ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DEPARTMENT OF COMPUTER SCIENCE
AND ENGINEERING

ARTIFICIAL INTELLIGENCE

MASTER THESIS

in

Machine Learning for Computer Vision

BALL TRACKING IN PADEL VIDEOS USING CONVOLUTIONAL NEURAL NETWORKS

CANDIDATE

Michele Calvanese

SUPERVISOR

Prof. Samuele Salti

CO-SUPERVISOR

Prof. Carlos Andujar Gran

Academic year 2022-2023

Session 2nd

Contents

1 Introduction **1**

1.1 Context 1

1.2 Motivation 2

1.3 Goal 4

1.4 System description 5

1.5 Structure of the document 6

2 Theory **7**

2.1 Overview on Convolutional Neural Networks 7

2.2 Why convolutions 8

2.2.1 Translation equivariance and locality 9

2.2.2 Channels 12

3 State of the Art **15**

3.1 Existing products for sports analysis 15

4 System description **17**

4.1 Overview 17

4.2 Ball detection 17

4.2.1 Architecture 18

4.2.2 Extracting the ball position from the heatmap 20

4.2.3 Results overview 21

4.3 Trajectory fitting 21

4.3.1 Fitting a parabola to each frame 21

4.3.2 Linking trajectories 28

4.3.3 Results overview 31

5 Results **33**

5.1 Dataset 33

5.2 Ablation study on the ball detection 34

5.2.1 Training procedures 34

5.2.2 Detection results 38

5.3 Analysis of the trajectory fitting 42

5.3.1 Choosing the right fitting parameters 42

5.3.2 Analysis of different game situations 42

5.4 Discussion 44

6 Conclusion **45**

Bibliography **46**

1 Introduction

1.1 Context

The following context section provides an overview of the academic background and research collaboration pertinent to the thesis. It highlights the affiliation with the University of Bologna, the experience as an exchange student at the Universitat Politècnica de Catalunya (UPC) in Barcelona, and the collaboration with Professor Carlos Andujar in the VirViG research group.

The academic journey leading to this master's thesis began at the University of Bologna, renowned for its rich academic tradition and commitment to excellence in higher education. As a student of Artificial Intelligence, I have had the privilege of studying and engaging with a diverse range of subjects, acquiring a solid foundation of knowledge in the latest developments in AI and Computer Vision.

Furthermore, as part of my academic pursuits, I had the valuable opportunity to embark on an exchange program at the Universitat Politècnica de Catalunya (UPC) in Barcelona. This immersive experience allowed me to broaden my horizons, both culturally and academically, by exploring new perspectives and engaging with a vibrant academic community.

Throughout my time at the UPC, I had the privilege of collaborating closely with my provessor, a highly regarded expert in the field of computer graphics. He leads the VirViG research group at the UPC, known for their groundbreaking research and innovative contributions in computer graphics and computer vision. This collaboration has been instrumental in shaping the focus and direction of my master's thesis.

By combining the academic resources and diverse perspectives of the University of Bologna, the enriching experience as an exchange student at the UPC, this master's thesis aims to delve into the usage of Deep Learning algorithms to address the task of ball tracking in videos of Padel matches and make meaningful contributions to the field of sports video analysis.

The subsequent chapters will explore the current State of the Art in ball tracking in videos, the developed system, and a thorough evaluation of its performance, providing a comprehensive analysis and discussion of the research findings. The ultimate goal is to contribute to the existing body of knowledge, foster academic discourse, and offer potential recommendations for future research and practical implications.

1.2 Motivation

The motivation behind this research stems from the growing popularity of Padel, particularly in Spain and Italy, and the need for advanced video analysis techniques in this sport. Padel, a racquet sport that combines elements of tennis and squash, has experienced a remarkable surge in popularity over the past decade, captivating enthusiasts and attracting a wide range of players. The game's fast-paced nature, strategic elements, and accessibility have contributed to its rapid expansion, making it an intriguing subject for research and technological advancements.

In recent years, the availability of professional Padel matches on the internet, with standard views and high-quality video footage, has presented a valuable resource for training and developing machine learning models. These publicly accessible matches offer an excellent opportunity to collect large-scale datasets and extract valuable insights from the gameplay. By leveraging this wealth of data, researchers can advance the field of sports analytics

and contribute to the overall understanding of Padel strategies, player performance, and game dynamics.

While previous projects have focused on positional tracking of players in Padel matches, this research specifically centers on the challenging task of ball tracking. Tracking the ball in Padel poses unique difficulties due to its small size and high speed, necessitating specialized techniques and algorithms. By focusing on ball tracking, this study aims to provide a comprehensive analysis of ball trajectories, interactions with players, and their impact on the overall game flow. Such insights can have far-reaching implications for coaches, players, and even technology developers seeking to enhance training methodologies and improve player performance.

Moreover, the potential benefits of this research extend beyond professional matches. Amateur Padel clubs and players can also derive considerable advantages from advanced video analysis tools. By accurately tracking the ball's movements and interactions in real-time, these clubs can gain valuable feedback on their gameplay, identify areas for improvement, and enhance their overall performance. The application of automated ball tracking systems can empower players at all skill levels, facilitating a deeper understanding of the game and helping them refine their strategies and techniques.

In summary, this research project aims to address the emerging needs in the field of Padel analysis by focusing on the challenging task of ball tracking. With the rising popularity of Padel in Spain, Italy, and beyond, the increasing availability of professional match footage, and the potential benefits for both professional and amateur players, this study seeks to contribute to the existing body of knowledge in Padel analytics. By developing an accurate and efficient ball tracking system, this research endeavors to provide valuable insights into the game dynamics, assist in player training and development, and pave the way for future advancements in Padel technology.

1.3 Goal

The primary goals of this research project are to design, implement, and evaluate a comprehensive system for ball tracking in Padel matches. The system aims to extract and analyze the trajectories of the ball from videos of amateur matches captured from an overhead or standard view. The following goals outline the specific objectives of this study:

- Develop a robust system: Design and implement a reliable and accurate ball tracking system capable of handling the challenges posed by Padel match videos. The system should effectively detect the ball and track its movements throughout the duration of the match.
- Trajectory extraction: Retrieve the associated trajectories of the ball, capturing its path between different events such as racket hits, wall interactions, or ground bounces. The system should generate a model that describes the behavior of the ball over time, providing a comprehensive understanding of its movement patterns during gameplay.
- Output specifications: Define the specific outputs of the system, which include the low-level detection of the ball in the video, a model of the ball's path between events, and ideally, a complete description of the ball's trajectory throughout the entire match duration. These outputs will facilitate further analysis, strategic insights, and performance evaluation.
- Evaluation and validation: Assess the performance and effectiveness of the developed system through rigorous evaluation methodologies. Conduct comprehensive experiments and comparisons against ground truth data to measure the accuracy, robustness, and reliability of the ball tracking system.

By achieving these goals, this research project aims to contribute to the field of Padel analytics by providing a reliable and efficient system for ball tracking. The outputs generated by the system will enable coaches, players, and researchers to gain deeper insights into the game dynamics, improve training methodologies, and explore new possibilities for enhancing player performance in Padel.

The system developed here focuses on representing the ball's trajectory in image space (2D). A future development is to consider the transition to 3D space. By exploring the possibility of incorporating depth information, the system can potentially enhance the accuracy and realism of the ball's trajectory representation.

1.4 System description

The system developed for ball tracking in Padel matches encompasses several key components. The following description provides an overview of the system's architecture and functionality.

The core of the system utilizes a Convolutional Neural Network (CNN), implemented using the PyTorch framework [1]. Specifically, the ball detection module employs the TrackNetV2 [2] architecture, which bears similarity to the well-known UNet [3]. The output is a heatmap that indicates the likelihood of finding the ball at different positions within the frame. In terms of tracking, two main variants of the input are tested. The first one uses multiple consecutive frames as input. The second one functions akin to a recurrent neural network, utilizing one input frame and the output of the previous frames. To optimize the detection performance, an ablation study is conducted, exploring these two input variants and different parameters such as the number of input frames and the training schedule.

Regarding trajectory fitting, the system employs a RANSAC-style method for fitting parabolic curves [4]. The process initiates with the identification of seed triplets across different frames. Subsequently, the trajectory is built incrementally, employing the RANSAC-style method to accurately model and describe the ball's path. This approach ensures the system can capture the ball's behavior during gameplay, providing a comprehensive representation of its trajectory.

Overall, the system integrates a CNN-based ball detection module, multiple tracking variants utilizing consecutive frames, and a trajectory fitting mechanism employing a RANSAC-style approach. These components collectively contribute to the system's ability to robustly track the ball in Padel matches, providing valuable insights into its movement patterns and behavior during gameplay.

1.5 Structure of the document

This document will delve into the development of a ball tracking system, specifically targeted at Padel, based on a Convolutional Neural Network that detects the ball in a match video. On these detections, trajectories are then fitted. The chapters are:

- Theory: some background on convolutional Neural Networks is given;
- State of the Art: a brief description of the current State of the Art in sports analysis, and especially on ball tracking in various sports;
- System description: a thorough description of the developed system;
- Results: analysis of the performance of the system, and possible improvements;
- Final remarks and conclusions.

2 Theory

2.1 Overview on Convolutional Neural Networks

Convolutional neural networks (CNNs) are a powerful family of neural networks that are specifically designed to handle image data efficiently. Unlike traditional fully-connected MLPs (Multi-Layer Perceptrons), CNNs are able to preserve the spatial structure of images by using convolutional layers and pooling layers and use it at their advantage.

