar Architecture

Before spending time in the implementation and design of a new algorithm, the same previously explained model, defined in Subsection 3.3.12, has been used to classify the binary ball position. Using *relative x, relative y* and the *size* of the ball in a 25 frames sequence the output is a $(0, 1)$ for the ball flying-not flying position of the $12th$ frame. With the same parameters and similar architecture, 99% of Accuracy 3.4 on the noisy test set has been obtained. Few edits have been applied to the model in order to fit the

task. As output layer, an one neuron fully connected layer with Sigmoid [53] activation function has been used for classification purposes. The decision of the loss function was based on the best practices when dealing with binary classification tasks. For this reason, Binary Cross Entropy (BCE) 3.5 has been chosen. In Figure 3.12 the architecture used for the classification task.

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3.4}$$

$$BCE = -\sum_{i=1}^{n} y_i' \log(y_i) \tag{3.5}$$

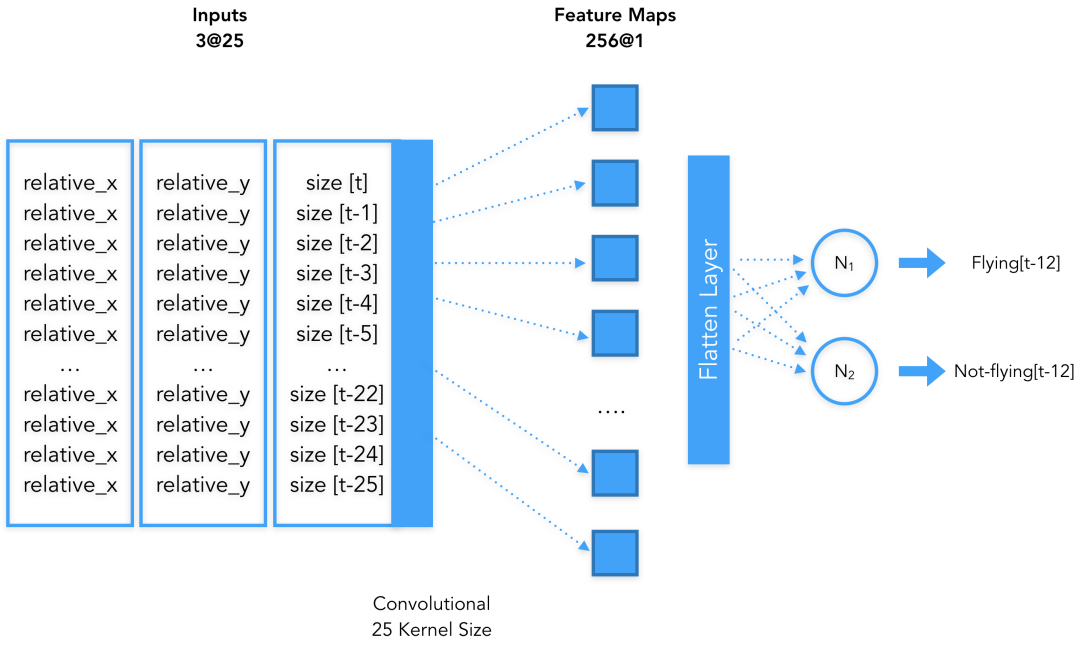


Figure 3.12. The 1D Convolution architecture for the classification task.

Having this score, a conclusion regarding the flying-not flying task can be done. As mentioned in Section 3.2 this problem can be considered a sub-problem of the estimation of the 3D position of an object; if $y = 0$ the ball is on the ground, else it is flying. After analyzing these results, a merge of the task has been performed. Using the same architecture as explained in Subsection 3.3.12, a test on the new more general dataset explained in Subsection 3.4.1 took place. The MAE is not significantly affected by the new dataset, and the score is similar to the only noisy trajectories case. Considering these results, a single model has been used to predict the 3D position of the ball starting

from the *relative x, relative y* and *size* of the ball features, from here after referenced as *temporal* net.

3.5 2D Object Detection

After validating the possibility to create a robust solution for the estimation of the 3D coordinates of an object, the last step was to develop a model that outputs the input features of the previously explained network. As mentioned in Subsection 2.4.2, much effort has been spent in the design of deep models able to detect objects in an image. The most famous models for real-time applications are SSD [34] and YOLO [33], which can be considered state of the art algorithms. Even if they are the most performing networks and use the most advanced deep learning techniques, a model based on these architectures do not fit the problem requirements. The reasons behind this assumption are listed below:

- Both SSD and YOLO try to detect all the possible objects in an image, while here the task is to identify the ball (single object). For this reason, there is no need for region proposal algorithms.
- The size of the ball is another feature that should be predicted by the model, which means re-adapting the implementation to another more modular solution.
- SSD and YOLO are not perfectly suited for the detection of small objects. One of the best practices when dealing with small objects is to avoid pooling layers, to not destroy the image information along the forward pass. Both architectures use a base network which employs several pooling layers.

All the assumptions and conclusions mentioned above have been investigated and validated during the network design step. It has been an iterative/learning approach in which the objective was to find a suitable solution combining best practices and empirical experiments. More details in Subsection 3.5.4.

Due to the lack of an existing solution, and relevant studies toward this direction, the necessity to create a new architecture was needed. Some steps have been taken to reach an ad hoc model:

1. Dataset: the same dataset generated for the flying-not flying task has been used.
2. Data preprocessing: the images are preprocessed before feeding the model.
3. Network design: the process of finding a suitable model for the task.

4. High-resolution images: the rise of a computational problem when dealing with high-resolution images.
5. Debugging convolutions: the activated images have been debugged for better design choices.
6. Definition of a good result: a metric has been defined, to stop the iteration and further improvements of the algorithm.
7. Hyperparameters tuning: best parameters have been searched using a "babysitting" approach.

A step by step explanation will take place in the next subsections.

3.5.1 Dataset

The dataset used for the object detection task is the same as the one built for the flying-not flying classification explained in Subsection 3.4.1. The difference is about what to consider features and labels. In this specific case, the images will be the input features and the *relative x, relative y, size*, the labels. The dataset is composed of 4513 examples that can be used to train and test the model. When dealing with deep models for object detection, the need of having a significant amount of data to train is often a limitation, while, here, in lack of samples, a generation step can always be performed using Unity. The last assumption led the entire tuning process.

