

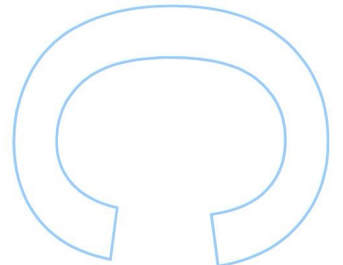
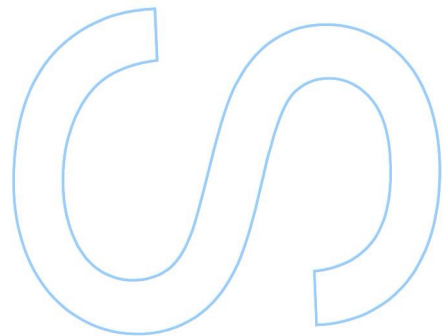
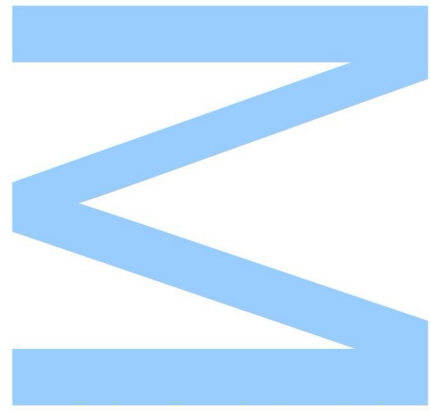
Real time tracking of Padel game movements, statistics and points

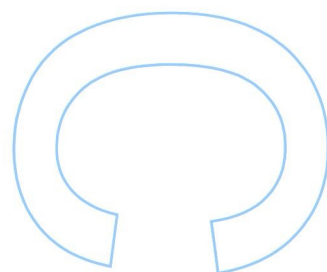
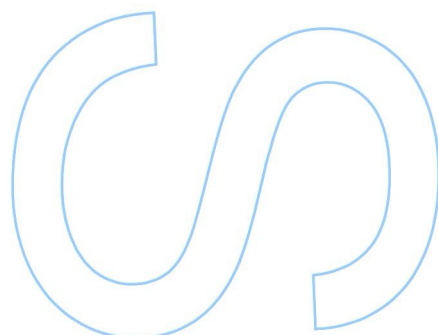
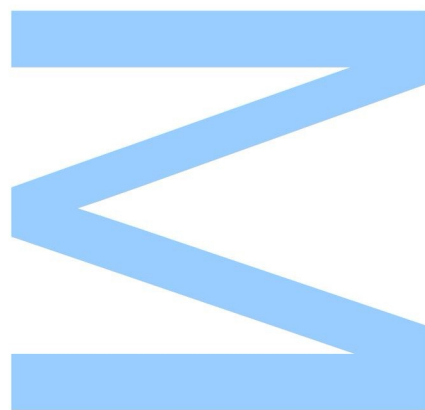
João Pedro Andrade Ferreira

Master's in Data Science
Department of Computer Science
2023

Supervisor

Miguel Tavares Coimbra, Full Professor
Faculty of Sciences of the University of Porto





Abstract

Padel is a popular racquet sport that is played in pairs. It is played on a smaller court with enclosed walls and is a cross between tennis and squash. People of all ages and skill levels enjoy the fast-paced, exhilarating sport of padel. It accounts 25 million players yearly, with a market size estimated at 327 millions US\$ and is expected to grow 9.6% until 2027. Additionally, the search interest in padel increased 385% over the past five years in Google's search engine. In 2021, the World Padel Tour's official Youtube channel hit an average of 2.3 million live viewers per tournament, meaning there is an opportunity to enhance the viewing experience by providing spectators in-depth insights of the game or player performance metrics. It might be challenging to provide insight of the game, like statistics, in an autonomous way as the game moves quickly along and how many things can influence how a point is scored. Conventional game analysis methods rely on human observation leading to biased results, and with manual data collection and analysis, the time spent on them is high. Artificial Intelligence (AI) makes these processes way preciser, delivering immediate feedback during the match.

In this thesis, solutions for the automation of the analysis of a padel match like the detection and tracking of players, with the additional detection of the strokes executed by them, were developed. Additional insight that the system could offer include the average ball speed, the average time to serve, the average time to hit a winner, etc... The results were obtained on the testing data which is independent from the training one.