CNN-based architectures are now ubiquitous in the field of computer vision, and have become so dominant that hardly anyone today would develop a commercial application or enter a competition related to image recognition, object detection, or semantic segmentation without building off of this approach. The design of modern CNNs owes its inspiration to biology, group theory, and a substantial amount of trial and error. Not only are they efficient in achieving accurate models, but they are also computationally efficient because they require fewer parameters than fully-connected architectures, and are easy to parallelize on GPUs.

CNNs are not limited to handling image data and have been successfully adapted for use in other domains such as audio, text, time series analysis, graph-structured data, and recommender systems. In fact, some clever adaptations of CNNs have brought them to bear on graph-structured data and in recommender systems.

This chapter covers the basic operations of CNNs, which include convolutional layers, padding and stride, pooling layers, the use of multiple channels at each layer, and a brief discussion of the structure of modern architectures.

Convolutional layers use filters to identify features in the input image and generate a feature map. Padding and stride control the size of the feature map, while pooling layers aggregate information across adjacent spatial regions to reduce the dimensionality of the output and improve the efficiency of the network.

In Chapter 3, the state of the art in object tracing will be explained, and in Chapter 4 the specific architecture used in this project will be shown.

2.2 Why convolutions

Fully connected network are not well suited for learning representations of images. The reason is that the number of parameters grows very fast with image size and hidden dimension size: given an image $\mathbf{X}$ with dimensions $W \times H$, the hidden representation after the first fully connected layer of an MLP and before the activation function looks like this:

$$\mathbf{h}_i = \mathbf{u}_i + \sum_{k=1}^{C} [\mathbf{W}]_{i,k} \, \mathbf{x}_k, \qquad i \in \{ 1, \ldots, W \cdot H \}, \tag{2.1}$$

where $\mathbf{x}$ is the flattened input image, $\mathbf{h}$ its flattened hidden representation, $\mathbf{W}$ the weight matrix and $\mathbf{u}$ the bias vector. The number of parameters for the weight matrix $\mathbf{W}$ is therefore $C \cdot W \cdot H$. For instance, a 100×100 pixel image with a hidden representation of dimension 1000 would require 10^7 parameters. A system with these characteristics has a poor representation capability. Any practical system would require a larger input image and a higher dimensional hidden representation. This means that an enormous amount of memory, computational resources and training data would be required to have satisfactory results, which makes MLPs unsuited for processing images [5]. Figure 2.1 shows a weight matrix of a fully connected layer which is made for detecting vertical egdes. As can be seen, a lot of entries in the weight matrix are 0 and

are therefore not needed. This shows the inefficiency of MLPs when used on images. In this example, just sliding the kernel $[-1, 1]$ over the image is equivalent, but with much lower resource consumption. This is the principle behind convolutional neural networks (CNNs).

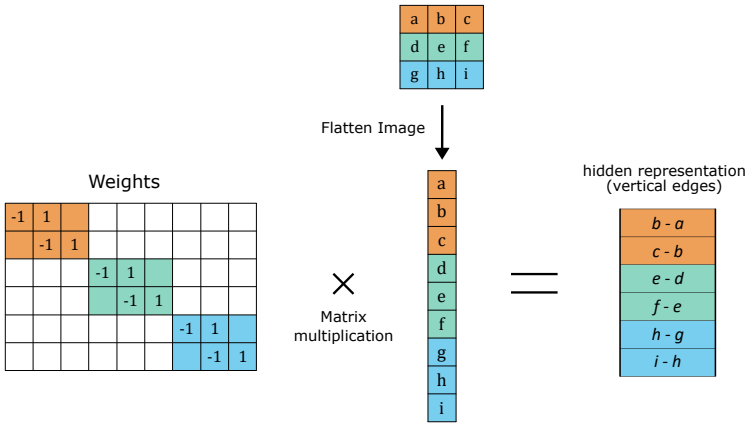


Figure 2.1: Weight matrix of the first a fully connected layer of an MLP designed to detect vertical edges. The empty cells have value 0. The bias vector is not shown. Image created by the author, based on the lectures of the Machine Learning for Computer Vision course of prof. Samuele Salti.

2.2.1 Translation equivariance and locality

Images have the handy characteristics that it does not really matter where the object of interest is exactly located for it to be recognizable. This means that the the neural network should respond similarly to the same patch, regardless of its specific location within the image. Additionally, the network response sould shift accordingly. This principle is called *translation equivariance*, and is shown clearly in Figure 2.2. In addition, the earliest layers should also mainly focus on local regions, ignoring the more distant ones. This is the *locality* principle. Eventually, these local representations can be aggregated to make predictions on the whole image.

We want to exploit the spatial structure of the images to apply the princi-

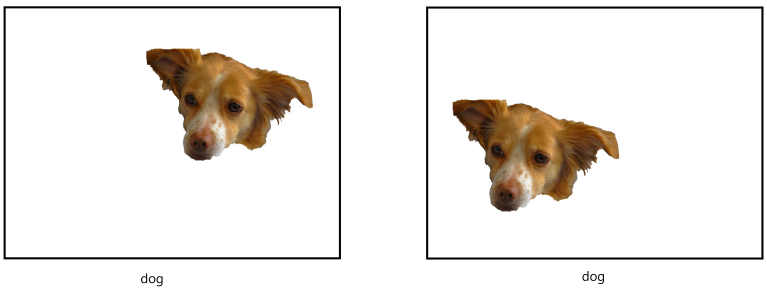


Figure 2.2: Translation equivariance. No matter where the dog is placed, the system recognizes it in the same way.

ples of translation equivariance and locality. We start by re-indexing the terms in equation (2.1) to have an unflattened representation of the images:

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_{k}\sum_{l} [\mathsf{W}]_{i,j,k,l}\, [\mathbf{X}]_{k,l}. \tag{2.2}$$

Now $\mathbf{X}$ is the (unflattened) input image, $\mathbf{H}$ its hidden representation, W the weights and $\mathbf{U}$ the biases. Note that with this new indexing the weigh matrix $\mathbf{W}$ becomes a fourth-order tensor W. This apparently cosmetic change is important, because neighboring indices now represent neighboring locations in the image.

Translation equivariance

To apply translation equivariance, it is convenient to re-index the weight tensor W as follows:

$$[\mathbf{H}]_{i,j} = [\mathbf{U}]_{i,j} + \sum_{a}\sum_{b} [\mathsf{V}]_{i,j,a,b}\, [\mathbf{X}]_{i+a,j+b}, \tag{2.3}$$

where $[\mathsf{V}]_{i,j,a,b} = [\mathsf{W}]_{i,j,i+a,j+b}$. The range of the indices a and b is now over positive and negative offsets, so that $i + a$ and $j + b$ span the entire image. To calculate the value of a specific hidden location $[\mathbf{H}]_{i,j}$, we perform a summation over the corresponding pixels in the input image $\mathbf{X}$ centered around

(i, j), where each pixel is weighted by the corresponding value in $[\mathsf{V}]_{i,j,a,b}$.

Since we want translation equivariance, a shift in the input $\mathbf{X}$ should simply shift $\mathbf{H}$ by the same amount. This means that V and $\mathbf{U}$ must not depend on the spatial location (i, j):

$$[\mathsf{V}]_{i,j,a,b} = [\mathbf{V}]_{a,b}$$
$$[\mathbf{U}]_{i,j} = u.$$

Equation (2.3) threfore becomes:

$$[\mathbf{H}]_{i,j} = u + \sum_a \sum_b [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}. \tag{2.4}$$

Ignoring the bias u, this operation is a *cross-correlation* between the matrices $\mathbf{V}$ and $\mathbf{X}$. In the field of Deep Learning it is always referred to as *convolution*. However, to formally be a convolution, one should use $(i-a, j-b)$ instead of $(i+a, j+b)$ when indexing $\mathbf{X}$. This misnomer is in fact practically irrelevant, as the two notations can be matched by simply flipping $\mathbf{V}$ and the signs of a and b.

Note that $[\mathbf{V}]_{a,b}$ has much fewer parameters than $[\mathsf{V}]_{i,j,a,b}$. An illustration of this operation is shown in figure 2.3.

Locality

As mentioned before, the locality principle states that locations far away from (i, j) have no effect on the hidden representation. Therefore, we can impose $[\mathbf{V}]_{a,b} = 0$ for any value $|a| > \Delta$ and $|b| > \Delta$. We can rewrite $[\mathbf{H}]_{i,j}$ as:

$$[\mathbf{H}]_{i,j} = u + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{V}]_{a,b} [\mathbf{X}]_{i+a,j+b}. \tag{2.5}$$

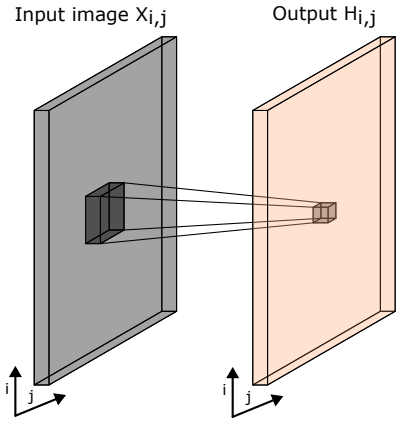


Figure 2.3: Illustration of a convolutional layer as in equation (2.4). The kernel is the small dark square, and the bias is not shown. The indices (i, j) refer to the spatial dimensions, along which the kernel is slid. Image created by the author. Image created by the author, based on the lectures of the Machine Learning for Computer Vision course of prof. Samuele Salti.