3.5.2 Data Preprocessing

The input images extracted from Unity have all the same dimensions: RGB pictures with width 1360 and height 814. A good practice when dealing with high-resolution images is to apply a resize step in order to speed up the training procedure. However, in this specific case, decreasing the image dimension can delete relevant information that can be exploited by the detection algorithm. For this reason, only a rescale (/255) step has been applied to all the images, to have each pixel value between 0 and 1. This simple procedure helps deep neural network to converge faster and avoid problems in terms of gradient exploding in the first training epochs. As well as for the input, a good practice when dealing with object detection algorithms and big images is to rescale the labels $(x, y, size)$ to help the model in the convergence phase. For the *relative x* and *relative y*, a rescale method has been applied using the maximum possible values (1360, 814 respectively). While for the ball size a 2nd root operation was more appropriated as suggested in [54] where the objective was to estimate the size of an object to have a reliable object localization algorithm in the region proposal step. In [54] the authors created a network using VGG [55] base model with a Kernel Ridge Regressor [56] on top

of it. The observed approach can be recognized as a validation step toward the feasibility of the solution and the opportunity to obtain a robust result.
After the rescaling step a train, validation and test split was necessary for the algorithm learning procedure. As mentioned in Subsection 3.3.2, when dealing with deep learning algorithms a different separation is used compared to machine learning methods. Instead of splitting the data over a certain percentage, it is better to define a number of samples that can be viewed as enough to test and validate the model. In this case, 100 samples to test and 50 to validate were enough to measure the model performances.

3.5.3 High-resolution Images

A particular aspect that should be pointed out regards the images dimension. High-resolution images are difficult to train due to two main reasons:

1. Number of parameters: if the image dimension is not decreased over the forward pass when reaching the dense layer, the number of parameters is too large, resulting in super-heavy training.

2. Memory requirements: Tensorflow [57] saves all the activated images over the layers in order to faster compute a training epoch. At the same time, this results in a costly memory requirement. Due to a RAM limitation of the used GPU, a lower batch size has been used for training, making the gradient update less precise over each epoch. More information about the used hardware in Subsection 3.7.2.

Best practices have been exploited to solve the hardware limitations, but not all of them were suitable for the problem definition. E.g., the need for a lower image dimension was in contrast with the small object size, and the risk of destroying useful information if a re-size step would have been performed.

3.5.4 Network Design

Before reaching a performing solution, many attempts have been performed in terms of network design. The following decisions were taken to obtain the perfect model.

1. The reasoning of exploiting pre-trained SSD [34] from Tensorflow was the initial penetration into the problem, but after a better understanding of the algorithm, the probable difficulty in the integration of another branch (size prediction) would have created unnecessary complexity in the implementation. The decision of discarding the first thoughts was almost immediate.

2. Following SSD pattern, a model with VGG base network and 512 neurons fully connected layer on top of it, has been implemented. The poor results led to the

conclusion that the realized model was not able to fulfil the needs of the task. Even if the best weights were used to face the problem, the network was not able to converge properly.

The necessity of adjusting the model architecture was required. After a profound analysis of SSD network, an important discovery took place. SSD implementation suffers from the detection of small objects due to the presence of pooling layers in the base network.

3. After the first two attempts, the call for starting from scratch was mandatory. Following VGG implementation, a first deep model has been created using seven convolutional layers and one fully connected layer (half the number of layers in VGG16). In each layer *stride=2* has been used to avoid the waste of information, while at the same time decreasing over layers the dimension of the image. A $3x3$ kernel has been applied to convolve the images as in VGG Net. A *"VALID"* padding option, instead of *"SAME"* was used, to push further the reduction of the image dimension due to the fact that the border information was not essential. After each hidden layer a ReLU [50] activation function has been applied. Following the seven convolutional layers, the output has been flattened and passed to a 512 neurons dense layer; connected to the output layer composed of 3 neurons, with a linear activation function, for the final prediction of the $(x, y, size)$ features.

3.5.5 Debugging Convolutions

After a first investigation regarding the model performances with different parameters, an intuition regarding the number of convolutional layers took place. In this specific case, the dimension of the object constraints the number of layers that the model can use. Thinking about the architecture and the small size of the ball, after a certain amount of layers the ball will not be in the activated images anymore due to the linear dimensionality reduction performed over the forward pass. For this reason, a debugging step to see the output after each convolution has been performed. After the 4th convolutional layer, the ball was not present in any of the filtered images. In Figure 3.13 an example of an input image and in Figure 3.14 an example of the same image after the first convolutional layer are provided.

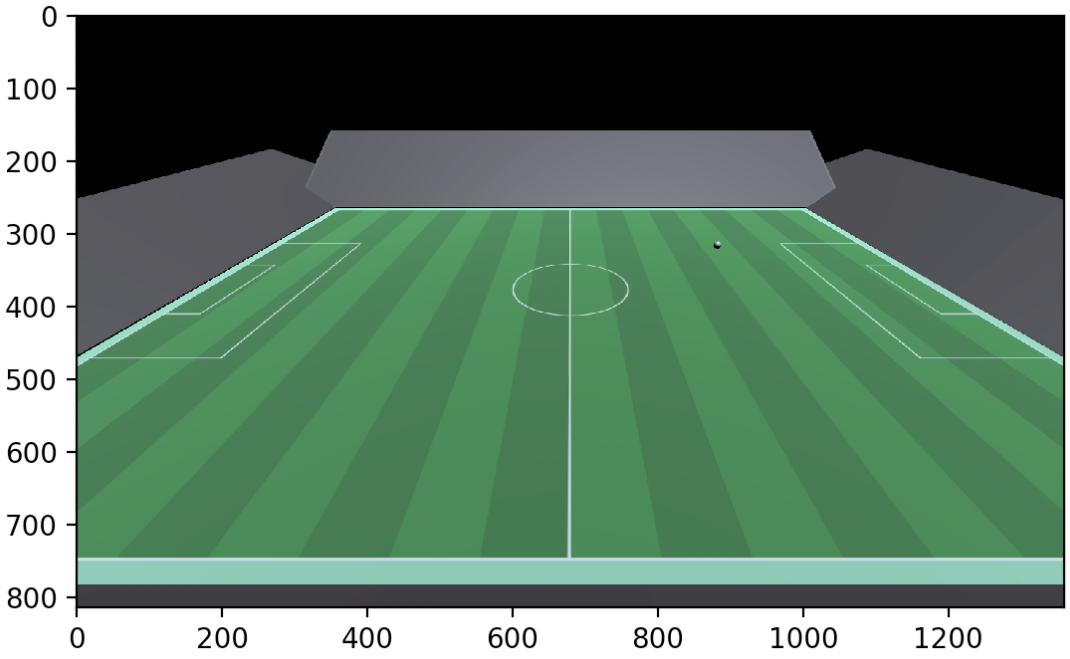


Figure 3.13. Example of input image (814, 1360).

