Universitat Autònoma de Barcelona

Master Thesis

Master in Modelling for science and engineering

A Study of Ball and Player Detection System in Paddle Tennis with Deep Neural Networks

Juan Carlos Barroso Ruiz

Supervised by **Dr. Sundus Zafar**

Year

2022/23

Acknowledgments

I like to see life as a weighted random walk, with time-dependent weights and these being all influences of people we meet along the way. So, if I were to thank someone, I should probably thank everyone and everything I met up until this point. I'm afraid this would be impossible and not very heart-touching which is what acknowledgements are supposed to be, I guess.

Therefore, I would like to thank, first of all, my family, specially my parents for supporting me during my education. I would also like to thank my girlfriend too, for putting up with me during the writing of this work and my hesitating nature.

Last but not least, I would like to thank my supervisor Dr. Sundus Zafar for her dedication to me and her guidance. I would like to specially thank the effort she made to review this work in August which should be typically a holiday and resting period.

Thank you all,

Abstract

Object detection holds significance in sports technology, enhancing gameplay assessment and training methods. This work contributes to this field, showcasing the role of computer vision in refining performance analysis and strategic decision-making.

This thesis presents a ball and player detection system designed for paddle tennis, using computer vision techniques. The system leverages the OpenCV library Python API along with Deep Neural Networks, specifically YOLOv5 and ResNet50, for ball detection.

Because of the lack of large amounts of data for paddle tennis we had to manually create data and use different data augmentation and image processing to be able to obtain a performant ball detection system.

We suggest that higher image resolution and higher fps (frames per second) video footage can substantially improve the performance of the system without modifying the architecture.

Finally, we use the extracted player positions to draw a heatmap for visualizing the areas where the players stayed the most time during the game. As for the ball detection, we use the positions detected to generate a comprehensive graph in polar coordinates showing the density of shots directions for every angle.

The findings suggest that the extracted information can be valuable for coaches and players in making well-informed decisions about game tactics.

Contents

1 Introduction and motivation

Object detection in image and video footage technology has highly increased its relevance in our society during the past decades. On one hand, computing technology and digital storage capacity has exponentially improved, meaning that it became feasible to store large quantities of visual media and the exponential increase in computing speed has also contributed to speed up image and video processing resulting in devices capable of recording high quality large amounts of videos and images.

On the other hand, with the rise of social media and smartphones, the volume of visual media data to exploit has exploded during the last decade. Due to all this progress, object detection systems are increasingly gaining momentum and will probably continue to do so in the future as they offer new automation possibilities.

Object detection in the 90s and 2000s consisted mostly in computer vision algorithms and heuristics which helped identify patterns in a region of an image or a video. In the last decade, the field has also adopted Deep Neural Networks and in particular Convolutional Neural Networks as a tool and has even displaced the main role domain specific computer vision algorithms had had since the start of the object detection study field.

In this work we present an example of object detection work for sports, utilising computer vision algorithms, image processing and Neural Networks to accomplish player and ball detection in paddle tennis matches. We give a comprehensive explanation of the thought process behind the designing the object detection system and discuss the quality of results along with the encountered data quality constraints. Finally, we hint how coaches or the industry could use the position data to enhance gameplay analysis and tactics.

The following chapter, Chapter 2, *Preliminaries*, gives a brief introduction to Machine Learning, in particular Supervised Machine Learning, types of tasks such as Regression and Classification.

In Chapter 3, *Research problem*, we define precisely the research problem at hand and provide the foundations of the technology/algorithms used to solve the problem.

Next, in Chapter 4, *Implementing a ball and player detection system* we expose the design process and final implementation architecture of the object detection system.

Finally, in the last chapter,*Conclusions and future work* we give a couple of examples of how the system can be used to generate data useful for gameplay tactics.

2 Preliminaries

2.1 Brief introduction to Statistical Learning

Statistical Learning is concerned with estimating the Probability Distribution Function (PDF) of random variables $\mathcal{X}$ using a finite number of observations of $\mathcal{X}$. Knowing or estimating this PDF can then be used to predict future observations of these random variables. The process of estimating this PDF, inevitably introduces some uncertainty in the future observation estimate, because we can never be 100% sure that the estimated PDF is the correct one, as for that we would need an infinite number of observations.

In general we have a set of observations of random variables $\mathcal{X}_i$ in the following way:

$$\begin{Bmatrix} \boldsymbol{x}_{11} & ... & \boldsymbol{x}_{1n} \\ . & ... & . \\ . & ... & . \\ . & ... & . \\ \boldsymbol{x}_{n1}, & ..., & \boldsymbol{x}_{nn} \end{Bmatrix}$$

where $\boldsymbol{x}_{ij}$ is the jth observation of the ith random variable $\mathcal{X}_i$.

There are many different ways to estimate this PDF, and depending on the methods used different names are used to refer this process of estimation.

2.1.1 Maximum Likelihood Estimation

In *Supervised Learning* we are interested only in a fraction of the random variables observed. Hence, we use the others as *predictors* or *features* to model the PDF of the ones we are interested in, called the *response* or *target* variable(s).

Let $(\mathcal{X}_1, ..., \mathcal{X}_K)$ be the features and $(\mathcal{Y}_1, ..., \mathcal{Y}_N)$ be targets. Ideally we want to find the following probability distribution:

$$p(\boldsymbol{y}_1, ..., \boldsymbol{y}_N | \boldsymbol{x}_1, ..., \boldsymbol{x}_K) \tag{1}$$

where $y_i \in \Omega_{\mathcal{X}}$ and $x_i \in \Omega_{\mathcal{Y}}$ and Ω is the *sample space* of the respective random variable. This information would be sufficient to estimate the probability of any $\boldsymbol{y}_i$ value.

2.1.2 Parametric Supervised Learning

As aforementioned, there are many ways of estimating the PDFs we are interested in. One way is to parameterize the conditional probability distribution (1) and then reformulate the problem of estimating (1) as an optimization problem.

To do this, we start with the joint PDF of N observations

$$p(\boldsymbol{y}_{11}, ..., \boldsymbol{y}_{1M}, ..., \boldsymbol{y}_{NM}, \boldsymbol{x}_{11}, ..., \boldsymbol{x}_{1Q}, , ..., \boldsymbol{x}_{NQ})$$

which we will call $p(\mathcal{D})$ to abbreviate. Then we condition it on some parameters $\boldsymbol{\theta}$. We now argue that the optimal value for the parameters $\boldsymbol{\theta}$ is:

$$\boldsymbol{\theta} = \text{argmax}_\theta \; p(\mathcal{D}|\boldsymbol{\theta}) \tag{2}$$

that is, the optimal value of $\boldsymbol{\theta}$ will be the one that maximizes the likelihood of obtaining these exact N observations.

We now use the so called *independent and identical distributed (iid)* assumption on the observations of the joint random variable $\mathcal{Z} = (\mathcal{Y}_1, ..., \mathcal{Y}_N, \mathcal{X}_1, ..., \mathcal{X}_K)$:

- **Independent**: we assume that the observations are independent of each other: that is $p(\mathcal{Z} = \boldsymbol{z}_i, \mathcal{Z} = \boldsymbol{z}_j) = p(\mathcal{Z} = \boldsymbol{z}_i)p(\mathcal{Z} = \boldsymbol{z}_j) \quad \forall \; i, j$
- **Identically distributed**: we assume that all observations follow the same probability distribution $p_{\mathcal{Z}}$

With this assumption the equation (2) turns into:

$$\boldsymbol{\theta} = \text{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{N} p(\boldsymbol{y}_{i1}, ..., \boldsymbol{y}_{iM}|\boldsymbol{x}_{i1}, ..., \boldsymbol{x}_{iQ}, \boldsymbol{\theta}) \tag{3}$$

where we assumed that the features are independent of the parameters: $p(\boldsymbol{x}_{i1}, ..., \boldsymbol{x}_{iQ}|\boldsymbol{\theta}) = p(\boldsymbol{x}_{i1}, ..., \boldsymbol{x}_{iQ})$. By taking the logarithm we can rewrite the optimization problem in (3) as:

$$\boldsymbol{\theta} = \text{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{N} \log p(\boldsymbol{y}_{i1}, ..., \boldsymbol{y}_{iM}|\boldsymbol{x}_{i1}, ..., \boldsymbol{x}_{iQ}, \boldsymbol{\theta}) \tag{4}$$

The optimal $\boldsymbol{\theta}$ will be the same as in (3) as the logarithm is a strictly increasing function; $\max X = \max(\log X)$.