This equation describes a *convolutional layer*. The matrix $\mathbf{V}$ is called *convolutional kernel*, *filter* or simply as the layer *weights*. Together with the bias, it is usually a learnable parameter.

2.2.2 Channels

In the definition given in (2.5) assumes the image $\mathbf{X}$ to be grayscale. This is a major limitation for 2 main reasons:

- images are usually RGB and thus have 3 channels;
- only one hidden representation can be generated.

To solve the first problem, it is sufficient to add a channel dimension to $\mathbf{X}$, rendering it a third-order tensor. This means that we index it as $[\mathsf{X}]_{i,j,k}$. Accordingly, the convolutional kernel will go from $[\mathbf{V}]_{a,b}$ to $[\mathsf{V}]_{a,b,c}$. This is shown in Figure 2.4.

The second problem can be addressed similarly, i.e. by adding channel dimension to the hidden representation. This is shown in Figure 2.5. In this

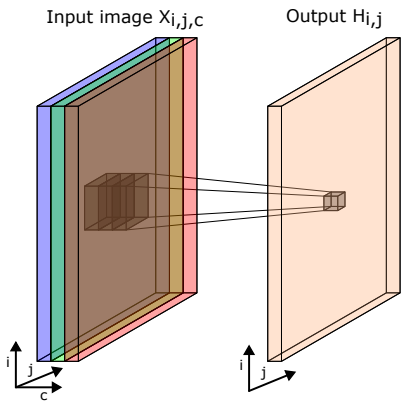


Figure 2.4: Illustration of a convolutional layer in which the input image has multiple channels. The kernel is the stack of darker squares, and bias is not shown. The indices (i, j) refer to the spatial dimensions, along which the kernel is slid. The index c refers to the channels of the input image. Image created by the author, based on the lectures of the Machine Learning for Computer Vision course of prof. Samuele Salti.

way we go from $[\mathbf{H}]_{i,j}$ to $[\mathsf{H}]_{i,j,d}$. Each element along the channel dimension of H is called a *feature map*, and contains different representations of the input. For example, there could be a feature map for all the vertical edges, one for the horizontal edges, one for the corners and so on. We must also add this additional output channel dimension to the convolutional kernel, which therefore becomes a fourth-order tensor $[\mathsf{V}]_{a,b,c,d}$. The bias becomes a vector $\mathbf{u}_d$ indexed on the output channel dimension. The final convolutional layer for multiple channels therefore is:

$$[\mathsf{H}]_{i,j,d} = \mathbf{u}_d + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c} [\mathsf{V}]_{a,b,c,d} \, [\mathsf{X}]_{i+a,j+b,c}, \tag{2.6}$$

where (i, j) index the input and output spatial dimensions, and d indexes the output channel.

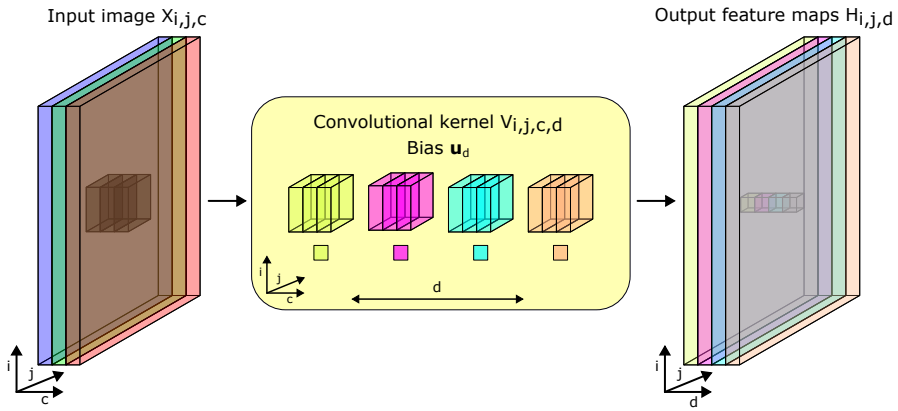


Figure 2.5: Illustration of a convolutional layer with 3 input channels and 4 output channels. The indices (i, j) refer to the spatial dimensions, c to the channels of the input image and d to the channels of the output image. Image created by the author, based on the lectures of the Machine Learning for Computer Vision course of prof. Samuele Salti.

3 State of the Art

This chapter covers the state of the art of ball tracking in ball sports are covered.

Since Padel is a relatively new sport, there is not so much research in this field. Other sports such as association football, basketball, volleyball and tennis there have more research going towards them.

3.1 Existing products for sports analysis

There are various commercial systems that perform ball tracking. The two most important ones are HawkEye and Clutch.

Hawk-Eye

The Hawk-Eye system [6] uses computer vision technology to track and visualize the trajectory of a ball in various sports, such as cricket, tennis, football, badminton, hurling, rugby union, association football, and volleyball.

The system uses multiple cameras, and calculates the 2D trajectory in the image space of each one. It then combines the 2D tracks of each camera and projects the tracks in 3D space using triangulation. Hawk-Eye operates on the principle of triangulation, using visual images and timing data obtained from a multitude of high-speed video cameras stationed around the playing field at various positions and angles.

The system also creates a graphic depiction of the ball's path and the playing area. This information is quickly made available to referees, television viewers, or coaching staff.

Clutch

The Clutch [7] system is made for badminton, and provides a service that can turn the phone to a badminton coach. It provides statistics of the game, with the length of the rally lengths, winner and mistakes of the players. It also employs a tracking system for the ball, which is necessary in order to get the game statistics.

4 System description

4.1 Overview

The system consists of two parts:

- ball detection: a convolutional neural network detects the position of the ball in each video frame;
- trajectory fitting: parabolas in image space are fitted across multiple frames and linked together. The procedure is explained more detail in Sections 4.3.

The source code can be found in the repository in [8].

4.2 Ball detection

The detection of the ball is performed by a Convolutional Neural Network (CNN). The goal is, given a sequence of n consecutive input frames, to extract the position of the ball in the last frame. Two main approaches have been tested:

- convolutional encoder architecture with a regression head, in the RegNetY design space [9]. It takes as input a frame sequence and outputs the (x, y) position of the ball in normalized pixel coordinates;
- convolutional encoder and decoder. Like before, it takes a frame sequence as input, but outputs a heatmap with the likelihood of finding the ball in a given position. In this variant, the output of the previous h

frame sequences is also saved as an internal state, and given as input to the network for the detection in the next frame.

The second architecture has proven to be the best performing one. More details about the results for each architecture with different hyperparameters are shown in more detail in chapter 5.

4.2.1 Architecture

The encoder-decoder architecture used for this project is a Unet-style network [3] with a VGG13 [10] encoder, called TrackNetV2 [2]. Its structure can be seen in Table 4.1 and Figure 4.1.

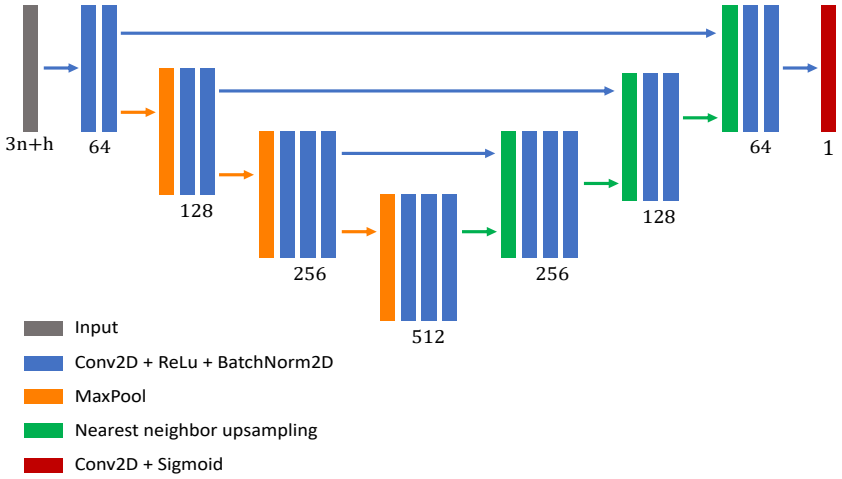


Figure 4.1: Architecture of TrackNetV2 with n input frames and h heatmaps of the previous position. The displayed number refers to the number of output channels in each convolutional layer. For the final system, $n = 2$ and $h = 0$ has been chosen. Image created by the author.

The model takes as input a sequence of n consecutive video frames and the heatmaps of the previous h detections, as previously mentioned. Since each video frame is an RGB image, the model's input consists of $3n + h$ channels. The output is a heatmap where a 2D Gaussian distribution is centered around

Table 4.1: Network Parameters of TrackNet. MaxPooling layers use a stride of 2, and upsampling layers upsamle by a factor of 2, using neighrest neighbor upsampling. Between the final convolution of each block and the upsampling layer with the same resolution there is a skip connection, where the activations are concatenated along the channel dimension.