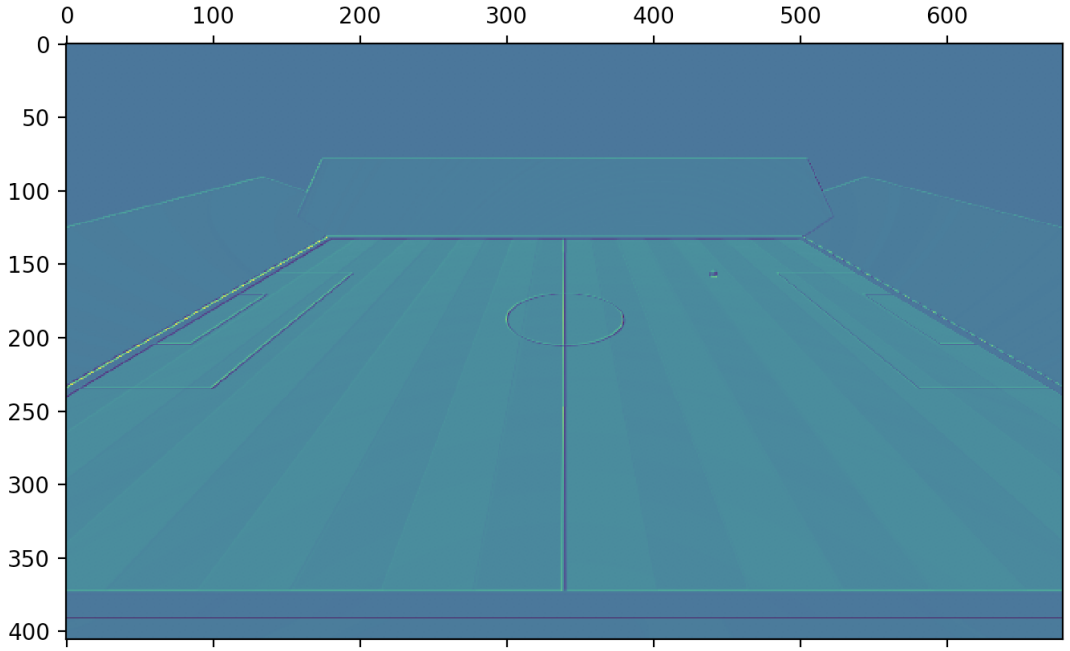


Figure 3.14. Example of filtered image after the first convolutional layer (406, 679).

This assumption and validation brought to a meaningful design choice. Instead of using seven convolutional layers, the new model used just four convolutional layers. This new architecture resulted in a more accurate prediction as well as a faster product. In Figure 3.15 the final model architecture is displayed.

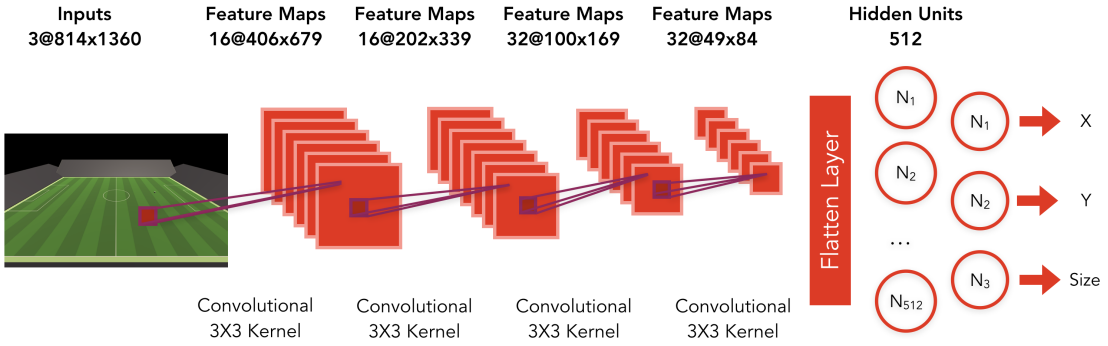


Figure 3.15. The deisgn network to estimate the $(x, y, size)$ features.

3.5.6 Definition of a Good Result (Object Detection and Size Estimation)

Due to a rescaling step of the data labels, loss and accuracy values were not relevant to have an insight into the 3D accuracy of the next model. For this reason, a training step of the *temporal* net defined in the previous subsections has been taken using the samples used for training by the *convnet* when a specific loss value was given. After some iterations and attempts leading to different train loss values *[5e-07, 5e-06, 5e-05, 5e-04, 5e-03, 5e-02]*, a threshold to recognize the problem solved has been discovered. With a test loss value in the order of *5e-04*, a performing 3D prediction could have been obtained.

3.5.7 Hyperparameter Tuning

To train the model an incremental approach over the number of train samples has been adopted. Starting with a lower number of samples to test the convergence power of the model and ending up using almost 1000 examples to increment the generalization capability of the algorithm. A hyperparameter search has been accomplished "babysitting" the training with different optimizers and learning rates. Adam [18] with the paper parameters seems the one that converges faster and better. Regarding the number of filters, due to the fact the ball features shouldn't be hard to detect, 16 and 32 have been used for the first two convolutional layers and the last two convolutional layers, respectively, considering the RAM constraint too. From the point of view of activation functions, for each hidden layer, ReLU [50] has been employed. The reason for this choice regards the fact that no other paper dealt with a similar task and iterate over all possible activation functions would not be feasible in terms of time. ReLU performs well when convolutional layers are adopted, and it does not suffer from the vanishing of the gradient problem as Tanh and Sigmoid [53]. In each layer, Xavier initializer [51] has been adopted for weights initialization purposes. Other more recent initializers are now available, but for the same reason concerning the activation functions, no further investigation has been arranged.