The parameters $\boldsymbol{\theta}$ define a family of possible PDFs for $p(\boldsymbol{y}|\boldsymbol{x}, \theta)$.

At this step, and depending on the type and domain of the target variable, we might assume that the conditional distribution $p(\boldsymbol{y}|\boldsymbol{x}, \theta)$ follows a certain distribution.

We mainly have two types of cases: discrete targets and continuous ones. The problem of predicting a discrete target is called **classification** while the one of predicting a continuous one is called **regression**. There are of course situations in which we might want to predict both at the same time.

The general method to formulate a model is to start with the conditional probability distribution the target variable will follow with respect to the targets and parameters. To do this, we postulate that the target variable follows a certain distribution $\rho(\boldsymbol{y}|\boldsymbol{\mu})$, where $\boldsymbol{\mu}$ are the characteristic parameters of the distribution that define the particular probability distribution within a given family. For example, in the one dimensional Gaussian distribution the parameters are μ, σ^2 where they are respectively the mean and the variance of the distribution.

Now, we let these parameters depend on the features $\boldsymbol{x}$ and a new set of parameters $\boldsymbol{\theta}$. In other words, we let $\boldsymbol{\mu} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{w})$. These new set of parameters are the ones appearing in (1) and the ones that are subject to our optimization problem to

maximize the log-likelihood.

Classification

In classification we start by choosing a discrete probability distribution for the target variable. For example, our variable y might have only two possible values $y \in \{0, 1\}$ and we might choose the *Bernoulli* distribution to model it:

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \text{Ber}(y|\theta(\boldsymbol{x}, \boldsymbol{w})) = \theta^y(1-\theta)^{1-y} \tag{5}$$

the parameter θ is the characteristic parameter of the Bernoulli distribution, it represents the probability of obtaining $y = 1$.

Usually, discrete probability distributions have as parameters the probability of obtaining a particular outcome for the target variable. This poses an issue or constraint over the type of dependence that the parameter θ can have with respect to the features $\boldsymbol{x}$ and *weights* $\boldsymbol{w}$. The function $\theta(\boldsymbol{x}, \boldsymbol{w})$ must be inside the $[0, 1]$ domain.

One way to fulfill this constraint is to apply a **softmax** transformation to θ defined as

$$\text{softmax}(\theta) \equiv \sigma(\theta) = \frac{1}{1 + e^{-\theta}} \tag{6}$$

With this transformation any possible value $\mu(\boldsymbol{x}, \boldsymbol{\theta})$ will be mapped to the interval $[0, 1]$. This is the particular case for only one parameter μ but can be extended to include any number of parameters $\boldsymbol{\mu}$:

$$\text{softmax}(\boldsymbol{\theta}) = \boldsymbol{\sigma}(\boldsymbol{\theta}) = \left(\frac{e^{\theta_1}}{\sum_{i=1}^{N} e^{\theta_i}}, ..., \frac{e^{\theta_N}}{\sum_{i=1}^{N} e^{\theta_i}} \right) \tag{7}$$

This is not the only type of transformation possible, of course. Any transformation T fulfilling $T : \boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{w}) \longrightarrow [0, 1] \in \mathbb{R}$ would be valid too.

Regression

In regression we deal with continuous targets $y \in \mathbb{R}$ and its continuous probability distribution. For example, a very common approach is to model it using a Gaussian distribution with mean and variance μ and σ^2 respectively :

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(y|\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{w}), \sigma^2(\boldsymbol{x}, \boldsymbol{w})) \tag{8}$$

In general it is also very common to drop the dependence of the variance with respect to the features and weights, so we are often left with:

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \mathcal{N}(y|\boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{w}), \sigma^2) \tag{9}$$

Now, once the model is formulated, we can plug in the particular expressions for $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})$ into (3) and solve the optimization problems for the weights $\boldsymbol{w}$.

The particular expression we will get depends on the function $\boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{w})$. The function imposes a constraint over the search space for the possible θ's.

If we were optimizing directly for θ, we would be able to find the global maximum for (3) given the target observations $\boldsymbol{y}_i$. However, as we imposed a constraint that θ has to depend on the weights and parameters in a certain way, this global maximum is not guaranteed to be found. Instead we will find a local maximum whose proximity to the global one will depend on the constrain form $\boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{w})$.

For instance, a linear function/constraint of the type $\sum_i \boldsymbol{w}_i \boldsymbol{x}_i$ is more restrictive than a non-linear function of the type $\sum_i \boldsymbol{w}_i (\boldsymbol{x}_i)^i$. By choosing more complex functions we can potentially obtain a model that fits better the training data, but that may come at a cost of increasing the complexity and thus the explainability of the model.

2.2 Deep Neural Networks

Deep Neural Networks (DNN) are non-linear models where the parameter function is of the form $\boldsymbol{\theta}(\boldsymbol{x}, \boldsymbol{w}_1, ..., \boldsymbol{w}_L) = \boldsymbol{\varphi}_1(\boldsymbol{w}_1) \circ \boldsymbol{\varphi}_2(\boldsymbol{w}_2) \circ ... \circ \boldsymbol{\varphi}_N(\boldsymbol{w}_L)(\boldsymbol{x})$ where $\boldsymbol{\varphi}$ are non-linear transformations called **activation functions**. The simplest type of Neural Network is perhaps the Multilayer Perceptron (MLP) in which the activation functions are of the form:

$$\boldsymbol{\varphi}_l(\boldsymbol{\varphi}_{l-1}, \boldsymbol{W}_l) = \boldsymbol{\varphi}(\boldsymbol{W}_l \boldsymbol{\varphi}_{l-1} + \boldsymbol{b}_l) \tag{10}$$

where $\boldsymbol{b}_l$ are the biases at layer l. Again, by plugging the nested expression $\boldsymbol{\varphi}_1(\boldsymbol{w}_1) \circ \boldsymbol{\varphi}_2(\boldsymbol{w}_2) \circ ... \circ \boldsymbol{\varphi}_N(\boldsymbol{w}_L)(\boldsymbol{x})$ into (3) and finding the maximum we can determine the set of weights $\{\boldsymbol{w}\}_1^L$ that output the highest likelihood for the conditional probability $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w})$.

When maximizing the likelihood irrespective of the model distribution chosen, we deal with the partial derivatives of the weights $\boldsymbol{w}$ w.r.t the transformations the initial input propagates through. The calculation of these partial derivatives using the chain rule going all the way down to the first transformation layer is referred as **backpropagation**. This in reference to the "flow" of the gradient, that goes backwards through the layers in the network. As complementary term, we also have **forward propagation** which refers to the flow of the inputs as they are transformed throughout the layers of the network.

These non-linear models are very powerful to the point that they can approximate any function with adequate tunning of the weights. In fact, we have that a DNN model as defined in (10) with $l = 1..3$ is already a **universal function approximator**[11]. However, the idea of supervised statistical learning is to predict future observations, that is, given a feature observation predict the associated target value. Therefore, maximizing without measure the likelihood over the training data $p(\mathcal{D}|\boldsymbol{\theta})$ is not always the best strategy for predicting future targets based on future feature observations as this may lead to **overfitting**.

In addition to the aforementioned statistical concern, we also have to deal with optimization concerns. Even if we have formulated a promising model for predicting future data, we have to solve the optimization problem for the likelihood to find the best value for the weights. This can present different complications. Because these optimization problems often doesn't have a closed form for the solution, we have to use different kinds of algorithms to find the global maxima for the weights.

Depending on the form of the optimization problem we might have more or less difficulty for the algorithm to converge to a maxima, leading to increase computation cost and long times until convergence.

Because of this, different architectures and algorithms for DNNs are proposed depending on the task and data at hand.

2.3 Deep Neural Networks for Image data

DNN for image data are used for a variety of tasks, like detecting objects in an image or classifying different types of images (e.g. if an image is a Dog or a Cat). However, DNNs used for these tasks aren't usually simple MLPs. This is because feature dimensionality in an image can be easily in the order of 10^6 - 10^7 if we consider each pixel to be a feature. Then, the number of weights will be several orders of magnitude more than that considering that the dimensions of the weight matrix $\boldsymbol{W}$ have to be (number of weights $\cdot\, 10^6$). If we want a decent model the number of weights will have to be at least a few orders of magnitude, so this clearly gets out of hand quickly in terms of computational power and convergence time.