Layer	Kernel Size	Depth	Resolution	Activation
`Input`	-	$3n + h$	360×640	-
`Conv1_1`	3×3	64	360×640	ReLU+BN
`Conv1_2`	3×3	64	360×640	ReLU+BN
`Pool1`	2×2	-	-	max pooling
`Conv2_1`	3×3	128	180×320	ReLU+BN
`Conv2_2`	3×3	128	180×320	ReLU+BN
`Pool2`	2×2	-	-	max pooling
`Conv3_1`	3×3	256	90×160	ReLU+BN
`Conv3_2`	3×3	256	90×160	ReLU+BN
`Conv3_3`	3×3	256	90×160	ReLU+BN
`Pool3`	2×2	-	-	max pooling
`Conv4_1`	3×3	512	45×80	ReLU+BN
`Conv4_2`	3×3	512	45×80	ReLU+BN
`Conv4_3`	3×3	512	45×80	ReLU+BN
`UpS1`	2×2	-	-	upsampling
`Conv5_1`	3×3	256	90×160	ReLU+BN
`Conv5_2`	3×3	256	90×160	ReLU+BN
`Conv5_3`	3×3	256	90×160	ReLU+BN
`UpS2`	2×2	-	-	upsampling
`Conv6_1`	3×3	128	180×320	ReLU+BN
`Conv6_2`	3×3	128	180×320	ReLU+BN
`UpS3`	2×2	-	-	upsampling
`Conv7_1`	3×3	64	360×640	ReLU+BN
`Conv7_2`	3×3	64	360×640	ReLU+BN
`Conv8`	1×1	1	360×640	Sigmoid

the ball position in the last frame of the input sequence. This approach enables multiple detections, if multiple balls are present.

For training, a dataset containing multiple videos is used, where each video is accompanied by labeled information indicating the ball's position in every frame. These labels are represented in normalized pixel coordinates. The dataset is split into a training set and a validation set. The training procedure follows a fixed schedule, rendering a labeled test set unnecessary. However, to assess the model's robustness, an unlabeled video is employed for testing purposes.

During training, two hyperparameters need to be set:

- The spatial resolution of the input;
- the width of the Gaussian on the target heatmap σ.

The input resolution is set to a fixed value of 360×640, and the width of the target Gaussian to $\sigma = 5\,\mathrm{px}$.

Further details regarding the training and validation data can be found in Section 5.1. Figure 4.2 provides an example visualizing the input data, target heatmap extracted from the validation set, and the corresponding output produced by the model.

4.2.2 Extracting the ball position from the heatmap

Given a heatmap, the ball position must be extracted by finding its local maxima. This enables to detect multiple balls. The process involves the following steps: First, a Gaussian blurring with a width of σ_s is applied to the heatmap to reduce noise, and the resulting heatmap is rescaled to its original value range. Next, each location in the blurred heatmap is compared to its neighboring ones, the locations greater than all of their neighbors are selected. This comparison

includes diagonal neighbors as well. Finally, among the identified local maxima, only those surpassing a threshold v_{th} are kept. Fixed values of $\sigma_s = 5\,\text{px}$ and $v_{th} = 0.1$ chosen.

Table 4.2: Values for the fixed detection parameters. The input resolution is 640×360.

Parameter	Value
input resolution	640×360
σ	5
σ_s	5
v_{th}	0.1

4.2.3 Results overview

The encoder-only architecture did not achieve satisfactory performances. For the TrackNetV2 architectures, variants with $h = 0$ and $n = 2$, 3, 4 and 6, and with $n = 1$ and $h = 3$ have been tested. The one with $h = 0$ and $n = 3$ performs best. The hyperparameters are summarized in Table 4.2. For $n = 3$ and $h = 0$, the detection rate on the validation set is 93.0%. Of the detected balls, the mean positioning error Δp_m is $1.42\,\text{px}$. The rate of positioning error $\Delta p < 1\,\text{px}$ is 49.1%, for $\Delta p < 5\,\text{px}$ it is 96.9%, and for $\Delta r < 10\,\text{px}$ it is 98.5%. An example of ball detection can be seen in Figure 4.2. A more detailed analysis of the results for each architecture can be seen in Section 5.2.2.

4.3 Trajectory fitting

4.3.1 Fitting a parabola to each frame

The section explains the fitting of parabolic trajectories to position candidates (which may include false positives) in each frame and is based on the paper

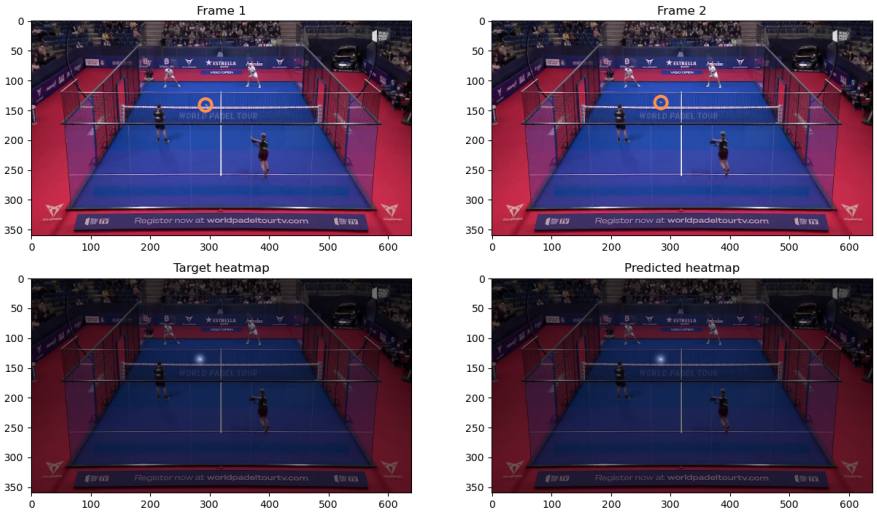


Figure 4.2: Example of a training sample for a sequence length of $n = 2$ frames and resolution 640×360. The upper images show the two input frames of the video. The lower left image shows the target heatmap superimposed on frame 2. The width of the Gaussian is $\sigma = 5\,\mathrm{px}$. The lower right image shows the output heatmap of the model superimposed on frame 2.

by Yan et al. [4].

The idea is to extract information about the ball trajectory, after having detected its position in each frame using a CNN. After choosing a sliding window centered on a seed frame k and detection candidate, a parabolic model in image space is fitted incrementally along it. The process works as follows:

1. detect all triplets of candidate detections in neighboring frames (*seed triplets*), so that the distance between candidates of neighboring frames is below a certain threshold distance r;

2. for each seed triplet fit a parabolic model, and compute the estimated positions;

3. find the supports and the cost of the model. The supports are the position candidates close enough to the estimated positions;

4. fit a new parabolic model using the supports of the previous model, and

compute the new estimated positions;

5. find the supports and the cost of the new model. If the number of supports does not increase or if the cost increases, keep the previous model. Otherwise, go back to point 4 and use the new model as the starting point;

6. after having found a parabolic model for each seed triplet, filter implausible models out and choose the one with the lowest cost.

A parabolic model is chosen, because it requires only 3 parameters and is a good approximation of the true ball path, which is assumed to have a constant acceleration due to gravity, air drag and ball spin. This incremental approach is useful for the following two reasons:

- using only the seed triplet for fitting yields to poor results, as the relative error of the detection position in neighboring frames is higher than in frames that are further away in time;

- using frames further in time reduces the relative error, of the fitted model, but since each path has a different length, a criterion for finding the furthest possible frames is necessary.

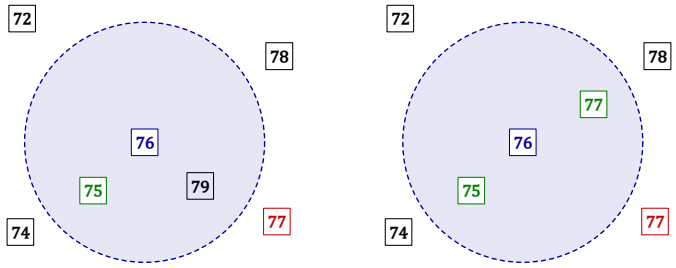
In the process described above, one parabolic model per frame is fitted. In order to describe the final path of the ball, which might include bounces on the ground and hits by the players, a strategy for connecting trajectories is needed. This is further explained in 4.3.2.

Detection of the seed triplets

As previously mentioned, we already have position candidates for each frame. The set of candidates in frame k is:

$$\mathcal{C}_k = \{c_k^j\}, \quad j \in \{\, 1, \ldots, m_k \,\}, \tag{4.1}$$

where, m_k is the number of candidates in frame k and c_k^j is the j^{th} candidate in $\mathcal{C}_k$. The sliding window has size $2N+1$ and moves across the video sequence. In this work, $N = 10$ is chosen.



(a) **(b)**

Figure 4.3: The squares denote the position candidates c_k^j. The number inside them denotes the frame k in which it belongs to. Here we consider a sliding window centered on frame $k = 76$. A candidate $c_{76}^j \in \mathcal{C}_{76}$ is highlighted in blue. The area A_{76}^j is highlighted in blue, and is surrounded by the dashed circle of radius r. In (a) no seed triplet is found: one candidate in $\mathcal{C}_{75}$ falls in A_{76}^j (green), but no candidate in $\mathcal{C}_{77}$ does (red). There is one candidate from $\mathcal{C}_{79}$ within A_{76}^j (black), but it is too far in time to be considered for a seed triplet. In (b) a seed triplet is found, because there are candidates in both $\mathcal{C}_{75}$ and $\mathcal{C}_{77}$ that fall within A_{76}^j (green). Image inspired by [4, Fig. 1].

A seed triplet consists of position candidates in neighboring frames $k-1$, k and $k+1$, so that the distance between each neighboring candidate is less than a specific threshold distance r. At time k, the interval I_k covers frames from $k-N$ to $k+N$. Within interval I_k, we focus on each c_k^j, centered at the candidate and with a radius r. We define a circular area A_k^j that encompasses this region. We examine whether there exists at least one candidate from C_{k-1} and at least one candidate from C_{k+1} that fall within A_k^j (see Figure 4.3). Let's assume $c_{k-1}^j \in \mathcal{C}_{k-1}$ and $c_{k+1}^j \in \mathcal{C}_{k+1}$ are located inside A_k^j. If this condition is met, we can utilize c_{k-1}^j, c_k^j, and c_{k+1}^j as a seed triplet to solve a constant acceleration motion model (see Figure 4.3b).