As mentioned in Subsection 3.5.1, the simplicity of finding the model parameters was led by the opportunity of generating more data quickly. The only concern regarded the learning capability of the algorithm. After determining a model able to overfit, the remaining step is to increase the generalization power of the algorithm adding more data until a suitable solution is obtained. For this reason, after tuning over the optimizers and the learning rate values, a model was immediately found in the next training iteration. Increasing the number of samples, the network generalization capabilities improved significantly.

The result is a model capable of predicting the $(x, y, size)$ of a small object in high-resolution images; from here after referenced as *ball* net.

3.6 Plugging the Models together

After pushing the *ball* net to the maximum test accuracy using almost 1000 samples, it was necessary to retrain the *temporal* net using the *ball* net predictions over new samples. The other $\sim$ 3000 examples have been employed for this step. The *ball* net predicted the $(x, y, size)$ of each image, then after a first preprocessing the temporal model had been fed with the predictions for further training, resulting in impressive final test error. In Figure 3.16 the final architecture is provided.

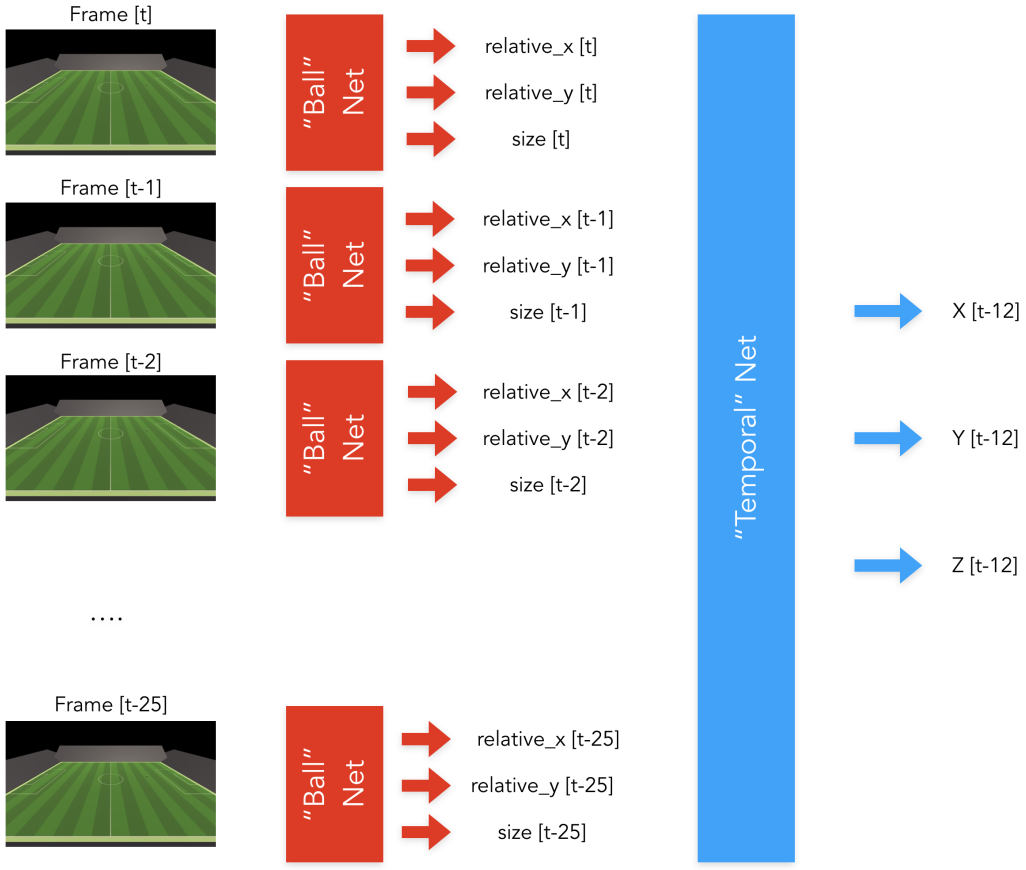


Figure 3.16. The combination of the explained models in a unique single pipeline.

The last result brought to the conclusion that the noise used in the first definition of a model for the 3D position reconstruction was more than the actual error created by the features predictor.

3.7 Technologies and Code

3.7.1 Software

For the models development, Keras library [58] with Tensorflow [57] backend has been used. Keras let fast implementation and the possibility to integrate functions and methods easily. It incorporates the most famous base networks as VGG and ResNet [23], which can be quickly called and used for image classification/detection purposes.

For preprocessing purposes other libraries as Scikit-learn [44], Pandas [59] and OpenCV [28] have been exploited to speed up the entire pipeline.

3.7.2 Hardware

To train deep neural networks, a GPU is required. The *ball* net, even if composed just of five layers, is an expensive model due to the limitation concerning images dimensions. For this reason, the final training took almost 3 hours with 850 samples, 32 batch size and 500 epochs. The GPU is a Nvidia GTX 1080Ti with 11 GB of RAM. On the other hand, the *temporal* net was not expensive in terms of computational requirements, due to the presence of a single layer and only $3 * 25$ features for each input sample. For this reason, until the definition of the object detector, the code has been run locally using a PC.

3.7.3 Code

The implementation of the work is released and made available under GNU General Public License, at the URL https://github.com/FilippoPedrazziniFP/Ball-3d-coordinates on Github.

Chapter 4

Results

This chapter represents the results obtained using the empirical method. For this reason, an analysis of the collected values is taken, trying to investigate and explain the motivations behind them. Both a quantitative and a qualitative analysis have been employed to discuss the retrieved results. Numerical performances can be considered an excellent metric to prove the effectiveness of a method, but when a point of comparison is not present, the definition of what is a good result is challenging to manage. For this reason, the context led to the decision of determining a suitable boundary for the problem in a qualitative process.

4.1 Definition of a Good Result (3D Position Estimation)

The work addresses the problem of defining a solution capable of estimating the 3D position of an object starting from a sequence of 2D images. A single case study has been investigated to test the designed architecture. Based on the application and the domain, different requirements regarding performances are required. In the specific case of soccer match statistics, to achieve a good result, the ball 3D position must be computed with a certain accuracy such that the predictions can be then aggregated and compared with real statistics. The thesis did not analyze the results from this perspective due to lack of real data, but a boundary has been defined to classify a good/bad outcome. An error in the range of $(0.0, 1.0)$ of MAE, which corresponds to the sum of error in the three directions up to 1.0 meters, has been considered as an excellent achievement and a positive answer to the formulated problem. A qualitative analysis was required due to the limitations in the available data.