On top of this, the one dimensional feature vector $\boldsymbol{x}$ feed into MLPs representing the image, will be treated differently depending on the order of the features. This means that if there's a dog in the top right corner of the image and another in the top left corner these "spatial" patterns won't be leveraged by the network.

In order to leverage the spatial structure of the images, we need to feed the image in a higher dimensional tensor form. Usually images can be thought of 3-rank tensors X_{whc} where w, h, c are respectively the width, height and channel of the image. The channel index contains the color information. If we are dealing with a color image $c = 0, 1, 2$ that represents the Red Green and Blue (RGB) spectrum of a pixel in the position w, h.

Having changed the input shape, we can engineer a new type of transformation that takes advantage of spatial correlations among the different regions in the image:

$$H_{whc} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} \sum_{c=0}^{2} V_{abc} X_{w+a,h+a,c} \tag{11}$$

In this case $\boldsymbol{H}$ is called the **hidden** representation of $\boldsymbol{X}$, which is the image tensor representation and $\boldsymbol{V}$ is the weights tensor which is known as the **kernel**. Depending on the **kernel size** $\Delta \in \mathbb{Z}$ the dimensions of w, h will shrink more or less. Hence, what we are effectively doing is a weighted sum over a region of pixels centered at $a, b = 0$. As a result, the hidden representation $\boldsymbol{H}$ will have a pixel at w, h that is created by the weighted sum of a square region of sides size Δ.

The operation just defined is a **cross correlation** operation of $\boldsymbol{V}$ over $\boldsymbol{X}$. In practice though, in the deep learning literature, it is called a **convolution**. This is because they're both deeply related:

$$\text{Conv}(\boldsymbol{W}, \boldsymbol{X})_{ij} = \sum_{a=\Delta}^{-\Delta} \sum_{b=\Delta}^{-\Delta} W_{ab} X_{i+a,j+b} \tag{12}$$

So we see that there's only a flip in the order of evaluation in the summed indices $-\Delta \to \Delta$.

This new transformation also reduces the number of parameters from 10^{6+} to an order of Δ^2. Given that Δ is around 10 in order of magnitude, this new layer greatly reduces the number of weights to optimize, and also provides spatial invariance, as the same weights are applied to different spatial regions of the image.

Convolutional Neural Networks (CNNs) are based in this type of transformations and are extensively used for image data.

In the next section we discuss two particular types of DNNs that are used in image object classification and detection: ResNet [10] and Yolo [13].

3 Research problem

3.1 Introduction and motivation

Over the past decade, there has been a remarkable surge in sports-related technology aimed at extracting crucial information to enhance athletes' performance. This includes real-time statistics displayed during games and visual graphics that assist in assessing the game's status in team sports. Notably, this information benefits not only spectators but also athletes, players, and coaches, empowering them to make decisions that increase the likelihood of winning or improve performance in the medium to long term.

Paddle tennis is a sport resembling tennis, primarily played in doubles at the competitive level. The court consists of a 20 x 10 meter platform surrounded by specialized crystal walls and metal grilles. The rules of paddle tennis are similar to tennis, where the ball can only bounce once on the floor, and the objective is to hit it over the net and into the opposing team's court in a way that makes it difficult for them to successfully return it. The game follows the same scoring system as traditional tennis, with players striving to win points, games, and sets.

In contrast to tennis, paddle tennis allows the ball to bounce off the walls and grilles. However, it is important to note that the ball must first bounce on the floor before contacting the walls or grilles directly.

There is many different data that could be extracted to analyze the unfolding of a match such as number of points won (which doesn't necessarily imply winning the match). The breakpoints generated and won[1]

However, this information is highly abstracted to what actually happens in the gameplay. If we think of the gameplay in terms of a physical system, it is just a bunch of shots performed by different players that go in different directions, that end up in a point won by one of the couples. This motivates the definition of a state variables that can define the state of the system at each point in time. A reasonable choice could be:

$$\text{State} = \text{State}(t, \boldsymbol{x}_{\text{ball}}, \boldsymbol{x}^1_{\text{player}}, \boldsymbol{x}^2_{\text{player}}, \boldsymbol{x}^3_{\text{player}}, \boldsymbol{x}^4_{\text{player}}) \tag{1}$$

where $\boldsymbol{x} = (x, y, z)$ are the coordinates of the ball and/or players respectively at time t.

Having this information could be used to obtain new insights that are not evident at a high level (points, games, sets) and it could be even possible to establish a correlation and/or causation between a series of "microstates" and a "macrostate" like at a game level. For instance, we could determine that when 80% of shots go through the right side of the court one couple tends to win the game.

In order to translate visual information into numeric features (microstates) it is required to do some video processing and analysis. What we need is to be able to detect the players and the ball position in each frame (or brief series of frames) and

[1]It is known that the probability for a couple to win their serve is substantially greater than that of losing it. Hence, the couple that wins the other couple's serve at some point in the set usually wins the set. When a couple wins the other's serve, it is called a *break*, because the one couple *broke* the other couple's serve. This is akin to tennis.

then aggregate all this data into dataset that we can further process and extract statistics from.

In order to extract the microstate information we will mainly be using Deep Neural Networks, some Computer Vision algorithms and some heuristics to transform raw video footage into a numeric dataset that we can further analyze.

3.2 Problem definition

The goal is to be able extract the x, y position on the image plane of the ball and the players for each image. The video quality of the footage used is 1080p and 25fps, which is the quality of the video footage uploaded to `https://www.worldpadeltourtv.com`. World Padel Tour is the main international paddle tennis tournament at the present time of this writing.

In order to detect the ball and players in each video frame of a padel match we use different image processing techniques and countour dection algorithms.

In addtition, we use neural networks to further refine detection using neural networks. In particular, two particular CNN architectures: ResNet and YOLO. The specific versions are: ResNet50 and YOLOv5. These are implemented using PyTorch[12][1].

We now introduce the theoretical foundations of these architectures

3.2.1 ResNet Architecture

The ResNet model is a type of CNN with a **backbone** (main part of the network that characterizes the architecture) made of several so called **residual blocks** stacked together ending in a custom MLP head that depends on the particular task at hand.

The key benefit of the introduction of residual blocks is it allows for a more efficient backpropagation making the optimization process faster. Moreover, given the optimial mapping $f(\boldsymbol{x})$ optimizing for the difference $f(\boldsymbol{x}) = g(\boldsymbol{x}) + \boldsymbol{x}$ is easier than directly optimizing for $f(\boldsymbol{x})$. The following image (1) shows the general architecture of a residual block and how it is used in a ResNet CNN for image tasks.

On the left, we have a normal block, and on the right the modified block with the so called residual connection (in the form of an arrow) that connects the input of the block with the output of the block. This whole structure is the residual block

3.2.2 YOLO Architecture

YOLO (You Only Look Once) [13] networks propose a framework for object detction, reframing it as in regression problem. This type of networks are used to detect several objects in an image belonging to different classes. The network name comes from the fact that the image is only processed once ("looked at once") to predict the bounding boxes coordinates and the probabilities for the different classes. This is different from dividing the image into N patches and then running a classifier model on each one to detect if that patch belongs to a certain class.

To achieve this, the input image is subdivided into a $S \times S$ grid. Then, each cell in the grid predicts up to B bounding boxes which are defined by (x, y, w, h, IOU)

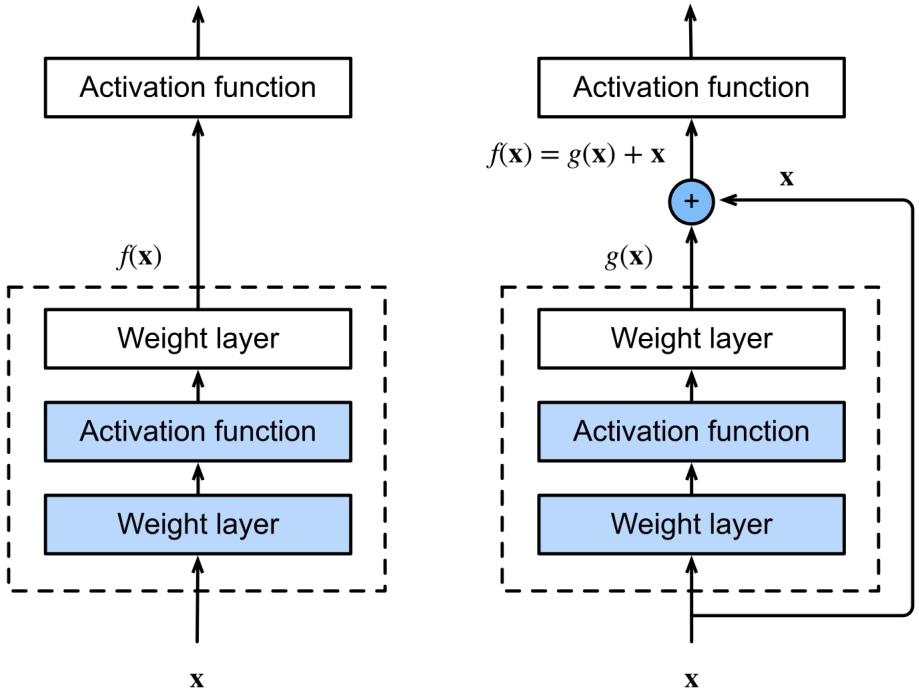


Figure 1: Residual block compared to a non-residual block [14].

where x, y are the center coordinates of the box B with respect to the cell bounds and w, h are the width and height coordinates of the center of the box with respect to the width and height of the whole image. Finally, the IOU is the Intersection Over the Union of the predicted bounding box and the ground truth bounding box.