Fitting a parabolic model to a candidate triplet

Given three position candidates at the frame indices k_0, k_1 and k_2, the parabolic model that needs to be fitted is the following:

$$\mathbf{a} = 2\,\frac{\Delta k_{0,1}\,(\mathbf{p_2} - \mathbf{p_1}) - \Delta k_{2,1}\,(\mathbf{p_1} - \mathbf{p_0})}{\Delta k_{1,0}\;\Delta k_{2,1}\;(\Delta k_{1,0} + \Delta k_{2,1})} \tag{4.2}$$

$$\mathbf{v_0} = \frac{\mathbf{p_1} - \mathbf{p_0}}{\Delta k_{1,0}} - \frac{a}{2}\Delta k_{1,0}, \tag{4.3}$$

where $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ are the candidate positions at k_0, k_1 and k_2, $\Delta k_{2,1} = k_2 - k_1$, $\Delta k_{1,0} = k_1 - k_0$, $\mathbf{a}$ is the acceleration and $\mathbf{v_0}$ is the velocity at time k_0.
The estimated position at any time k can be calculated as:

$$\hat{\mathbf{p}}_\mathbf{k} = \mathbf{p_0} + \Delta k\;\mathbf{v_0} + \frac{\mathbf{a}}{2}\;\Delta k^2, \tag{4.4}$$

where $\Delta k = k - k_0$. This can be used to calculate the trajectory along the whole sliding window.

Finding the supports of a model

A position candidate c_k^j is said to be a support of a model if it is consistent with it. This is the case, if the distance between the candidate c_k^j and the position estimated by the model is smaller than a threshold distance d_{th}:

$$d(\hat{\mathbf{p}}_k, \mathbf{p}_k^j) < d_{th}. \tag{4.5}$$

This is shown in Figure 4.4. The set of all supports of a model is referred to as $\mathcal{S}$. Given a set of supports, we use it to find the frames and position candidates

to use for fitting the next model. We define:

$$k_{min} = \min k \qquad \forall c_k^j \in \mathcal{S} \tag{4.6}$$

$$k_{max} = \max k \qquad \forall c_k^j \in \mathcal{S} \tag{4.7}$$

$$k_{mid} = \arg\min_k ||k_{max} - k| - |k_{min} - k|| \qquad \forall c_k^j \in \mathcal{S}, \tag{4.8}$$

where k_{min} is the first frame used for the new fit, k_{mid} the middle one and k_{max} the last one. In Figure 4.4a, the support set encompasses frames 74 to 78. According to the previous equations, in this case $k_{min} = 74$, $k_{mid} = 76$ and $k_{max} = 78$. These frames are then used to fit a new model using equations (4.2) and (4.3).

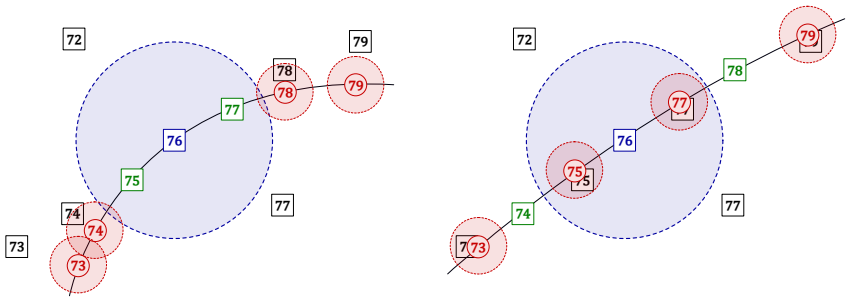


Figure 4.4: The squares represent position candidates in various frames, and the circles the estimated positions. In (a), a parabolic model is applied to the seed triplet, represented by the blue and green squares. The solid red circles indicate the positions estimated by the fitted model. The dashed red circles have radius d_{th} and are used to determine the support of the fitted model. The support set encompasses frames 74 to 78. In (b), a second model is fitted based on the support obtained from the model in (a). Image inspired by [4, Fig. 2, Fig. 3].

Computing the cost of a model

The cost function of a trajectory fitted on a model is given by:

$$C = \sum_{k=i-N}^{i+N} \sum_{j} \min\left(d_{th}^2, d^2(\hat{\mathbf{p}}_\mathbf{k}, \mathbf{p}_\mathbf{k}^\mathbf{j})\right), \tag{4.9}$$

where d_{th} is the threshold distance mentioned before, $\hat{\mathbf{p}}_\mathbf{k}$ the position estimated by the model, $\mathbf{p}_\mathbf{k}^\mathbf{j}$ the positions of the candidates and $d(\cdot,\cdot)$ the euclidean distance.

Filtering and extraction of the final model

After finding the optimized model for each seed triplet in a frame k, it is necessary to choose the best fitting one. This is done by first applying some filtering steps, and then choosing the trajectory with the lowest cost. There are three filtering steps, not present in [4]:

- first, the models with an acceleration vector pointing upwards are filtered out. This makes sense, because the ball is subjected to the force of gravity. Since the models are computed in the image reference frame, with the origin in the upper left corner, only the models with $\mathbf{a}_y \geq 0$ are kept;

- a second filtering step is that the magnitude of the acceleration must have a minimum magnitude a_{min}. The reason is the possible presence of stationary balls on the floor. In that case, a parabola with $a \approx 0$ and $v_0 \approx 0$ would be fitted, and would have a very low cost. This is undesirable, therefore such trajectories are filtered out;

- the third and final filtering step imposes that the angle of the acceleration vector with respect to the vertical must not be too large. This is

necessary to filter out wall bounces, for which a parabola with a horizontal acceleration vector might actually be the best fit. Therefore, only accelerations with the ratio of the horizontal to vertical component ${}^{a_x}/_{a_y} \leq g$ are kept. This parametrizes the maximum lateral acceleration due to ball spin.

After these filtering steps, the trajectory with the lowest cost is picked for frame k.

4.3.2 Linking trajectories

A trajectory $\mathcal{T} = \{ \mathcal{M}, \mathcal{S} \}$ is defined as the union of a parametrized model $\mathcal{M}$ and its support set $\mathcal{S}$. Due to the optimization step explained earlier, each frame k will have exactly one trajectory, which will be referred to as $\mathcal{T}_k = \{ \mathcal{M}_k, \mathcal{S}_k \}$.

The goal of this section is to explain how to link multiple trajectories together. If two frames a and b are far enough in time, there might be bounces and hits between them. In addition, a fitted trajectory might not exactly span the whole flying path of the ball due to approximations of the constant acceleration model. For this reason, it is necessary to find a way to link multiple trajectories together in order to correctly describe the path of the ball. In order to do this, a weighted directed graph, referred to as *trajectory graph*, is built. The edges connect earlier trajectories to later ones, and the weights are given by a specific distance function.

Distance between trajectories

In order to define a proper distance function, the concepts of overlapping and conflicting trajectories must be defined first.

Two trajectories $\mathcal{T}_a$ and $\mathcal{T}_b$, $a < b$, are said to be *overlapping* if $k_{max,a} \geq k_{min,b}$, meaning that the end of the first trajectory occurs at the same time or

after the beginning of the second one. Two overlapping trajectories are said to be *conflicting*, if $\exists k \in \{ k_{min,b}, \ldots, k_{max,a} \}$ such that:

$$\begin{aligned} &(\exists c'_k \in \mathcal{S}_a \quad \wedge \quad \nexists c''_k \in \mathcal{S}_b) \quad \vee \\ &(\exists c'_k \in \mathcal{S}_b \quad \wedge \quad \nexists c''_k \in \mathcal{S}_a) \quad \vee \\ &(\exists c'_k \in \mathcal{S}_a \quad \wedge \quad \exists c''_k \in \mathcal{S}_b \quad \wedge \quad \mathbf{p}'_k \neq \mathbf{p}''_k). \end{aligned} \tag{4.10}$$

This means that two trajectories are conflicting if a support of one trajectory is not present in the other one or if two supports of the same frame have different positions. Now we can properly define the trajectory distance. If $\mathcal{T}_a$ and $\mathcal{T}_b$ are overlapping, their distance is:

$$D(\mathcal{T}_a, \mathcal{T}_b) = \begin{cases} 0 & \text{if } \mathcal{T}_a \text{ and } \mathcal{T}_b \text{ are not conflicting;} \\ \infty & \text{if } \mathcal{T}_a \text{ and } \mathcal{T}_b \text{ are conflicting.} \end{cases} \tag{4.11}$$

If they are non-overlapping, their distance is given by:

$$D(\mathcal{T}_a, \mathcal{T}_b) = \min d(\hat{\mathbf{p}}_{k,a}, \hat{\mathbf{p}}_{k,b}), \quad k_{max,a} \leq k \leq k_{min,b}, \tag{4.12}$$

where $\hat{\mathbf{p}}_{k,a}$ and $\hat{\mathbf{p}}_{k,b}$ are the estimated positions for trajectories $\mathcal{T}_a$ and $\mathcal{T}_b$ respectively.