4.2 Results

A deep learning approach with the combination of multiple networks and techniques has been created to achieve an optimal result to tackle the problem of estimating the 3D position of an object. The architecture is composed of two networks: the *ball* net and the *temporal* net. The former is a five layers convolutional neural network, with the intent to extract the position and the size of the ball in the 2D images, while the latter is a shallow network with a single 1D convolutional layer that exploits the temporal information and reconstructs the 3D position of the ball. Following, for each intermediate step the numerical results obtained during the implementation:

1. Metric: recap over the metric used to test the different models.
2. Trajectory extraction: the intermediate/final results obtained in the phase of estimating the 3D position of the object when flying.
3. Flying - Not Flying Classification: the results obtained over the task of estimating if the ball is flying or not flying employing a model similar to the one designed for the trajectory estimation.
4. Two tasks one Model: the scores obtained using the same model to predict the 3D position of the ball over a more general dataset assuming the inherent solved problem of estimating if the object is flying or not.
5. Size estimation and Object Detection: an example of graphical prediction over the test set.
6. Final results: the final results obtained combining the two models over a new portion of data.

4.2.1 Metric

As explained in Subsection 3.3.3, to test the models MAE has been used when dealing with numerical values. MAE is a good performance validator due to the possibility to measure the distance between the prediction and true value without applying transformation over the error. Although to cover the classification task, accuracy metric has been employed. All the results shown below will refer to these two metrics. In order to keep the same consistency over different models, the same number of samples have been used to test the different networks. In this case, 200 examples were considered enough to prove the generalization capabilities of each model.

4.2.2 Trajectory Extraction

Going back to Subsection 3.3.4, three different baseline approaches have been combined to have an idea about the performances of the method before going further in the implementation of more complex models.
Table 4.1 shows a summary of the accomplished scores. The best coherent result will be employed as the point of comparison to measure the new model performances. In such a case the baseline with 25 frames as input and the no-coplanar points to compute the projection matrix will be taken into account to reflect the difference with the other models. Even if the baseline with more frames has reached a more accurate result, the comparison wouldn't be meaningful due to different input data.

	MAE	
Extracted Points	25 Frames	100 Frames
coplanar	2.738	1.835
no-coplanar	1.159	0.381

Table 4.1. The baseline scores using different configurations and number of frames.

The two methods to improve the simple baseline, create a better solution in terms of MAE; as a matter of fact, increasing the number of frames to take into account generates a more robust prediction. Furthermore, considering the no-coplanar points in the computation of the projection matrix increase the information regarding the 3D environment, resulting in a more solid projection matrix.

After the definition of what to consider as the baseline, the first deep approach has been tested as explained in Subsection 3.3.6. Table 4.2 shows the summary of the results obtained using both the most performing baseline and the LSTM approach over the clean and noisy dataset.

	MAE	
Method	No-Noise	Noise
Baseline	1.159	5.967
LSTM	3.243	3.824

Table 4.2. The scores of the first deep approach compared to the baseline method.

Different levels of noise have been tested in order to have a more general idea about the performances of these models under different configurations; here a comparison over the higher values of noise have been analyzed to show the considerable gap between the different approaches. As expected, the baseline method reaches an incredibly low result over the noisy dataset (~ 6 of MAE), while the recurrent approach is more robust with

a similar score to the without-noise solution ($\sim$ 3.5 of MAE as in the without noise implementation). The obtained MAEs don't prove to be safe, but from the results, it is evident that a network-based approach represents the right path to follow due to the capabilities in dealing with a certain amount of noise. As a matter of fact, the design choices followed these partial results.

In terms of error, the sequence to one approach reaches $\sim$ 3.5 of MAE, which cannot be regarded as a reliable result as explained in Section 4.1. After an architecture iteration and some experiments, a stable solution for the 3D trajectory estimation has been discovered. An additional information has been added to the features which correspond to the size of the object in the 2D images. Moreover, a new network has been designed to speed up the prediction in order to exploit the solution in a real case scenario. Table 4.3 represents a summary of the results with the new feature compared to the basic approach.

	MAE	
Method	No-Noise	Noise
Baseline	1.159	5.967
LSTM	0.028	0.237
1D Conv	0.029	0.875

Table 4.3. The new deep approaches compared to the baseline .

As defined in Section 4.1 the results can be considered valid and robust compared to the basic geometrical scores. Even with the strong assumption regarding the noise over the size of the ball, a good result has been obtained using the same model. Moreover, even if LSTM seem to perform better, a trade-off between speed and error should be taken into account in the decision of which model choose in a real-case scenario.

4.2.3 Flying vs Not Flying Classification

As explained in Section 3.4, the same architecture used for the trajectory estimation task has been used to solve the binary classification issue. High-performing results have been obtained using the *temporal* net, giving assurance in the definition of the same network for two tasks. As explained in Section 3.2, the problem of classifying if an object is flying or not can be generalized to the more significant problem of estimating the 3D position of the same object while doing a trajectory. In Table 4.4 the classification accuracy obtained over the clean and noisy dataset is provided.

	Accuracy	
Method	No-Noise	Noise
1D Conv	1.000	0.998

Table 4.4. The results of the binary classification task in terms of accuracy.

4.2.4 Two Tasks one Model

The hypothesis over the assumption of using the same network for two different steps has been confirmed looking at the results. A new test over a more general dataset with the same network was the next step toward the investigation of employing a single network for the prediction of the 3D position of the ball having as input a sequence of $(x, y, size)$ features. Table 4.5 shows the results on the more extensive dataset. In the analysis, the baseline approach is not present since the basic method has been designed to tackle the only problem of estimating the 3D position of the ball while flying. The presence of the baseline method would have resulted in an inconsistent comparison, and a logical proof regarding the error gap has already been investigated.

	MAE	
Method	No-Noise	Noise
1D Conv	0.090	0.508

Table 4.5. The results of the more general task of estimating the 3D position of the ball.

4.2.5 Size Estimation and 2D Object Detection

As explained in Subsection 3.5.6, a results-based approach has been followed to decide when to stop the training/tuning procedure of the deep convolutional neural network. Due to the complexity of plugging the model together for a single training step, the need of learning them separately was required. A pipeline is now available to predict the 3D position of an object having as input a sequence of 2D images. In Figure 4.1 the prediction of one sample belonging to the test set is provided.