Each grid cell also outputs a set of C class probability predictions conditioned to the probability that there's an object in the cell: $P(\text{Class}_i|\text{Object})P(\text{Object})$. At test time the class-specific confidence scores will be $P(\text{Class}_i) \cdot IOU$.

Figure (2) shows the original YOLO architecture as proposed in the original paper [13] although since then many improvements have been suggested and successfully implemented, increasing the network's original performance.

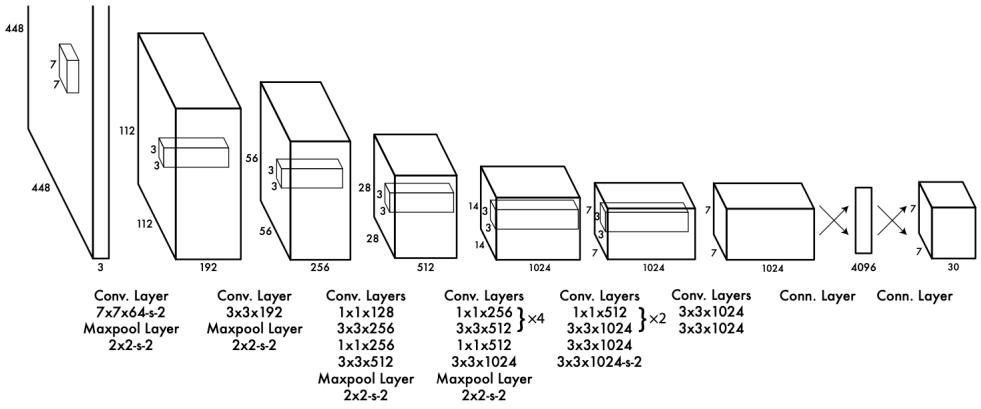


Figure 2: Original architecture of the YOLO CNN in article [14].

3.3 Input and output data

The source videos used for training and prediction are 1080p mp4 files running at 25fps (frames per second). One video frame is therefore a 1920×1080 RGB image (3). The goal is to detect the ball and the players positions for each one of the frames so that we can convert the timeline of a full match into a structured dataset containing (t, w, h) time, width and height respectively, for the ball and the 4 players. In order to extract the frames we use the utility `ffmpeg`[2] to extract frames in lossless `.png` format.

```
ffmpeg -i [video_file] -vsync 0 -f image2 frame%d.png
```

Figure 3: Example video frame.

The main challenge when trying to train the networks is the lack of large training datasets to leverage the power of modern CNNs for object detection, in particular, no dataset for the goal at hand was found so every piece of training data in this work had to be manually curated. To compensate the scarcity of data, we introduce data augmentation techniques along with computer vision preprocessing (implemented using OpenCV[3]) to reduce the number of features the networks have to deal with and perform an iterative process where we update the training dataset using the wrong predictions and train the ResNet model again so that the performance increases in each iteration of the process. A complete overview of the inference pipeline is shown in figure (4) which we explain in detail in the following sections.

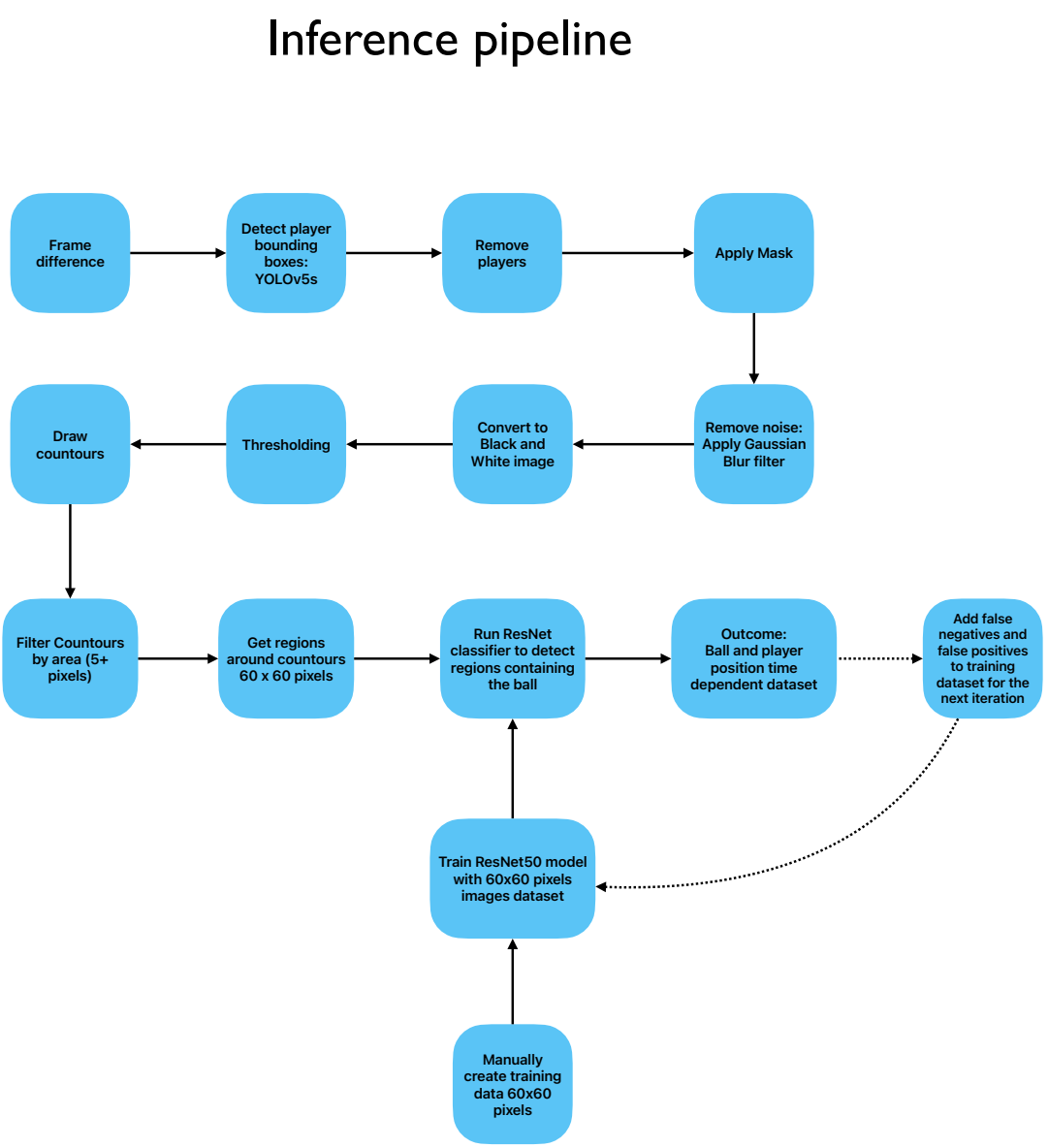


Figure 4: Inference pipeline.

4 Implementing ball and player detection system

Hereby we describe the solution implemented in this work, represented in (4). All steps explained in this section are implemented using Python and OpenCV and the PyTorch APIs. The full code used is available at a public Github repository at the following link: `https://github.com/Juild/padel-project-tfm`

We start off by applying a **frame difference** operation on every two consecutive video frames. Frame difference consists in doing an absolute difference operation with two consecutive frames on each channel value. The result is channels having the

same value will be set to zero[1] The result of this operation is shown in the following figure 5.