Building the trajectory graph

Using the trajectory distance defined in equations (4.11) and (4.12), a graph connecting each trajectory can be built, referred to as the *trajectory graph*, as shown in Figure 4.5. Since only candidates with a time difference of at most N are considered, for each trajectory $\mathcal{T}_k$, the distances between it and the ones in the range $\{ k+1, \ldots, k+N \}$ are computed. Each trajectory then constitutes a node in the trajectory graph, with outgoing edges to trajectories later in time

and weights equal to the distances between them. If the two trajectories are conflicting (their distance is infinite), no edge is added.

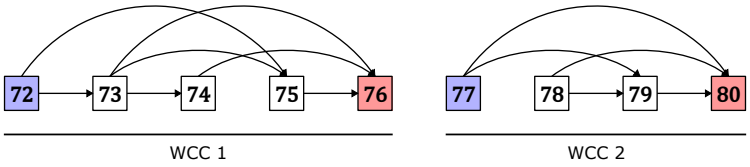


Figure 4.5: Example of a trajectory graph with 2 weakly connected components (WCC). The weight of each edge corresponds to the distance of the nodes it connects. Dijkstra's algorithm is used on each weakly connected component to find the shortest path between its first and last nodes, shown in blue and red respectively. In this example trajectories 74-75 and 77-78 are conflicting: they have infinite distance and have therefore no edge between them. Image created by the author.

In order to find the best fitting path of the ball, Dijkstra's algorithm is used. This might pose an issue, because there might be disconnected components in the trajectory graph (for instance, if the game is paused, or if the ball goes out of frame, no trajectory will be found). Therefore, all the weakly connected components (WCCs) in the trajectory graph are found first, and Dijkstra's algorithm is then applied between the first and last trajectory of each connected component. This partly diverges from the one described in [4], in which no mention of weakly connected components is made.

The result is a sequence of trajectories for each WCC.

Mapping each frame to a trajectory

After having found a trajectory sequence by applying Dijkstra's algorithm to the trajectory graph, the final mapping between frames and trajectories is found. For a frame k, a mapping is possible if there exists a trajectory $\mathcal{T}_l$ such that $l_{min} \leq k \leq l_{max}$. The set of possible trajectories is $\{ \mathcal{T}_l \}$. If a mapping is possible, the trajectory $\mathcal{T}_a$ assigned to frame k will be the trajectory in $\{ \mathcal{T}_l \}$

such that:

$$a = \arg\min_{l}(k - l_{min}) \quad \forall\, \mathcal{T}_l \in \{\mathcal{T}_l\}. \tag{4.13}$$

After this step, a trajectory can be uniquely assigned to a frame. This makes it possible to estimate parameters such as the ball position and velocity in image space at a given time step. Estimating the true trajectory in 3D space would require triangulation from detections of different cameras angles. Since a model for the ball path is obtained, the position in image space between frames can be interpolated. The synchronization of the cameras at each frame becomes therefore less critical.

4.3.3 Results overview

Examples of a successful and failed fit can be seen in Figure 4.6. Shots by both the near and far team and bounces on the floor and on the walls are successfully detected. A more thorough evaluation of the fitting results can be seen in Section 5.3.

(a) **(b)**

Figure 4.6: Examples of trajectory fits are illustrated in (a) and (b). In (a), a successful trajectory fit is obtained, while in (b), the fit fails. The neural network's heatmap is superimposed on the frame, and the network's detection candidates are indicated as red dots. The past, current, and future trajectories are represented by the orange, white, and green lines, respectively. The white boxes correspond to the k_{min} and k_{max} values of the fitted trajectory. The table in the upper left displays frame index k, estimated position p and velocity v, along with fit parameters p_0, v_0, and a, given in units of pixels, pixels per frame, and pixels per frame squared, respectively. Trajectory fitting is performed using the following parameters: $N = 10$, $r = 40$, $d_{th} = 10$, $a_{min} = 0.2$, and $g = 2$, with a video resolution of 1280×720 pixels.

5 Results

In this chapter the results of the ball detection are discussed in more detail. In particular, all the variants of the detector that have been tried are shown, and an analysis of successes and failures of the trajectory fitting in different scenarios are shown. An explanation of the training data is also provided.

5.1 Dataset

The dataset was partly provided by my supervisor and partly self-labelled. It consists of six videos with a resolution of 1280×720, for a total of 9947 labelled frames. An overview of the sources of the data is shown in Table 5.1.

The labels are saved as CSV files, which contain the frame index and the position of the ball in normalized pixel coordinates. The reason for the normalization of the pixel coordinates is for making implementation easier, as the video is scaled to a 640×360 resolution before being given as input to the detector.

The model was also tested qualitatively on various freely available videos of World Padel Tour matches.

Implementation details

The system is implemented in Python [12], version 3.9.7. The detector is written in PyTorch [1], version 1.12.1. The trajectory fitting is also implemented in Python, using the Numba JIT compiler [13], version 0.56.4.

Table 5.1: Dataset overview. All videos come from the World Padel Tour and are at 25 fps. The 2020 video is self-labelled, and is split between train and validation set, and the others were labelled as a part of a previous project [11].

Source event	frames	split
2022 Vigo Open Men's final	1456	train
2022 Estrella Damm Master 2022 Women's Final	1959	train
2021 Estrella Damm Master Men's Final	1492	train
2022 Circus Brussels Padel Open Men's Semifinal	1445	train
2020 Estrella Damm Open Women's Final	748	train
2022 Estrella Damm Open Women's Final	2690	validation
2020 Estrella Damm Open Women's Final	157	validation

5.2 Ablation study on the ball detection

Eight different configurations of models and training procedures have been tested. A combination of model architecture, hyperparameters and training procedure is referred to as *configuration*. The configurations are shown in Table 5.2. The ones with "tracknet" in their names use the TrackNetV2 architecture, as explained in Section 4.2. The other ones use the RegNetY design space, as explained in [9], and are pre-trained on ImageNet [14]. Before being given as input to the models, the video is scaled to a 640×360 resolution.

5.2.1 Training procedures

All the configurations use the Adam optimizer [15]. The parameters of the optimizer are set to the default ones: $lr = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. Only in tracknet_v2_rnn_s the learning rate is decreased during training. The loss function is the mean square error. The number of epochs and batch size are shown in Table 5.2. The loss histories are shown in Figure 5.1.

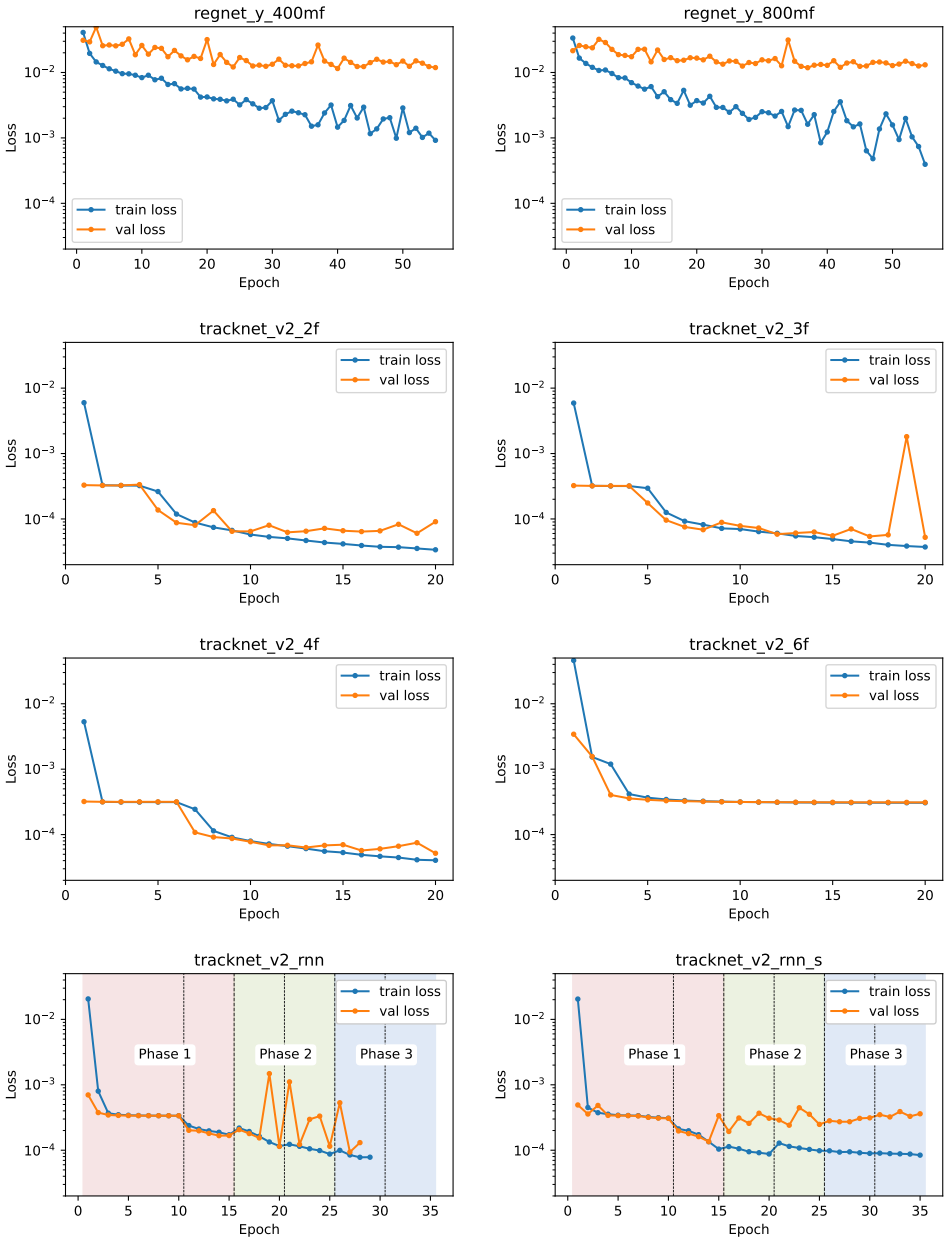


Figure 5.1: Loss histories for all the configurations. The training phases for tracknet_v2_rnn and tracknet_v2_rnn_s are explained below.