Resumo

O padel é um desporto de raquete popular que é jogado a pares, num campo mais pequeno com paredes fechadas, e é um cruzamento entre o ténis e o squash. Pessoas de todas as idades e níveis de competência apreciam o desporto rápido e estimulante que é o padel. Contabiliza 25 milhões de jogadores por ano, com uma dimensão de mercado estimada em 327 milhões de dólares e prevê-se que cresça 9,6% até 2027. Adicionalmente, o interesse de pesquisa no padel aumentou 385% nos últimos cinco anos no motor de busca do Google. Em 2021, o canal oficial do World Padel Tour no Youtube atingiu uma média de 2.3 milhões de espetadores ao vivo por torneio, o que significa que há uma oportunidade de melhorar a experiência de visualização, fornecendo a estes uma visão aprofundada do jogo ou das métricas de desempenho dos jogadores. Pode ser um desafio fornecer uma visão do jogo, como estatísticas, de forma autónoma, uma vez que o jogo se move rapidamente e muitas coisas podem influenciar a forma como um ponto é marcado. Os métodos convencionais de análise de jogos baseiam-se na observação humana, o que conduz a resultados tendenciosos, e com a recolha e análise manual de dados, o tempo despendido é elevado. A Inteligência Artificial (IA) torna estes processos muito mais precisos, fornecendo feedback imediato durante o jogo. Nesta tese, foram desenvolvidas soluções para a automatização da análise de um jogo de padel, como a deteção e localização de jogadores, com a deteção adicional das pancadas executadas por eles. Conhecimento adicional que o sistema poderia oferecer inclui a velocidade média da bola, o tempo médio para servir, o tempo médio para bater um vencedor, etc... Os resultados foram obtidos nos dados de teste, que são independentes dos dados de treino.

Keywords: Padel, Real time statistics, Player detection and tracking, Computer Vision, Machine Learning, Deep Learning, Court detection, YOLO, Camera lens distortion

Contents

Abstract	i
Resumo	iii
Contents	vi
List of Tables	vii
List of Figures	xi
Acronyms	xiii
1 Introduction	1
1.1 Objectives	2
2 Literature Review	3
2.1 Padel Sport	3
2.1.1 Padel court	3
2.1.2 Padel strokes	4
2.1.3 AI in padel sport	5
2.2 Player and Movement Detection	7
2.2.1 CNN-based detectors	8
2.2.2 Histogram based detectors	20
2.3 Camera lens distortion	21

2.3.1	Optical distortion	21
2.3.2	Perspective Distortion	22
2.3.3	Rectilinear vs Curvilinear Lenses	23
2.3.4	Lens distortion correction	23
2.4	Court detection	24
2.4.1	Using Hough transform and Homography	25
2.4.2	Model based approach	26
2.5	Ball tracking	28
3	Research Design and Results	31
3.1	Available hardware	31
3.1.1	Cameras	31
3.1.2	Computational Resources	32
3.2	Data	32
3.2.1	Pose estimation	33
3.2.2	Annotations process	35
3.3	Player detection and tracking	36
3.3.1	Model	36
3.4	Court detection	39
3.4.1	Fish-eye distortion correction	39
3.4.2	Detecting the court	41
3.5	Ball trajectory	44
3.6	System architecture	44
4	Conclusions	45
4.1	Future work	46
	Bibliography	47

List of Tables

2.1	YOLO-V5 Mean Average Precision (mAP)	15
2.2	YOLOv6 Variants and Performance	15
3.2	Defisheye parameter's values.	39

List of Figures

2.1	Padel court dimensions	3
2.2	The six main padel strokes.	4
2.3	Example of the four camera placements. Red dots correspond to the cameras. . .	6
2.4	LeNet-5 layers.	8
2.5	Example of a Region-based Convolutional Neural Network (R-CNN) process. . .	9
2.6	Region Proposal Network illustration. [6]	10
2.7	Comparison of inference time speed of object detection algorithms.	10
2.8	Original Yolo preliminary architecture. [29]	12
2.9	Original Yolo preliminary architecture. [28]	12
2.10	Yolo progression timeline. [29]	13
2.11	Optimization methodologies experimented in YOLO-v4 via bag-of-specials. [29] .	14
2.12	YOLO-V5 compared between its variants and EfficientDet. [29]	15
2.13	YOLO-V6 architecture. [29]	16
2.14	YOLO-V6 performance vs YOLO-V5. [29]	16
2.15	Compound scaling on YOLO-V7. [29]	17
2.16	YOLO-V8 performance relative to its predecessors.	17
2.17	Intuitive example of transfer learning. [34]	18
2.18	SSD framework. [35]	19
2.19	SSD architecture. [35]	19
2.20	Different types of optical distortion.	22
2.21	Perspective distortion by adjusting the angle of view and the camera distance. .	22