4.2.6 Pipeline Results

After training and tuning the different models, a final plugging was required to create a pipeline able to predict the 3D position of the ball starting from a sequence of 2D images. As explained in Subsection 3.6, final training has been executed to test on new samples the effectiveness of the two networks. A final score of 0.130 of MAE has been reached, concluding the implementation with a positive result starting from the

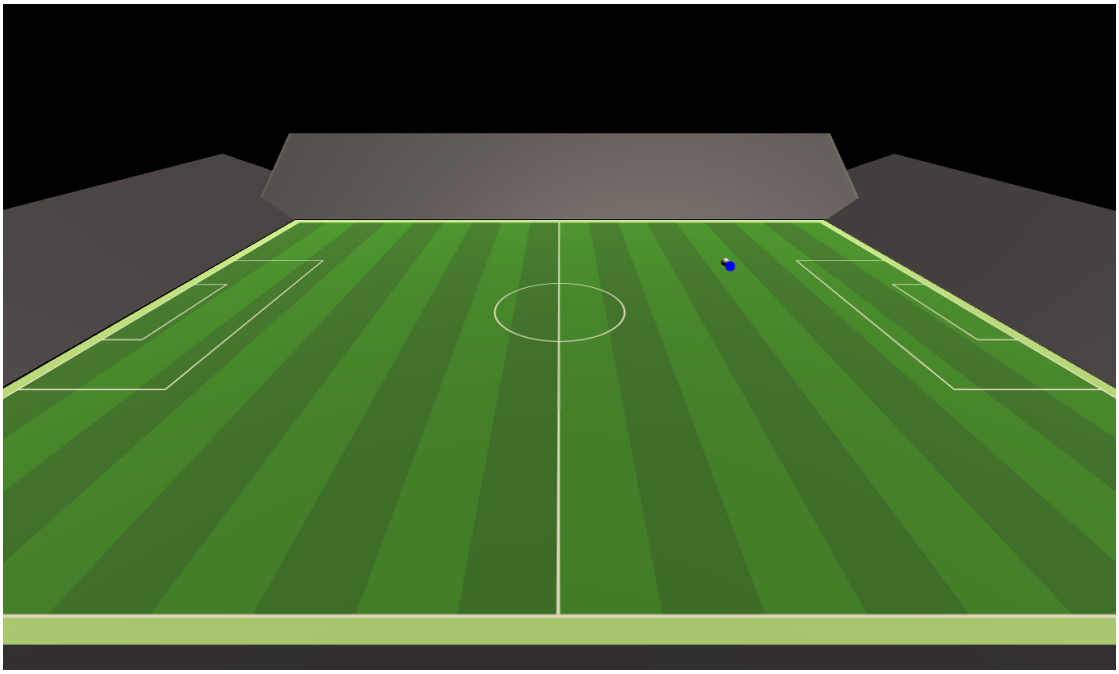


Figure 4.1. Example of prediction on a test sample.

assumption in Section 4.1 where a boundary to classify the result as good/bad has been defined. A comparison with a non-sequential method was required to prove the robustness of employing the *temporal* net. For this reason, a Neural Network (Simple Net) with a single hidden layer has been developed to test the task over the one to one mapping without exploiting the temporal information of multiple frames. In Figure 4.2 the Simple Net developed for comparison purposes is displayed.
Table 4.6 shows the comparison of the results over the *ball* net predictions and the dataset ground truth regarding the $(x, y, size)$ features. A consistent difference is present between the one-to-one neural network and the *temporal* net. When a perfect features prediction is available, the need for a temporal based solution is not required; on the other hand, a small error in the prediction will always be present due to the object and images characteristics.

	MAE	
Data	Temporal Net	Simple Net
Ball Net Predictions	0.130	0.641
Correct Values	0.090	0.100

Table 4.6. The final results over the real values and the predictions of the *ball* net.

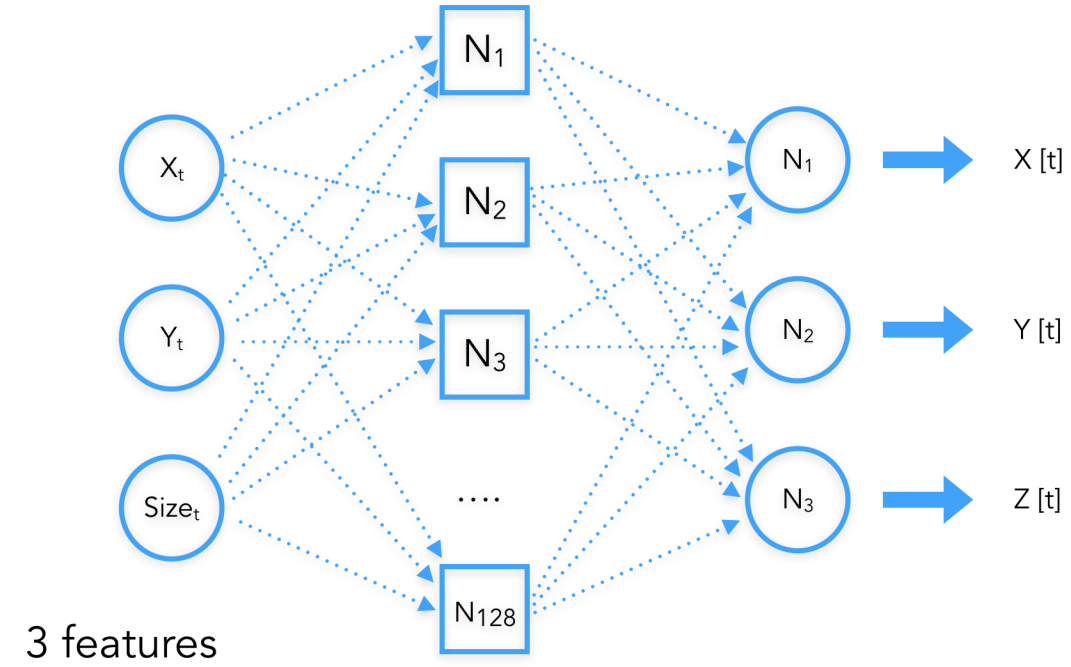


Figure 4.2. The Neural Network implemented for comparison purposes.