Table 5.2: Table with the training hyperparameters for the various configurations. The number of input frames is n, the number of memorized previous states is h. The Adam optimizer with the default values is used in all cases, except for tracknet_v2_rnn_s, where the learning rate is decreased during training.

configuration	n	h	epochs	batch size
regnet_y_400mf	3	0	40	16
regnet_y_800mf	3	0	40	16
tracknet_v2_2f	2	0	20	8
tracknet_v2_3f	3	0	20	8
tracknet_v2_4f	4	0	20	8
tracknet_v2_6f	6	0	20	8
tracknet_v2_rnn	1	3	35	8
tracknet_v2_rnn_s	1	3	35	8

Training procedure for $h \neq 0$

The training procedure is a bit more elaborated for the architectures with $h \neq 0$, i.e. tracknet_v2_rnn and tracknet_v2_rnn_s configurations. Previously, the training data was shuffled at each training epoch. Since now the model gets as input the output of the previous frame, the video is given as input sequentially. To handle batch sizes greater than 1, the parallel sequences are started in different points in the video.

Training is divided in 3 phases. Before explaining them, some concepts must be defined:

- *Total clearing*: when total clearing is performed, the memorized internal states are set to zero. This forces the model to detect the ball using only the input frame. At inference time, this corresponds to the situation at the start of a video sequence. The probability of performing total clearing in a training step is indicated as $p(tc)$;

- *Partial clearing*: when partial clearing is performed, each memorized

heatmap is cleared with a probability of $1/h$, where h is the number of input heatmaps. This forces the model to work with a missed detection in the past frames. The probability of performing partial clearing is indicated as $p(pc)$;

- *Ground truth input*: this is pretty self-explanatory. With ground truth input, the model is given ground truth heatmaps as input instead of its own previous outputs. The probability of having ground truth input is indicated as $p(gt)$. The sampling of $p(gt)$ is done independently for each input heatmap, so there can be a mix of ground truth and previous heatmaps as input to the model.

These actions are only performed at training time. Having defined the concepts of *total clearing*, *partial clearing* and *ground truth input*, the training phases can be defined:

1. Ground truth regime. In this phase, the model must learn to detect the ball from the input frames only, without having to rely necessarily on the previous detection. This is necessary to correctly initialize the model and make it capable of making the first detection of the ball. Initially $p(gt)$ is set to 1, and high values for $p(tc)$ and $p(pc)$ are used.

2. Hybrid regime. This is the transition phase from the ground truth regime to the RNN regime. Here, $p(tc)$ is set to 0, and intermediate values for $p(pc)$ and $p(gt)$;

3. RNN regime: here the model is taught to use its own previous outputs as input, therefore very low values of $p(pc)$ and $p(gt)$ are used.

Each training phase is divided in 2 sub-phases, in which the values for $p(tc)$, $p(pc)$ and $p(gt)$ are set closer to the ones of next phase. The exact values for each phase and sub-phase are shown in Table 5.3.

The only difference between tracknet_v2_rnn and tracknet_v2_rnn_s is that in the former case the learning rate is kept constant, and in the latter case it is decreased at each training phase. This is done, because the loss starts spiking in phase 2 of tracknet_v2_rnn, and explodes completely in phase 3, as shown in Figure 5.1. While lowering the learning rate stabilizes the loss, it ultimately leads to overfitting.

Table 5.3: Table with the training procedure values for tracknet_v2_rnn and for tracknet_v2_rnn_s. The lr column holds true only for tracknet_v2_rnn_s. For tracknet_v2_rnn the learning rate is always 10^{-3}.

Phase		End Epoch	$p(tc)$	$p(pc)$	$p(gt)$	lr
1	1.1	10	0.8	0.1	1	10^{-3}
	1.2	15	0	0.5	0.8	
2	2.1	20	0	0.33	0.5	10^{-4}
	2.2	25	0	0.2	0.2	
3	3.1	30	0	$1/30$	$1/30$	10^{-5}
	3.2	35	0	$1/200$	0	

5.2.2 Detection results

Table 5.4 shows the results of all the configurations on the validation set. The histograms of the positioning error distributions are shown in Figure 5.3. The positioning error is defined as the magnitude of the difference between the ground truth positions and the positions estimated by the detector, calculated on the 640×360 input resolution.

Both RegNetY architectures start overfitting after about 30 epochs, as shown in Figure 5.1. Due to their design, even if the ball is completely occluded in all the input frames, they will try to predict its position. This is a major limitation of their design. In addition, the mean positioning error is quite high at $24.6\,\mathrm{px}$ and $25.8\,\mathrm{px}$ for regnet_y_400mf and regnet_y_800mf respectively.

Among the two training schedules of TrackNetV2 with $h = 3$ and $n = 1$, tracknet_v2_rnn starts overfitting in Phase 3, and tracknet_v2_rnn_s even earlier in Phase 2. This is reflected on the validation set results, in which the mean positioning errors are $9.42\,\mathrm{px}$ and $37.5\,\mathrm{px}$ respectively. The detection rate is 92.7% and 99.4% respectively. The latter result is the highest of all the configurations, but it results from a high number of false detections, as shown in Figure 5.2b.

The TrackNetV2 architecture with $h = 0$ and $n = 6$ is not able to recognize the ball at all. Its output always consists of a heatmap containing only values very close to zero. This can also be seen in the loss function, which remains constant from epoch 6 onwards (see Figure 5.1). A heatmap containing only zeros represents a local minimum of the loss landscape, and the optimizer is not able to exit out of it. This situation happens in the training of the configurations with $n = 2, 3, 4$ as well, but here the optimizer eventually exits from that local minimum, which does not happen for $n = 6$. An example of output from this architecture variant can be seen in Figure 5.2c.

The TrackNetV2 configurations with $h = 0$ and $n = 2, 3, 4$ have the best results, and they all perform quite similarly. The configuration with $n = 3$ has a detection rate of 93.0%, slightly higher than the 92.0% of the other two. The mean positioning error is also the smallest among all configurations, having a value of $1.41\,\mathrm{px}$. However, the configurations with $n = 2$ and $n = 4$ are close, at $1.42\,\mathrm{px}$ and $1.53\,\mathrm{px}$ respectively. The percentage of positioning errors smaller than $1\,\mathrm{px}$ (i.e. exact detections) is also the highest for $n = 3$, at 49.1%. All configurations have very high values for the percentage of positioning errors smaller than $5\,\mathrm{px}$ and $10\,\mathrm{px}$.

Looking at the loss history of the configuration with $n = 2$, the training and validation loss start diverging after epoch 12. From there on, the training loss continues to drop, and the validation loss stays stable around $6 \cdot 10^{-4}$.

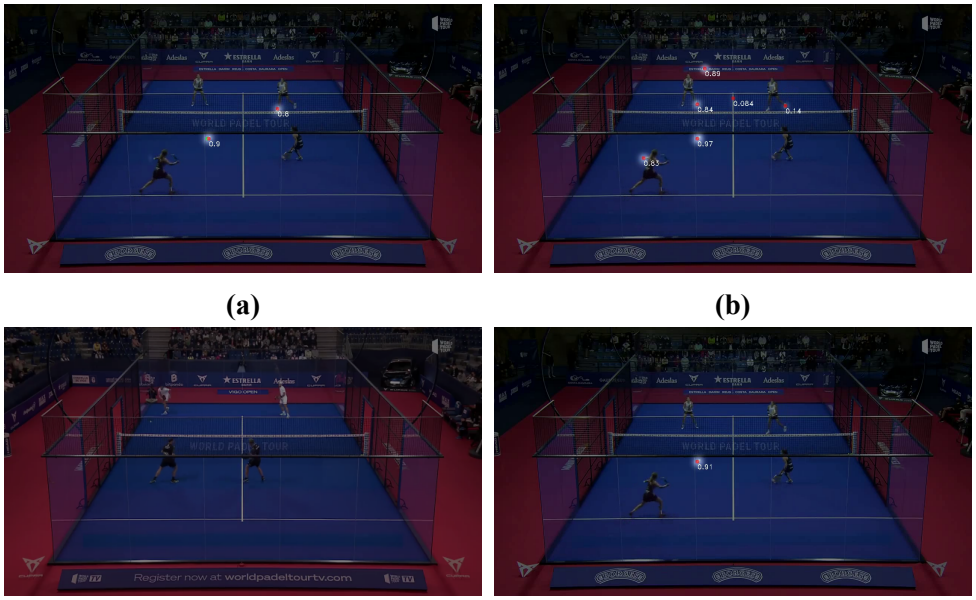


Figure 5.2: These example heatmaps are generated by different configurations of TrackNetV2. In (a) and (b), the configurations correspond to $n = 1$ and $h = 3$. Specifically, (a) represents tracknet_v2_rnn, and (b) represents tracknet_v2_rnn_s, which uses a different training schedule. In (c), the configuration is set to $n = 6$ and $h = 0$. Finally, (d) shows the result of the configuration with $n = 3$ and $h = 0$. The heatmaps for $n = 2$ and $n = 4$ show a high degree of similarity.