2.22	Curvilinear vs Rectilinear lenses.	23
2.23	GeoNetS and GeoNetM system architecture. [43]	24
2.24	Application of the Hough transform to detect the lines of the court. The noisy lines got filtered after the application of the equations. [44]	25
2.25	Illustration of the proposed solution. [45]	26
2.26	Some of the system's detection on different courts with different conditions. [45]	27
2.27	Frames' label files. [47]	28
2.28	TrackNet architecture. [47]	29
2.29	Heatmap on the ball. [47]	29
3.1	Camera placement.	32
3.2	Example of one of the extracted frames.	33
3.3	Pose Estimation model architecture.	33
3.4	Confidence map for left should in a certain image.	34
3.5	Part Affinity maps for left shoulder and neck.	34
3.6	Openpose model output.	35
3.7	Creating annotations with CVAT.	35
3.8	Confusion matrix for the model's performance on the training data.	37
3.9	Confusion matrix of the model when performing inference on the testing data. .	37
3.10	ByteTrack detecting the different players.	38
3.11	The system detecting the players and the executed strokes.	38
3.13	Frame without distortion.	40
3.14	Frame after applying canny edge detector.	41
3.15	Probabilistic Hough lines transform on the frame.	41
3.16	Frame after merging lines.	42
3.17	The two points identified by the first line detected.	42
3.18	Frame after performing template matching.	43
3.19	The four corresponding points between the images.	43

3.20 Flow of the system.	44
----------------------------------	----

Acronyms

AI	Artificial Intelligence	ML	Machine Learning
CIoU	Complete Intersection over Union	MPS	Metal Performance Shaders
CV	Computer Vision	PANet	Path Aggregation Network
CNN	Convolutional Neural Network	RoI	Region of Interest
DFL	Distribution Focal Loss	R-CNN	Region-based Convolutional Neural Network
DL	Deep Learning	RPN	Region Proposal Network
E-ELAN	Extended Efficient Layer Aggregation Network	SIoU	SCYLLA-IoU
FAR	False Alarm Rate	SOTA	State of the Art
FPS	Frames Per Second	SPP	Spatial Pyramid Pooling
FrRCNN	Faster Region Convolutional Neural Network (CNN)	SS	Selective Search
GIoU	Generalized-IoU	SSD	Single Shot MultiBox Detector
GPS	Global Positioning System	SVM	Support Vector Machine
IA	Inteligência Artificial	TRDR	Tracker Detection Rate
LReLU	Leaky Rectified Linear Unit	VFL	Varifocal Loss
mAP	Mean Average Precision	YOLO	You Only Look Once

Chapter 1

Introduction

Real-time statistics systems are sophisticated systems that track the ball and the players using a number of technologies and produce statistics in real time.

One typical method is to track the ball and the players using a network of cameras. The cameras can be placed all over the court/field, and computer vision algorithms can be applied to the video footage to follow the ball and the players.

Once the players and the ball have been successfully detected and tracked, the system can generate multiple different statistics for different things. It can provide insight into the balls' like the speed, the trajectory, etc... and also information regarding the player's position, movement, stroke techniques, number of shots per point, etc.. The possibilities are endless.

It can also be used to create interactive visualizations, for example, provide the player the ability to interactively watch its movement trajectory between points.

As it is a technology that is being developed, players and coaches aren't really aware of how valuable a system like this can be. It can increase their productivity by decreasing time to analyse certain things such as player movement. Once the technology is getting deployed, the sport will probably have an increase in player base, with more people learning and getting better at the sport.

This these presents a solution for a system that provides these statistics, limited by the hardware available at the time of development. It will showcase the most important steps in order to create this system and how important is data for it to be successful.

1.1 Objectives

The main objective of this thesis is to produce tools that can help with the automatic analysis of padel matches. More specifically, it aims to:

- Detect the presence and location of padel players.
- Track the movement of padel players.
- Detect the presence of specific padel strokes performed by players.
- Automatically extract padel court parameters.

Chapter 2

Literature Review

This chapter will be an essential piece of this master's thesis as it will serve as knowledge base, containing important information such as the existing research within the application of Artificial Intelligence (AI) on the padel sport to provide insight from it in an autonomous way.