Figure 5: Result of applying a absolute difference operation on two consecutive frames.

As it can be seen, this technique drastically reduces the number of features present in the image. The only regions that are left are regions influenced by the player's motion, the ball motion and some background noise due to small variations in lighting conditions or even due to spectators motion in the background. We now use the pretrained YOLOv5 small model from Ultralytics[4] to detect the players. The model has been trained on the COCO dataset that is commonly used to asses performance of networks for object detection tasks. The output of the model are the coordinates of the bounding boxes around the players and other objects. In our case we are only interested in the players bounding boxes and also the racket bounding boxes. The result of applying the model to our input frames works well to predict the bounding boxes of the players so that we don't need any further fine tunning:

[1] in the RGB scheme $(R = 0, G = 0, B = 0)$ represents the black color, and $(R = 255, G = 255, B = 255)$ the white color.

Figure 6: YOLOv5 predictions for different classes present in the COCO dataset along with their respective confidence scores.

We now leverage the player's bounding boxes prediction we obtained before and eliminate the players from the image by setting the pixels inside said bounding boxes to zero (black). The result is the folllowing:

Figure 7: Frame difference with players removed.

Now the only two non-black obvious regions of the image are the positions of the ball in the two consecutive frames.

This is the ideal scenario, however as previously mentioned there are other frame difference operations that have background noise, see for instance figure (8) where a player bounding box was not correctly predicted and therefore its mark in the frame difference couldn't be removed. Alternatively we may also have to deal with the reflection motion of the player projected in the court glass as figure (9).

Figure 8: Example of a frame difference where the player frame difference was not correctly removed.

Figure 9: Example of a frame difference with the player motion in the glass.

These motivates the use of another model to classify whether a region has or doesn't have a ball in it. To this end, we use the pretrained backbone of the ResNet50 model in the Pytorch Hub [5]. The model is based on [9] and the weights of the backbone are the ones that reproduce the results of [9]. For the head, we use a Fully Connected network with the following structure:

`Linear(2048, 512) + ReLU + Dropout` → `Linear(512, 256) + ReLU + Dropout` → `Linear(256, 2)`

The input dimensionality of the first Linear layer `2048` corresponds to the output dimensionality of the ResNet50 backbone. The final output dimensionality of the head is set to 2 because our target classes are either `ball` or `background`. We also use dropout to avoid overfitting.

In order to train the network we manually created a small dataset consisting of 60×60 pixel wide regions containing a either a ball or just background (i.e. positives or negatives classes respectively). The dataset contains 125 positive examples and 1281 negative examples. It is highly unbalanced as manually creating positives is more time costly that just using negatives; for negatives we just a full image with no balls and divide it in 60×60 regions whereas for positive we just have one positive class per image and we need to manually find it after dividing the image. Some positive and negative examples are shown in figure (10)

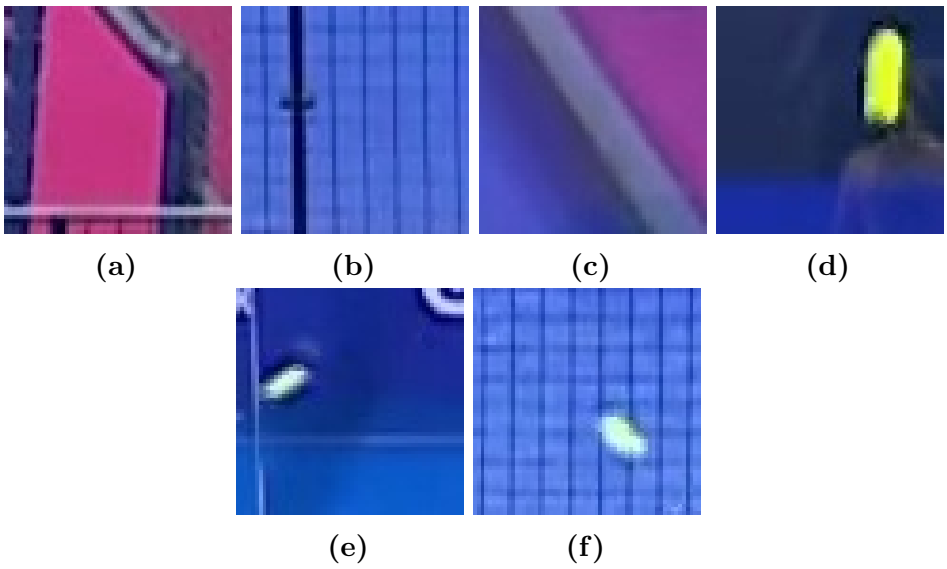


Figure 10: Training data examples of positive **(a)**, **(b)**, **(c)** and negative classes **(d)**, **(e)**, **(f)**

We now train the model for 35 epochs using the full training dataset. Using a batch size of 512 and before feeding the images to the network, we also normalize the pixel values using the dataset mean and variance values so that the output channels when normalized follow:

```
output[channel] = (input[channel] - mean[channel]) / std[channel]
```

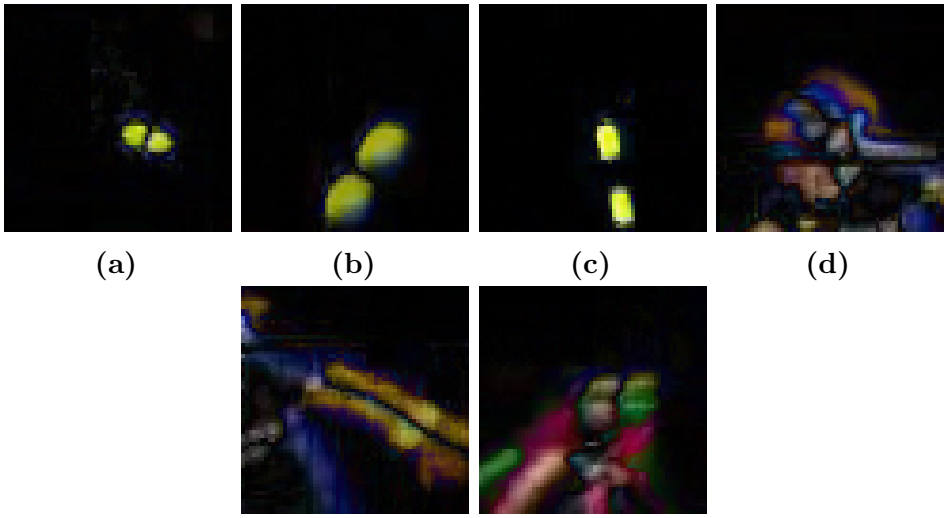
We do this in order to improve training time and performance. This is just the first step to obtain a larger training dataset by iteratively performing predictions on different video clips. Still, results are reasonably good as it can be seen on the following table (1):

Threshold = 0.5	True	False
Positive	103	9
Negative	1272	22

Table 1: Confusion matrix table results after training the initial model for 35 epochs.

At this point, we start processing the video frames by first applying the frame difference every two consecutive frames and removing the players. In order to further eliminate noise, we apply an RGB mask $(0, 100, 0) : (240, 255, 255)$. This will eliminate (set to 0) any pixel whose value is not within. The values for the mask were decided heuristically after a few experiments with different images. In order to reduce small noise regions we also apply a Gaussian Blur[6] filter 7×7 pixels wide, which will fade out small regions containing frame differences. This is specially important due to the next step, which is to find the coordinates of the regions showing the highest frame difference, as these are the candidates to contain the ball. We find these regions identifying the different countours of the shapes in the image. For this, the algorithm (function) needs the image to contain binary pixel values. The way to do this is to apply a *threshold*[7] operation over the image where we set pixel values above a certain threshold to its maximum value (255) and those below to the minimum value (0). Hence, it is useful and necessary to reduce the pixel values of small noisy regions (much small that the ball) that might interfere with this operation. Hence, the application of the Gaussian Blur filter aforementioned. Finally, we find the countours of each frame difference using [8] and obtain the center coordinates of each countour. We remove any countours that are less than 3 pixels in area, as these are unlikely to be the ball.

At this point we extract regions 60×60 of the *frame difference* images, using the center coordinates of the countours as center point. As the number of features is less than the same regions in a raw images and this will reduce the amount of data needed for our model to perform well. Examples of the regions extracted are shown in figure (11).



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 11: Training data examples of positive **(a)**, **(b)**, **(c)** and negative classes **(d)**, **(e)**, **(f)**

We leverage this process to generate positive and negative predictions for the classes and we split the regions 60×60 in two different directories automatically. This whole process is used then as a sort of filter to rapidly obtain positive and negative examples for the training dataset.