Chapter 5

Discussion

The chapter gives an insight into the applications and limitations of this work.

5.1 Applications and Limitations

The work addressed an essential aspect of the computer vision field. A final value of 0.130 of MAE has been obtained combining the two models. The result corresponds to an error of 13 centimetres in the sum of the three directions (x, y, z) on a comprehensive synthetic dataset. As explained in Section 4.1 the robust result is relative to the assumption that the defined boundary can be considered qualitatively good in case of soccer statistics. On the other hand, the images extracted using a real camera do not have the same quality and precision as a Unity camera; for this reason, the noise in the prediction of the $(x, y, size)$ features could be higher than the ones extracted using a synthetic dataset. In a real field, the presence of many different objects and obstacles can bring rumour and create confusion with respect to the object detector, decreasing the model performances. At the same time, testing the *temporal* net using a high noise value, it demonstrates the effectiveness and robustness of the method when dealing with error. In the last case, the performances are affected, but still in a consistent range of working values. In subsection 4.2.4, the detailed results with 25% of noise on the ball $size$ and a random uniform noise on the (x, y) features are provided.

The implemented architecture is suitable for different computer vision applications. Where the necessity of localizing an object in the 3D space relative to a specific field is required, the same pattern could be applied. The primary requirements regard the stability of the camera and the input format. An image sequence must be passed as the input of the network. Using a single frame is still not possible to reconstruct the 3D position of an object with high accuracy because the prediction concerning the $(x, y, size)$ of the object will always have a critical noise, which leads to an unstable one-to-one

forecast. Even from a human perspective, it is difficult to distinguish and estimate the relative position of an object with a single frame. This solution extends the opportunities of extracting more and useful information given a set of assumptions regarding the environment.

The work focused on the design of a new model able to predict the 3D position of an object having as input a sequence of 2D images. The underlined model has been trained to deal with a soccer ball, for this reason, the architecture is not able to generalize over other objects. Further training on different data is necessary where the need to localize another object is present. On the other hand, the same pipeline can be used to solve the estimation of the 3D position of a small object with high-resolution images.

Another consistent limitation regards the data. This work started with the creation of a suitable dataset that could simulate a real soccer match. On the other hand, the easily extracted information using Unity, cannot be similarly gathered in a real environment. The need for a complicated labelling process is required to get the truth about the 3D position of the ball as well as the 2D size. A possible solution is to use two synchronized cameras and apply a combined geometrical transformation. Still, the necessity of labelling the 2D images from different views is required. Resources are mandatory to achieve and collect a proper dataset.

Chapter 6

Conclusion

This chapter represents the end of the work. For this reason, a final section is followed by the investigation of further improvements of the solution.

6.1 Conclusions

A new pipeline has been created to deal with the estimation of the 3D position of an object starting from a sequence of 2D images. Two networks have been designed to reach high accuracy over the final prediction. The former (*ball* net) is a Convolutional Neural Network with the objective to extract the 2D information of the object in the image; more specifically the (x, y) location and the size of the ball. The latter (*temporal* net) is a deep model with the intention to extract the final 3D position of the object exploiting the temporal information gathered by the *ball* net. Combining the two models a final error of 0.130 of MAE has been achieved, resulting in an accurate prediction over new unseen samples. This last validation brings to a positive conclusion of this work, and an answer to the research question instantiated in Section 1.2 is given; the possibility to create a data-driven solution to extract the 3D position of an object is realistic and efficient, resulting in an excellent accuracy compared to older computer vision methods on the studied case. The work focused on a single case study, and the need to test this approach on real data and different domains is necessary to examine the robustness of the architecture and define a more general answer to the research question when dealing with other fields. A boundary to establish a good/bad result has been created using a qualitative process. The limitation over data and resources led to this last decision.

6.2 Future Works

Even if a performing model has been developed using multiple techniques, there are still some improvements and new phases that should be tackled in the future. In fact, the

implemented method starts from the assumption that the camera is fixed, which means that the network learns a single camera matrix intrinsically. In a real environment, when dealing with a broadcast camera, there is a high probability that the camera moves according to a specified pattern (following the density of the players or the ball position in the court); for this reason, a more robust approach should be implemented to deal with a moving camera. One possible solution is to compute for each frame the 12 camera parameters using OpenCV framework [28], and pass them as input with the $(x, y, size)$ features. In this way, for each frame, the camera parameters will represent an identifier that can be exploited by the learning algorithm for better mapping. The drawback of this suggested method regards the feasibility of getting real data. For each frame, the camera parameters are computed using some known points both concerning image and court coordinates. Given these points, the camera matrix can be estimated and passed to the model. To extract these points dynamically there are few methods as [60] and [61], but before applied, they should be validated in terms of accuracy and reliability. A proximate result can lead to noisy features and a worse 3D estimator.

Another less relevant aspect, which should be taken into account, is the presence of the ball in the 2D images. All the generated models started from the assumption that the ball is within the image, while in a more general application, the video, for few seconds, could have frames without the presence of the ball. A possible solution is to add another branch to the implemented *ball* net to classify if the ball is present or not in the frame. The task should not affect the performances of the overall pattern.

State of the art paradigms and best practices have been used to design the pipeline composed of the two networks; however, deep learning techniques proved to be better when an end-to-end approach could be applied. A single more complex error surface can be learnt effectively by a single model. For this reason, the possibility to create a unique network for both tasks could improve performances and prediction speed. One possible solution is to use 3D convolutions, creating a model able to extract both the individual images information and the temporal dependencies. Although, the new architecture could lead to different problems both concerning computational resources and convergence. The images and network dimensions require top hardware both regarding computation and memory.

Bibliography

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *ArXiv e-prints*, Nov. 2013.

[2] T. Kim, Y. Seo, and K.-S. Hong, "Physics-based 3D position analysis of a soccer ball from monocular image sequences", in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan. 1998, pp. 721–726.

[3] Y. Ohno, J. Miura, and Y. Shirai, "Tracking players and estimation of the 3D position of a ball in soccer games", in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, 2000, 145–148 vol.1.

[4] C. Huang, M.-C. Tien, Y.-W. Chen, W.-J. Tsai, and S.-Y. Lee, "Physics-based ball tracking and 3d trajectory reconstruction with applications to shooting location estimation in basketball video", pp. 204–216, Apr. 2009.