2.1 Padel Sport

Padel is a racket sport that was based on a not so popular variant of enclosed Tennis called Platform Tennis. In 1969 Enrique Corcuera, considered the inventor of padel¹ decided to adapt his Squash court based on Platform Tennis creating what he called "Paddle Corcuera".²

2.1.1 Padel court

It is a mix between Tennis and Squash and it's usually played in doubles on an enclosed court surrounded by walls of glass and metallic mesh. The court is one third of the size of a tennis court³. The court's dimensions can be seen on fig 2.1⁴:

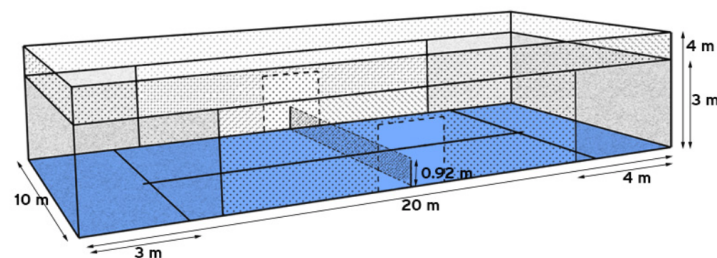


Figure 2.1: Padel court dimensions.

¹<https://www.palmaracketclub.es/wp-content/uploads/2020/12/EL-padel.pdf>

²<https://www.padel.ac/history-of-padel/>

³ https://www.head.com/en_GR/sports/padel/about

⁴ https://www.head.com/en_GR/sports/padel/about

The ball can spring back of any fence but can only hit the turf once before being turned back. When the ball bounces twice in the opponents' field, a point is attributed. Points are played before or after the ball bounces off the surrounding walls, adding a unique playstyle to the sport over conventional tennis.⁴

2.1.2 Padel strokes

It's fundamental to learn Padel basic main strokes, as these are what players use to maneuver the ball when playing with some of the popular ones are presented in fig 2.2⁵, with a brief explanation of each bellow:



(a) Serve



(b) Forehand



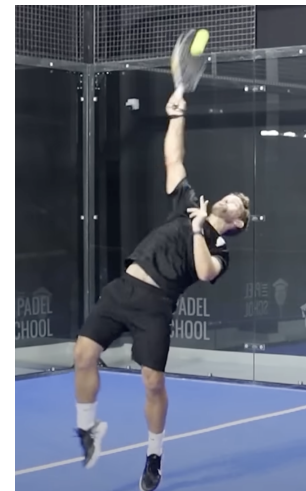
(c) Backhand



(d) Volley



(e) Bandeja



(f) Smash

Figure 2.2: The six main padel strokes.

⁵<https://www.youtube.com/@ThePadelSchool>

- (a) **Serve:** The serve is the initial shot that begins each point in padel. The player has two opportunities and it involves hitting the ball over the net to start the rally.
- (b) **Forehand:** The forehand stroke is a fundamental shot in padel, where the player strikes the ball on their dominant side using a forehand grip. Its objective is to keep the ball in play.
- (c) **Backhand:** The backhand stroke is executed on the non-dominant side of the player using a backhand grip and has the same objective as the forehand.
- (d) **Volley:** The volley is a shot where the player hits the ball before it bounces on the ground, often executed near the net. It's considered as one of the most attacking shots in padel.
- (e) **Bandeja:** The bandeja is a type of lob shot, typically used to place the ball deep into the opponent's court allowing the player to move forward and continue the attack at the net.
- (f) **Smash:** The smash is a powerful overhead shot, often used to finish a point.⁶

2.1.3 AI in padel sport

When **AI** comes to mind, one of the most immediate applications we can think of is the creation of player statistics. These statistics could be the number of times a player failed one point using some type of stroke like the number of times a player fails the first serve, a bandeja, what side of the field the player tends to fail more shots, etc...

Current technologies and solutions used in multiple sports like Global Positioning System (**GPS**) and wearable sensors make it possible to collect data regarding player's movements, including speed, distance covered, acceleration, and their positions along the court/field [?].

One of the most popular statistical uses of sports monitoring data is the enrichment of performance measurements. They are numerical measurements intended to characterize a certain athletic ability or performance output. Metrics based on tracking data stand out because they place a strong emphasis on performance's spatiotemporal aspects. Derived numbers can be determined directly from positional data, whereas model-based metrics are approximated through the use of a statistical model or algorithm. The use of quantitative techniques to extract tracking data from video is also an increasingly popular field of sports study. The search for more straightforward sports data capturing systems has become an appreciated study goal as a result of the development of the field of Computer Vision (**CV**). The ability of using analytical approaches to extract precise ball and player spatiotemporal data from broadcast footage is the ultimate objective of this line of research [1].