Figures (12) - (17) show an example of the whole process for a given frame. We see how the different image processing steps help in finally identifying the correct countour region containing the ball.

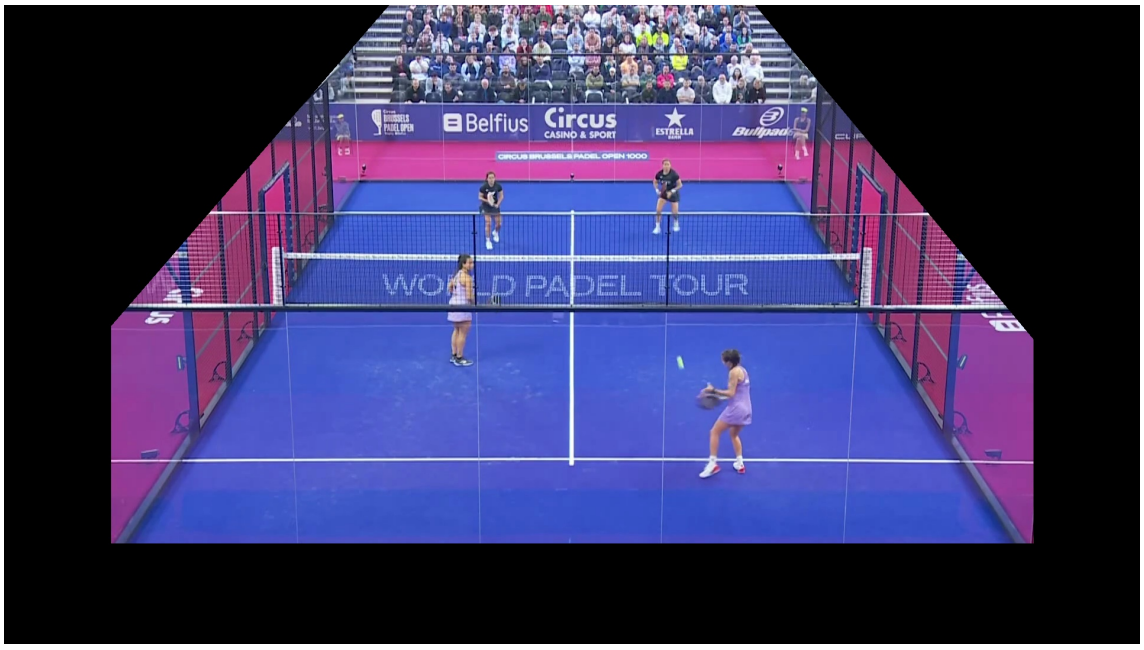


Figure 12: First step of the pipeline eliminate the image borders that are not relevant to the game.

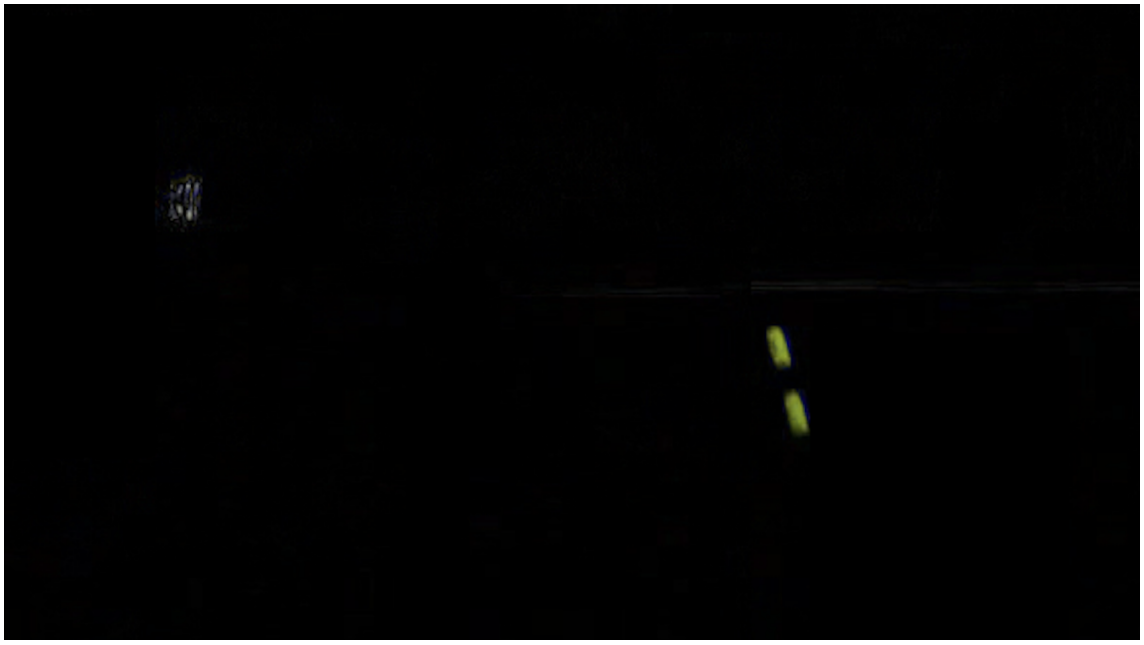


Figure 13: Second step, after eliminating the players apply a frame difference operation.

Figure 14: Third step, applying a RGB mask $(0, 100, 0) : (240, 255, 255)$.

Figure 15: Fourth step, apply Gaussian Blur filter 7×7.

Figure 16: Fifth step, apply thresholding with value 25

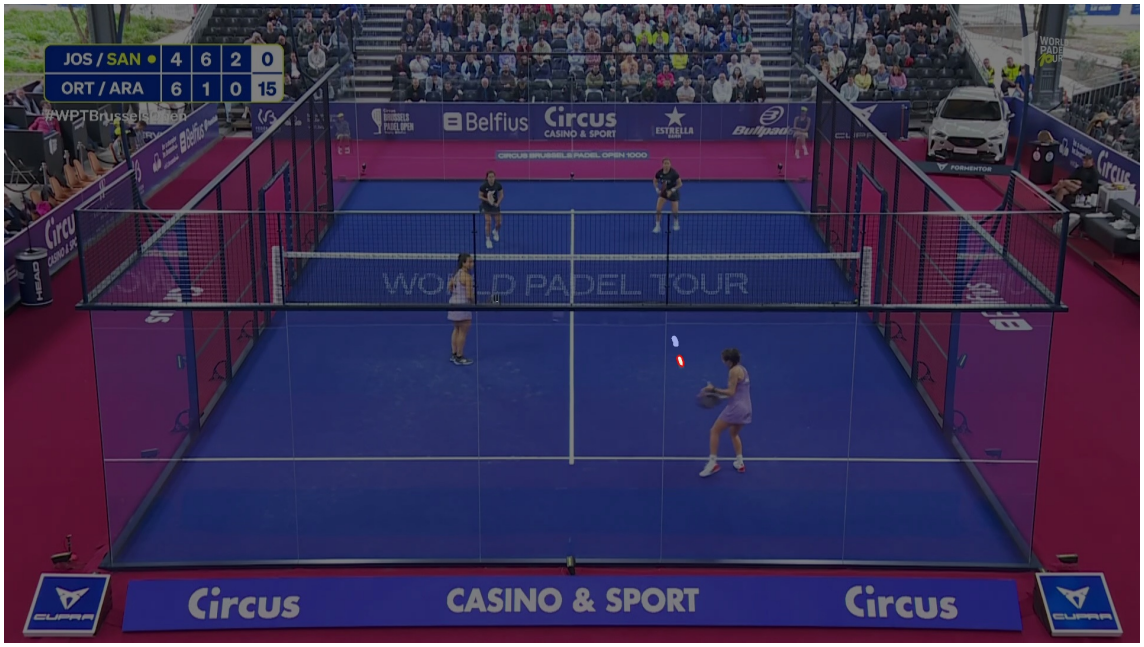


Figure 17: Countour detected in red.

We now retrain the network with the larger and improved dataset containing 567 negative examples and 488 positive examples. We also apply random rotations to the images at training time to augment the training data.

In order to decide the optimal training epochs we compute the mean and standard deviation of the training and test losses for each epoch. We also compute the Receiver Operating Characteristic (ROC) and Precision- Recall curves (PR-curve) along with the optimal thresholds.