[5] I. Reid and A. North, "3D Trajectories from a Single Viewpoint using Shadows", in *Proc. BMVC*, 1998, pp. 863–872.

[6] J. Ren, J. Orwell, G. A. Jones, and M. Xu, "A general framework for 3D soccer ball estimation and tracking", in *2004 International Conference on Image Processing, 2004. ICIP '04*, Oct. 2004, 1935–1938 Vol. 3.

[7] "Extracting 3D information from broadcast soccer video", en, *Image and Vision Computing*, pp. 1146–1162, Oct. 2006.

[8] E. Ribnick, S. Atev, and N. P. Papanikolopoulos, "Estimating 3D Positions and Velocities of Projectiles from Monocular Views", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 938–944, May 2009.

[9] *Unity engine*, https://unity3d.com.

[10] A. M. TURING, "I.—computing machinery and intelligence", *Mind*, pp. 433–460, 1950.

[11] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain”, *Psychological Review*, pp. 65–386, 1958.

[12] G. L. Shaw, “Donald hebb: The organization of behavior”, in *Brain Theory*, G. Palm and A. Aertsen, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 231–233.

[13] D. Clevert, T. Unterthiner, and S. Hochreiter, “Fast and accurate deep network learning by exponential linear units (elus)”, *CoRR*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”, *CoRR*, 2015.

[15] P. Ramachandran, B. Zoph, and Q. V. Le, “Searching for activation functions”, *CoRR*, 2017.

[16] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks”, in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[17] N. Qian, “On the momentum term in gradient descent learning algorithms”, *Neural Networks*, pp. 145–151, 1999.

[18] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization”, *ArXiv e-prints*, Dec. 2014.

[19] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization”, *J. Mach. Learn. Res.*, pp. 2121–2159, Jul. 2011.

[20] M. D. Zeiler, “ADADELTA: an adaptive learning rate method”, *CoRR*, 2012.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting”, *Journal of Machine Learning Research*, pp. 1929–1958, 2014.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *ArXiv e-prints*, Dec. 2015.

[24] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units", *CoRR*, 2015.

[25] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Comput.*, pp. 1735–1780, Nov. 1997.

[26] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[27] T. Shao-xiong, L. Shan, and L. Zong-ming, "Levenberg-Marquardt algorithm based nonlinear optimization of camera calibration for relative measurement", in *2015 34th Chinese Control Conference (CCC)*, Jul. 2015, pp. 4868–4872.

[28] *OpenCV library*, https://opencv.org/.

[29] M. T. Ahmed, E. E. Hemayed, and A. A. Farag, "Neurocalibration: A neural network that can tell camera calibration parameters", in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, 463–468 vol.1.

[30] R. B. Girshick, "Fast R-CNN", *CoRR*, 2015.

[31] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks", *CoRR*, 2015.

[32] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN", *ArXiv e-prints*, Mar. 2017.

[33] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", *CoRR*, 2015.

[34] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector", *CoRR*, 2015.

[35] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger", *CoRR*, 2016.

[36] *ImageNet*, en, http://www.image-net.org.

[37] Y. Cao, Z. Wu, and C. Shen, "Estimating Depth from Monocular Images as Classification Using Deep Fully Convolutional Residual Networks", *ArXiv e-prints*, May 2016.

[38] B. Li, Y. Dai, H. Chen, and M. He, "Single image depth estimation by dilated deep residual convolutional neural network and soft-weight-sum inference", *CoRR*, 2017.

[39] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. Yuille, "Towards unified depth and semantic prediction from a single image", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 2800–2809.

[40] F. Liu, C. Shen, G. Lin, and I. Reid, "Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2024–2039, Oct. 2016.

[41] *NYU Depth V2 dataset*, https://cs.nyu.edu/silberman/datasets/nyu_depth_v2.html.

[42] Z. Boukhers, K. Shirahama, F. Li, and M. Grzegorzek, "Object detection and depth estimation for 3D trajectory extraction", in *2015 13th International Workshop on Content-Based Multimedia Indexing (CBMI)*, Jun. 2015, pp. 1–6.

[43] FIFA.com, *Fédération Internationale de Football Association (FIFA) - FIFA.com*, en-GB, http://www.fifa.com/.

[44] *Scikit-learn: Machine learning in Python — scikit-learn 0.19.1 documentation*, http://scikit-learn.org/stable/.

[45] *Sklearn.preprocessing.StandardScaler — scikit-learn 0.19.1 documentation*, http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[46] D. Duan, M. Xie, Q. Mo, Z. Han, and Y. Wan, "An improved Hough transform for line detection", in *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, Oct. 2010, pp. V2–354–V2–357.

[47] B. Přibyl, P. Zemčık, and M. Čadık, "Absolute pose estimation from line correspondences using direct linear transformation", en, *Computer Vision and Image Understanding*, pp. 130–144, Aug. 2017.

[48] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities", eng, *Proceedings of the National Academy of Sciences of the United States of America*, pp. 2554–2558, Apr. 1982.

[49] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional Time Series Forecasting with Convolutional Neural Networks", *ArXiv e-prints*, Mar. 2017.

[50] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Haifa, Israel: Omnipress, 2010, pp. 807–814.

[51] S. Koturwar and S. Merchant, "Weight initialization of deep neural networks(dnns) using data statistics", *CoRR*, 2017.

[52] T. Lei, Y. Zhang, and Y. Artzi, "Training rnns as fast as cnns", *CoRR*, 2017.

[53] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, pp. 303–314, Dec. 1989.

[54] M. Shi and V. Ferrari, "Weakly supervised object localization using size estimates", *CoRR*, 2016.

[55] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *CoRR*, 2014.

[56] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004.

[57] *TensorFlow*, en, https://www.tensorflow.org/.

[58] *Keras Documentation*, https://keras.io/.

[59] *Python Data Analysis Library — pandas: Python Data Analysis Library*, https://pandas.pydata.org/.

[60] D. Farin, S. Krabbe, W. Effelsberg, and P. H. N. de With, "Robust Camera Calibration for Sport Videos using Court Models", in *Proceedings of SPIE*, Bellingham, WA: SPIE, 2004, pp. 80–91.

[61] S. Donné, J. De Vylder, B. Goossens, and W. Philips, "MATE: Machine Learning for Adaptive Calibration Template Detection", eng, *Sensors (Basel, Switzerland)*, Nov. 2016.