⁶<https://allforpadel.com/en/blog/10-essential-padel-strokes-n226>

In padel, data is usually extracted with the use of multiple cameras placed in strategic points across the court, normally one in each quarter of it so there's coverage of most of the angles. In fig 2.3⁷ we can visualize an example of this:

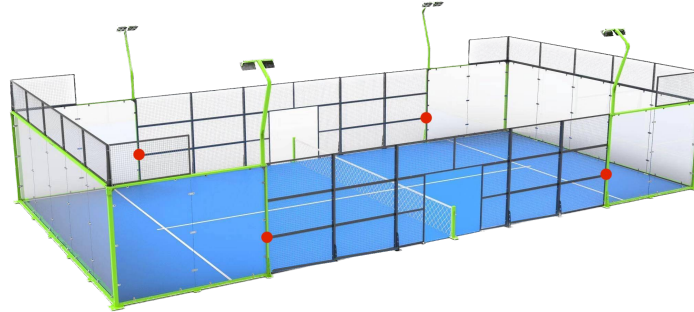


Figure 2.3: Example of the four camera placements.

Red dots correspond to the cameras.

Covering all the positions in the court from different angles makes it possible to create a projection of the game in another perspective, and with the addition of CV algorithms we can create systems that analyse the player's different strokes and techniques, providing feedback on what needs to be fixed, improving their performance so they can reach their max potential.

An additional application could be using it as a form of refereeing, confirming whether the doubtful shots were actually valid or not (i.e., Hawk-eye tennis system [2]). This can be done using CV techniques to track the ball and detect the court.

Overall, the application of AI in padel has lots of potential but it is worth mentioning that there are challenges and limitations when using systems like this such as the fact that the technology may not be affordable to some clubs and organizations and also it being under development and not being perfect.

⁷<http://en.hbqifan.com/panoramic-padel-court-p00267p1.html>

2.2 Player and Movement Detection

Player detection and tracking has been a hot topic for quite some time. This chapter serves the purpose of presenting some of the latest advances and research on the subject.

There are several types of algorithms that can be used for detecting and tracking players such as:

- Detection algorithms.
- Image Segmentation algorithms.
- Pose Estimation algorithms.

The recognition and location of many object instances in an image can be taught into **Detection** algorithms. These algorithms often produce a collection of bounding boxes that surround the recognized objects [3–6].

As for **Image Segmentation** algorithms (e.g., refs. [4, 7, 8]), each pixel in the image is labeled with the ID of the predicted instance class that corresponds to the pixel. The resulting masks reveal more about the detected items than just a box around them.

Pose Estimation algorithms calculate the positions of essential human body parts, such as the neck, hips, and feet. Top-down and bottom-up approaches are used to perform these estimations. Top-down approaches (e.g., refs. [5, 9–11]) identify person instances first, followed by their individual joints. An outside algorithm first locates the person and outputs the detection bounding boxes and then the method carries out single-person pose detection on the cropped and rescaled subimage. Although these approaches are typically slower and their efficiency is limited by the external detector, they are less sensitive to the scale variance of people. Bottom-up approaches (e.g., refs. [[12, 13]]) identify all of the image’s keypoints first, and then organize those keypoints into person instances. Although the majority of bottom-up solutions are extremely sensitive to scale fluctuation, these approaches are typically faster (even real-time [14]) because they behave end-to-end. [15]

In recent years, Convolutional Neural Network (**CNN**) oriented detectors surpassed State of the Art (**SOTA**) in object detection and tracking. Two of the most successful are Faster Region **CNN** (**FrRCNN**) and You Only Look Once (**YOLO**) (e.g. of application, refs. [16] [17], respectively).

2.2.1 CNN-based detectors

CNNs are a type of Deep Learning (**DL**) models created specifically for tasks involving images and visual input. They are inspired by the natural visual perception mechanism of the living creatures and were one of the biggest revolutions in the field of **CV**.

Although there are multiple variants available, these networks are usually made of by some basic components. One of the first **CNN** to be created, *LeNet-5* (refs., [18, 19]), was designed to perform handwritten digit recognition and is made up of three different kinds of layers:

- Convolutional layers.
- Pooling layers.
- Fully connected layers.

This network's architecture can be seen on fig. 2.4⁸:

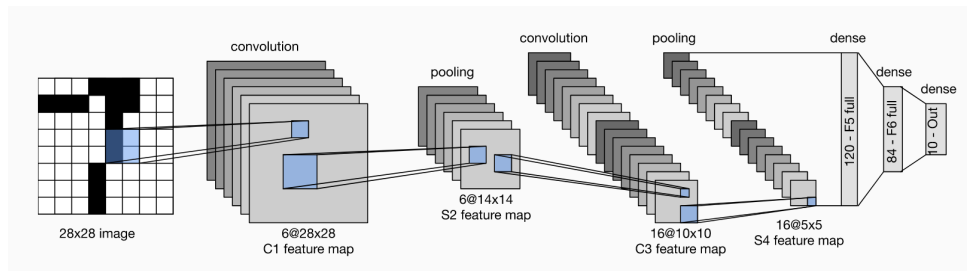


Figure 2.4: LeNet-5 layers.