For the ROC, we compute the optimal threshold, the operating point, by finding the confidence threshold that maximizes the difference between True Positive Rate (TPR) and the False Positive Rate (FPR). For the the PR-curve we compute the F_β score for $\beta = 1/2$ which corresponds to weight precision two times the recall, hence, we decide to have a higher probability of obtaining a positive prediction that is true positive rather than identifying all positives at the cost of an increase in false positives.

The following figures summarize the obtained results for the different metrics:

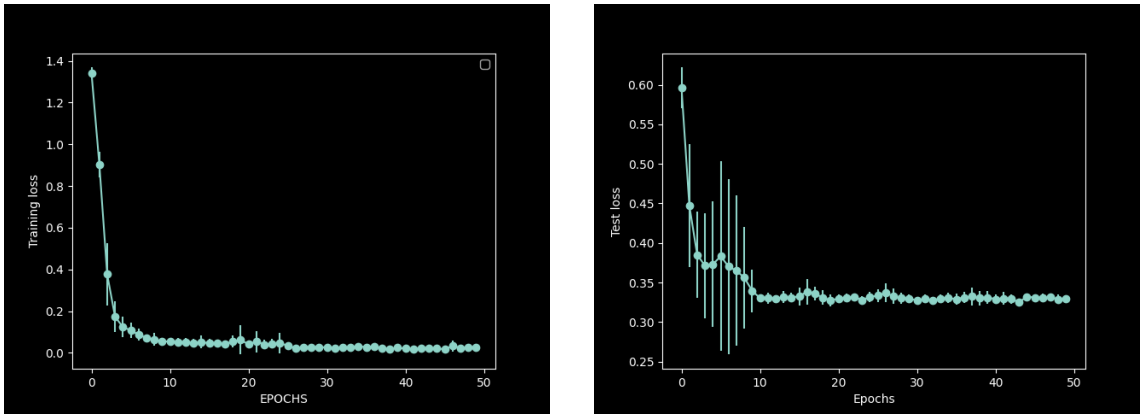


Figure 18: Mean train and test loss along 50 training epochs.

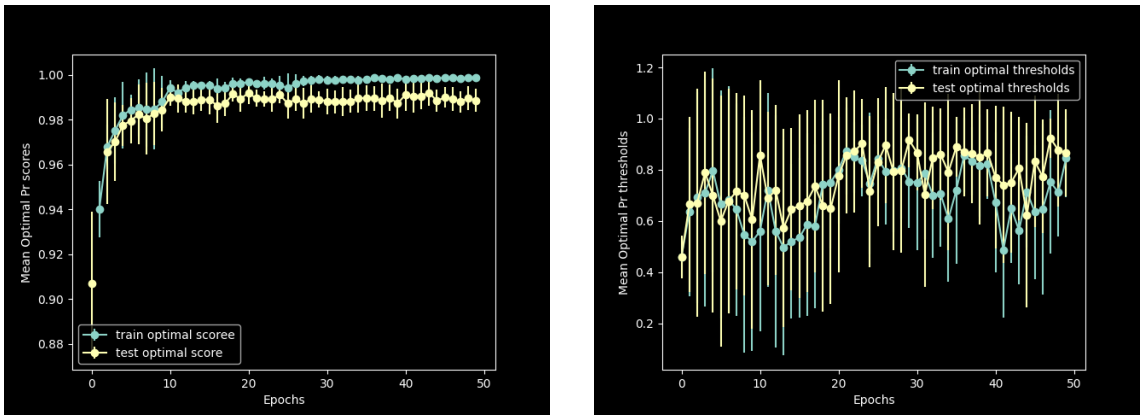


Figure 19: Mean $F_{\frac{1}{2}}$ score for the PR-curve and corresponding thresholds along 50 training epochs.

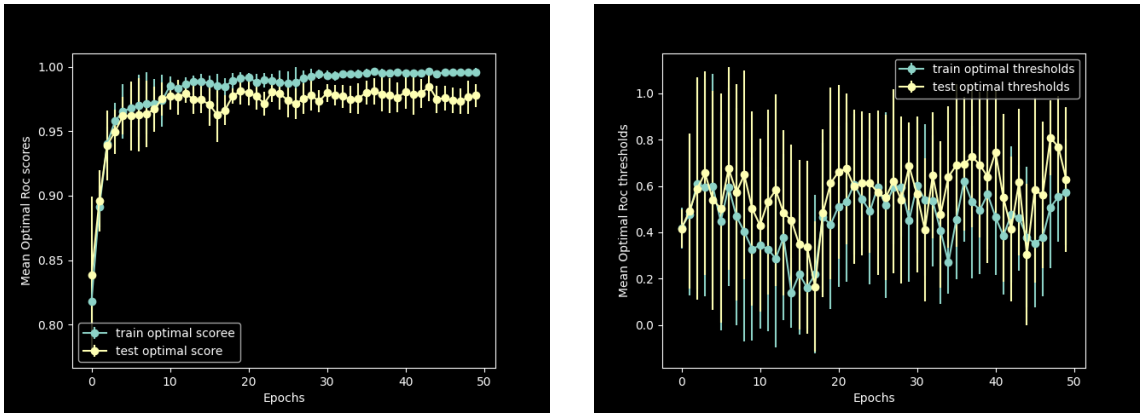


Figure 20: Mean operating point score (TPR - FPR) for the ROC and corresponding thresholds along 50 training epochs.

After examining the results we decide to train the final model for 23 epochs. training and test loss appear to converge at around 20 epochs while the optimal thresholds both for the ROC and the PR-curve tend to converge and minimize the standard deviation around the same number of epochs. The optimal confidence thresholds show a high variance, specially for the ROC. We attribute this kind of fluctuations to the presence of outliers in the training and test dataset. While the model is undergoing training, it will after each epoch iteration move in the direction that minimizes loss in the parameter's space. If such outliers exist in the dataset and the model is not able to generalize well it could produce the observed fluctuating pattern for the optimal thresholds: if the model corrects the loss with respect to some item while increasing the loss over another one and the model doesn't converge to a point where both losses are minimized at the same time, the optimal threshold will fluctuate. In particular, we argue that the reason why the optimal threshold metric fluctuates but the training loss or test loss doesn't fluctuate as much, is that the optimal threshold, because of its "optimal" nature, is much more sensible to model changes.

The ROC and PR-curves for the 23th epoch are shown in figure (21 They display an AUC of ~ 1. This indicates almost a perfect classifier, however given the amount of training data such high performance is difficult to extrapolate to a real distribution of data in lots of different video clips, lightning and noise conditions. However, when using the model in different video clips we see still a robust performance as will be demonstrated in the next section in the example video clip.