From epoch 18 on, the loss seems to start fluctuating, which indicates that the network is close to overfitting. In the loss history of the configuration with $n = 3$ the validation loss and the training loss do not diverge significantly until epoch 18. There is then a large spike in the validation loss at epoch 19. The spike then disappears at epoch 20. This indicates that the optimizer is close to a narrow local minimum of the loss landscape, and that the model is close to overfitting as well. In the loss history of the configuration with $n = 4$, the training and validation loss both decrease for the whole duration of the training process.

Figure 5.2d shows examples of output heatmaps for the various TrackNetV2 configurations on a sample frame of the validation set.

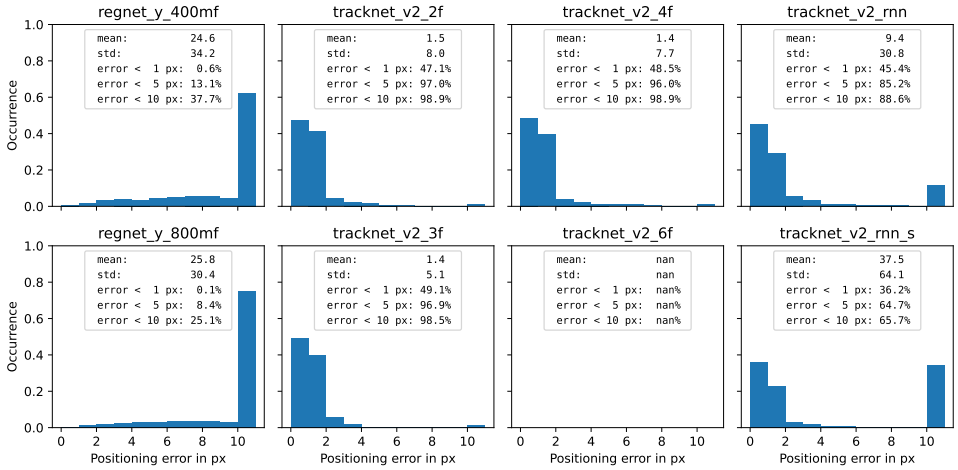


Figure 5.3: The distributions of positioning errors for all configurations are depicted. These errors are calculated based on a 640×360 frame, and only consider actual detections. The histograms provide a comprehensive overview of the entire error distribution, which is further summarized in Table 5.4.

Table 5.4: This table presents the results for all tested configurations. The detection rate of regnet_y_400mf and regnet_y_800mf is intentionally omitted, because it is 1 due to their design. For the TrackNetV2 configurations, the detection threshold v_{th} is set to 0.1, as specified in Section 4.2.2. The positioning error, expressed in pixels, is calculated based on a 640×360 frame and is calculated exclusively on the frames where the ball was detected. The results for tracknet_v2_6f show n/a as the model never detects the ball.

		Positioning error				
configuration	detection rate	mean	std	< 1 px	< 5 px	< 10 px
regnet_y_400mf		24.6	34.2	0.591%	13.1%	37.7%
regnet_y_800mf		25.8	30.4	0.148%	8.38%	25.1%
tracknet_v2_rnn	92.7%	9.42	30.8	45.4%	85.2%	88.6%
tracknet_v2_rnn_s	**99.4%**	37.5	64.1	36.2%	64.7%	65.7%
tracknet_v2_2f	92.2%	1.53	7.98	47.1%	**97.0%**	**98.9**%
tracknet_v2_3f	93.0%	**1.41**	**5.14**	**49.1**%	96.9%	98.5%
tracknet_v2_4f	92.2%	1.42	7.67	48.5%	96%	**98.9**%
tracknet_v2_6f	n/a	n/a	n/a	n/a	n/a	n/a

5.3 Analysis of the trajectory fitting

The detector chosen to extract the position candidates for the analysis of the trajectory fitting is TrackNetV2 with $n = 3$ and $h = 0$. On the position candidates found by the detector, parabolic trajectories are fitted as explained in Section 4.3.

5.3.1 Choosing the right fitting parameters

The right choice of the fitting parameters is crucial. The parabolic model of the true flight path of the ball is an approximation, as it does not take into consideration the fact that force exerted by air drag is dependent of the ball velocity, or that ball spin does not provide a constant acceleration. For instance, choosing a value too small for d_{th} might make the fit miss some supports due to the aforementioned approximations, but choosing it too large could lead to wrong fits, especially during wall bounces or shots made by the far team. Also, choosing a value for N might lead to similar results.

The finally chosen fit parameters shown in Table 5.5 provide satisfactory results. As a reminder, they are chosen on a video of size 1280×720 with a frame rate of 25 fps.

5.3.2 Analysis of different game situations

The trajectory fitting explained in Section 4.3 is useful to extract the velocity, position and acceleration of the ball in image space. This can be used to analyze the speed of player shots, as it corresponds to the v_0 parameter of the parabola fitted on the shot. An example of this can be seen in Figure 5.4. The fitted parabolas start the moment the players do a shot. Bounces on the floor and on the wall are detected as well, as shown in Figure 5.4a.

Sometimes, the fitting of the parabolas fails. This can happen whenever

Table 5.5: Parameter values used in the trajectory fitting process are displayed in the table. The video resolution is 1280×720 pixels. The units for N, r, and d_{th} are pixels, a_{min} is pixels per frame squared, and g is dimensionless. N indicates the number of frames considered before and after the seed frame, r represents the maximum radius considered for the initial seed triplet, d_{th} denotes the threshold between the estimated position by the fit and the nearest position candidates, a_{min} signifies the minimum magnitude of the fitted acceleration, and g represents the maximum ratio between the horizontal and vertical components of the fitted acceleration. For a more comprehensive explanation, please refer to Section 4.3.

Parameter	Value
N	10
r	40
d_{th}	10
a_{min}	0.2
g	2

the ball is occluded for a long time, usually when the trajectory goes across the frame and the ball goes behind the two horizontal bars, as shown in Figure 5.5.

(a) **(b)**

Figure 5.4: Example of a shot for players of the near team (a) and the far team (b). The heatmap produced by the neural network is superimposed on the frame, and the detection candidates of the network are shown as red dots. The orange, white and green lines represent the past, current and future trajectories respectively. The white boxes correspond to k_{min} and k_{max} of the fitted trajectory. The table on the upper left shows the frame index k, the estimated position p and velocity v, and the fit parameters p_0, v_0 and a in units of pixels, pixels per frame, and pixels per frame squared respectively.

(a) **(b)**

Figure 5.5: Example of a failed fit. For the meaning of the colors of the trajectories, refer to Figure 5.4. In addition to there, here the darker lines represent the continuation of the fitted trajectories outside their support set. What happens is this: in Figure (a) the detector fails to detect the ball for 3 consecutive frames (1496 to 1499). The position candidate in frame 1499 (Figure (b)) is then too far away from the estimated trajectory to be considered in the supports, and therefore the previous path is not refined. In this frame, the trajectory fitted by the seed triplet is too noisy, and is filtered out.

5.4 Discussion

There are various possible choices for the detector. At the end, using the previous output as input for detecting the ball in the next frame proved to be counter-productive, and training the model to deal with it is challenging. The RegNetY encoding-only performs poorly, and this is not surprising. The number of input frames does not affect much the performance of the detector, but setting it too high might make it impossible to train. The original number of 3 input frames is probably the best choice. Regarding the trajectory fitting, using the 2D image space is a starting point. Since a time-continuous position estimation is possible, multiple cameras can be used and the position triangulated.

6 Conclusion

This study has established that the proposed system is capable of identifying and tracking the ball in a padel match video effectively. The highest performing detector was found to utilize two input frames, but taking three frames as input dives similar performances.

A key strength of this system lies in its low rate of false positives, which greatly enhances the efficiency of trajectory fitting. However, there's room for further exploration into different types of detectors, such as more compact neural networks, which might reduce computational demand. A higher false positive rate could result from such a detector, but the effect could be mitigated during the trajectory fitting process.

This research opens avenues for more comprehensive analysis of trajectories, specifically in detecting floor and wall bounces. Such analysis could be realized by examining the angle of velocity vectors between neighboring trajectories. Moreover, the performance and applicability of the system could potentially be enhanced by combining views from multiple calibrated cameras, and by projecting the ball's position in 3D space. However, this would demand additional hardware resources, and in such instances, established systems like Hawk-Eye may prove more appropriate.

In conclusion, this research contributes significantly to the field of ball tracking and detection, providing valuable findings and opportunities for future work. The methods investigated within this study could potentially be commercialized and marketed to amateur padel clubs, providing them with a reliable ball tracking system.

Bibliography

[1] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[2] Yu-Chuan Huang, I-No Liao, Ching-Hsuan Chen, Tsì-Uí İk, and Wen-Chih Peng. Tracknet: A deep learning network for tracking high-speed and tiny objects in sports applications. 7 2019.

[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. 5 2015.

[4] Fei Yan, Alexey Kostin, William Christmas, and Josef Kittler. A novel data association algorithm for object tracking in clutter with application to tennis video analysis. volume 1, pages 634–641, 2006.

[5] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.

[6] N Owens, C Harris, and C Stennett. Hawk-eye tennis system, 2003.

[7] Your virtual badminton coach. powered by artificial intelligence.

[8] Michele Calvanese. *https://github.com/michele98/ball_tracking_padel*.

[9] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *CoRR*, abs/2003.13678, 2020.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 12 2015.

[11] Lluis Mendoza Vandrell. Seguimiento de una pelota de pádel a partir de vídeos. Bachelor's thesis, 2022.

[12] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[13] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

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

[15] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.