The **convolutional** layer aims to learn feature representations of the inputs and is composed of multiple convolution kernels which are then used to process the different feature maps. Each neuron of a feature map is connected to a region of neighbouring neurons in the previous layer. By first convolving the input with a learnt kernel and then applying an element-wise nonlinear activation function to the convolution results, the new feature map can be generated. Some of the most common activation functions are sigmoid, tanh and ReLU [20].

The **pooling** layer serves the purpose to achieve shift-invariance by reducing the resolution of the feature maps, in another words, its reduces the number of connections between convolutional layers. Each feature map of a pooling layer is connected to its corresponding feature map of the preceding convolutional layer.

After several convolutional and pooling layers, there may be **fully-connected** layers which aim to perform high-level reasoning. To provide global semantic information, they connect every neuron in the current layer to every neuron in the previous layer. [21]

⁸https://www.researchgate.net/figure/LeNet-5-Dataflow-18_fig2_349031731

Object detection has been one of the most important topics in computer vision and had a fast progressive evolution in recent years. It lies on successfully recognize the presence of objects but also to precisely locate and classify them within a visual scene. [22]

With the success of AlexNet in 2012 [23], many other networks were created such as the Region-based Convolutional Neural Network (**R-CNN**) series [24]. It uses Selective Search (**SS**) to extract around 2000 bottom-up region proposals that are most likely to contain objects. These region proposals are then warped to a fixed size of 227 to 227, and a pre-trained **CNN** is used to extract features from them, with a binary Support Vector Machine (**SVM**) classifier used for detection [21]. This process can be visualized in fig 2.5⁹:

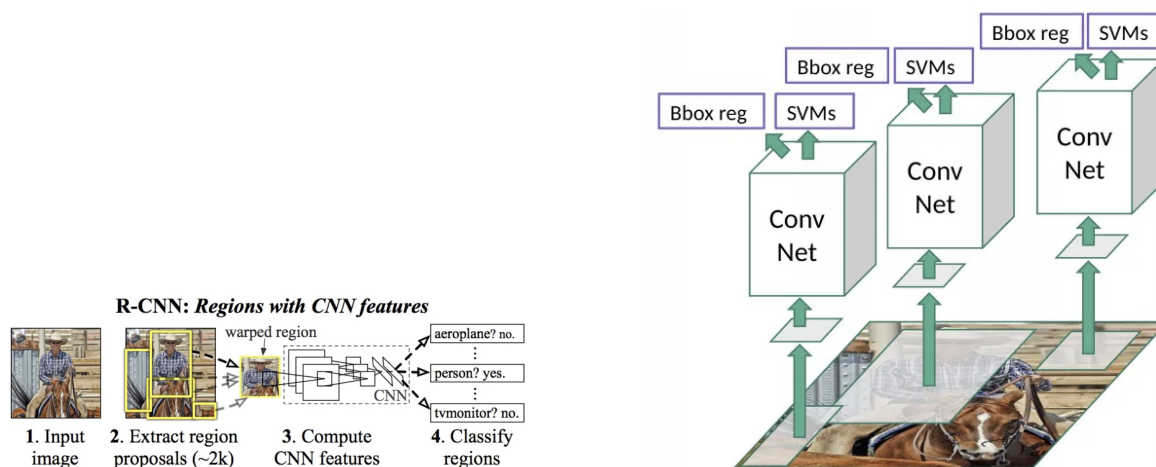


Figure 2.5: Example of a **R-CNN** process.

Although these were a huge jump, they faced some challenges:

- Having to classify 2000 regions would lead to a high training time.
- Not practical on a real time use as it would take almost one minute to inference one test image.
- Since **SS** is a fixed algorithm, the generation of bad candidate region proposals was possible as no training would be done until this stage.