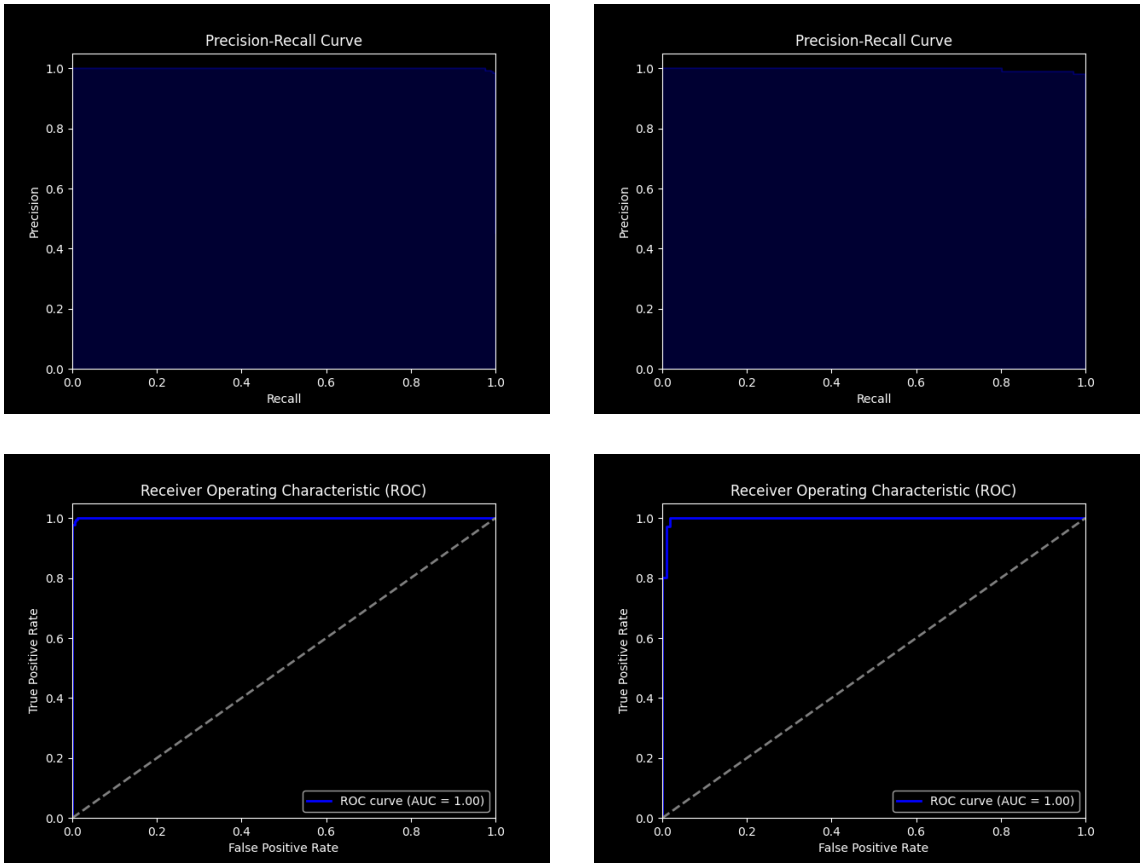


Figure 21: ROC and PR-curve for training and test splits at epoch 23.

5 Results and future work

Using the final prediction pipeline we are able to run the model with an inference speed of 10fps. The main results of this process is the dataset containing ball and player positions for each frame in the input video and the video clip with the drawn ball and player positions detected. An example of such predictions is uploaded at the following Google Drive link: `https://drive.google.com/file/d/1LumMJKmvP1nNSXaxsaOz2nyAOM_afUc_/view?usp=sharing`

Using the structured dataset obtained we have been able to drastically reduce the data dimensionality while preserving the relevant information for each frame, i.e. ball and player positions. This data can be analyzed using standard structured data analysis techniques, for instance we can visualize player position frequencies of the example video clip using a density plot (22) or analyze the most frequent ball directions (23b). This information can later be used for coaches and players to improve game strategy. For example, the ball direction histogram (23b) suggests that players in the left-right cross direction have a high ball volume[1]. Coaches can use this information to inform the players in the rest time and change strategy according to that.

[1]This term is commonly used in paddle to mean the amount of balls that a player strokes.

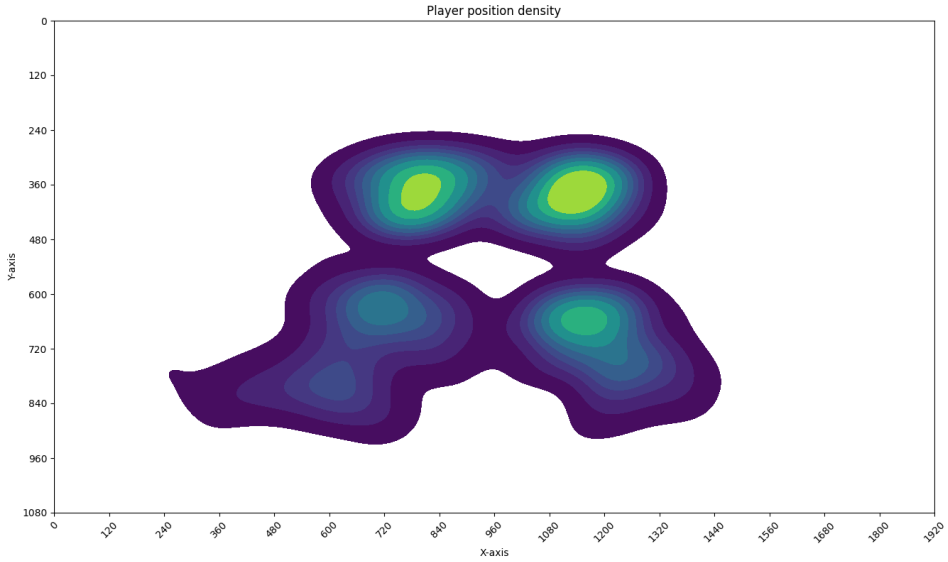
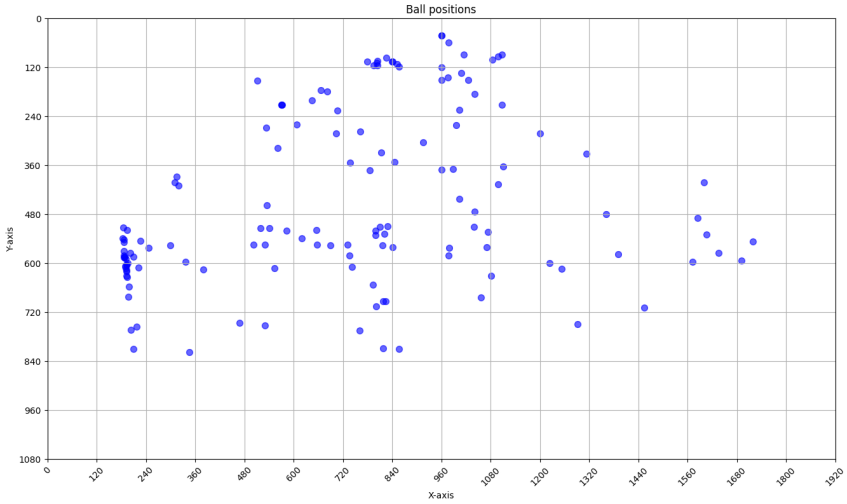
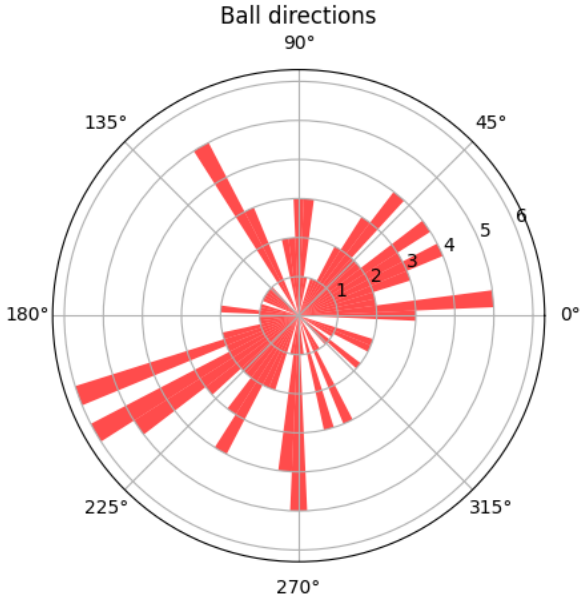


Figure 22: Density plot of player positions in the example video clip, lighter areas indicate higher density.



(a) Ball positions detected for the example video clip.



(b) Polar histogram showing most frequent ball directions. Only trajectories that cross the half of the court are shown.

As it can be seen, there are some frames in which the ball is not detected or there are still false positives. This could likely be improved by using more training data following the iterative process explained in the previous section.

The ball directions and player position densities shown can be affected by false positives or lack of predictions. However, given the classification performance obtained we expect that the noise weight should average out with the amount of predictions and then overall frequencies shouldn't be specially affected.

The present work hasn't dealt with the ball detection when this is passing in front of a player. Using video footage filmed from above court would reduce the amount of frames affected by this issue while also providing a way to estimate the depth of the ball in the court and hence obtaining the 3-dimensional position.

The work presented here shows that with rather low quality data it is possible

to obtain a compelling result for object video detection in paddle tennis. Higher resolution images and higher fps videos would help in reducing motion blur and reduce the variance in the ball shapes and forms and RGB spectra. This would reduce the variance of the distributions the network would have to fit, hence likely producing better inference results.

References

[1] URL: https://pytorch.org.

[2] URL: https://www.ffmpeg.org.

[3] URL: https://opencv.org.

[4] URL: https://github.com/ultralytics/yolov5.

[5] URL: https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Classification/ConvNets/resnet50v1.5.

[6] URL: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1.

[7] URL: https://docs.opencv.org/3.4/d7/d1b/group__imgproc__misc.html#gae8a4a146d1ca78c626a53577199e9c57.

[8] URL: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a.

[9] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[11] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[12] Adam Paszke et al. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: 1912.01703 [cs.LG].

[13] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[14] Aston Zhang et al. *Dive into Deep Learning*. 2023. arXiv: 2106.11342 [cs.LG].