In order to tackle these drawbacks, a new network called **Fast R-CNN** was built. The difference from the previous is that instead of feeding the region proposals to the **CNN**, the input image is passed to the **CNN** to generate a convolutional feature map, making it faster since the 2000 region proposals don't have to be fed to the convolutional neural network every time. From the convolutional feature map, the region of proposals is identified and warped into squares, and by using a Region of Interest (**RoI**) pooling layer they get reshaped into a fixed size so they

⁹<https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>

can be fed into a fully connected layer. From the **RoI** feature vector, a softmax layer is used to predict the class of the proposed region and also the offset values for the bounding box. [25]

These two methods use, as previously mentioned, **SS** to determine the region proposals. As a slow and very time consuming algorithm, **Faster R-CNN** [6] denied the need for its implementation and instead made the proposals be learnt by the network. The process is roughly the same but using a Region Proposal Network (**RPN**) where it fixes the regions as a grid of n by n . It needs to run fewer number of times as compared to **SS** [25]. In fig 2.6 we can see the **RPN**:

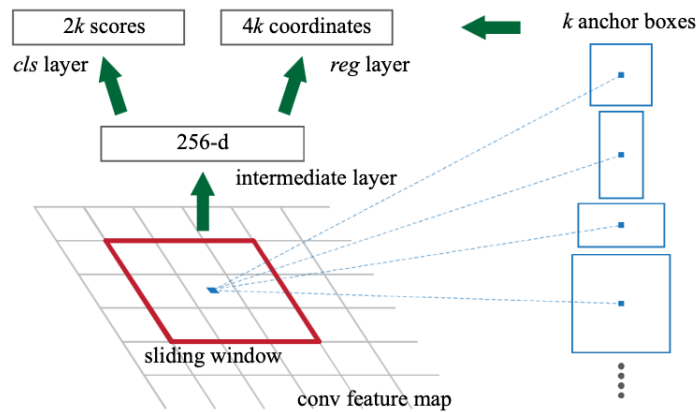


Figure 2.6: Region Proposal Network illustration. [6]

This proved to increase the speed on which the network trained and performed inference. Speed comparison between these three version can be seen on fig 2.7⁹:

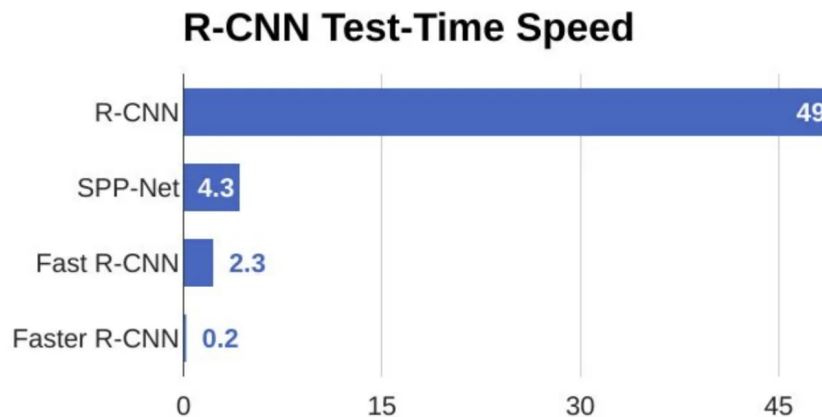


Figure 2.7: Comparison of inference time speed of object detection algorithms.

We can acknowledge that **Faster R-CNN** is the overall better architecture from this family of models, reducing training and inference time, consequently decreasing costs and making it possible to run with a less powerful hardware and with a better performance.

As for the loss function for the **Faster R-CNN** it is a combination of two loss terms:

- **Classification** loss: It is the error between the predicated class probability and the ground truth class probability. It is calculated using the cross-entropy function which is standard for multi-class classification problems.
- **Localization** loss: It is the error between the predicted bounding boxes and the actual bounding boxes. It is calculated using the smooth L1 loss function as it is less sensitive to outliers than the normal L1 and L2 functions. [26, 27]

The smooth L1 loss function is defined as follows:

$$\text{smooth_l1_loss}(x, y) = \begin{cases} 0.5 * (x - y)^2 & \text{if } |x - y| < 1 \\ |x - y| - 0.5 & \text{otherwise} \end{cases} \quad (2.1)$$

The loss function for **Faster R-CNN** can be defined by the following equation:

$$\text{loss} = \text{classification_loss} * \alpha + \text{localization_loss} * \beta \quad (2.2)$$

where:

- `classification_loss` is the cross-entropy loss function.
- `localization_loss` is the smooth L1 loss function
- λ_a and λ_b are hyperparameters that control the importance of each loss term.

Another very successful family of models called **YOLO** is known for its precision and accurate object detection with really fast inference time. The original **YOLO-V1**, based on the Darknet framework had 2 sub-variants. The first had 24 convolutional filters followed by 2 fully connected layers as shown in fig 2.8. Of these 24 convolutional layers, only four are followed by max-pooling layers. The model's authors have initially trained and fine-tuned it with first twenty layers followed by an average pooling layer and a fully connected layer on ImageNet 2012 dataset for approximately one week. Afterwards, for the second version inspired by the GoogleLeNet architecture [28], the model is further fine-tuned for the object detection task and consisted of only nine convolutional layers hosting fewer filters each with a sequence of (1×1) convolutional layers being implemented for reducing the resultant feature space from the preceding layers [29]. Leaky Rectified Linear Unit (**LReLU**) was used as the activation function for all the convolutional and dense layers except the last layer which has a linear activation function. [30]

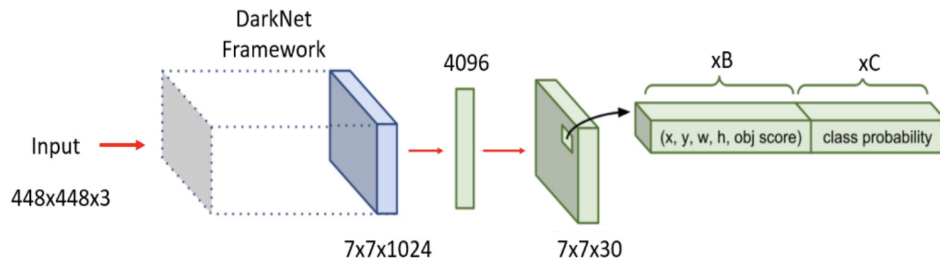


Figure 2.8: Original Yolo preliminary architecture. [29]

The model treats the problem as a regression dividing the image into an $S \times S$ grid and for each predicts bounding boxes, confidence for those boxes, and class probabilities. This process can be visualized in fig 2.9:

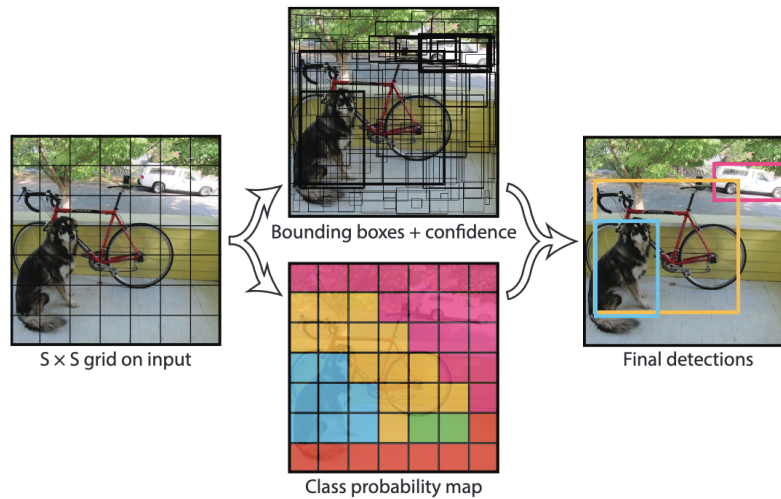


Figure 2.9: Original Yolo preliminary architecture. [28]

This original version had several limitations such as being constrained on bounding box predictions since each grid cell only predicts 2 boxes and can only have one class, restricting how many objects the model can anticipate. Consequently, the model struggles with small objects that appear in groups. Another limitation is that it struggles to generalize to objects in new or irregular aspect ratios. [28]

YOLO-V2 came to fix some of these issues, introducing batch normalization, leading to faster training. This led to an increase in Mean Average Precision (**mAP**) by 2% relative to the original YOLO. It replaced the fully connected layer responsible for predicting bounding boxes by adding anchor boxes for bounding box predictions. Anchor boxes are a list of predefined dimensions (boxes) aimed at best matching the objects of interest [29]. A couple more features can be seen of the following list:

- Better bounding box regression for improved accuracy.
- Introduction of anchor boxes to handle different object sizes.
- Multi-scale training and detection.
- Support for the detection of a large number of object categories (9000).

It achieved a performance of 76.8 **mAP** at 67 FPS Frames Per Second (**FPS**) and 78.6 **mAP** at 40 **FPS**.

YOLO-V3, its successor, included bigger modifications. It had a hybrid design that took elements from Darknet, YOLO-V2, and the ResNet idea of residual networks. This made it possible for the gradient flow from shallow layers to deeper layers, which helped preserve fine-grained characteristics. Darknet has 53 layers already for feature extraction, and an additional stack of 53 layers was added for the detection head, making a total of 106 convolutional layers. The design made predictions at three separate degrees of granularity for better output, facilitating multi-scale detection as well as raising the likelihood of detecting smaller items. [29]

From this version on, the main author had discontinued further work from the YOLO project, with the **CV** community then keeping the research and experimenting. The model's successor's can be seen on fig 2.10:

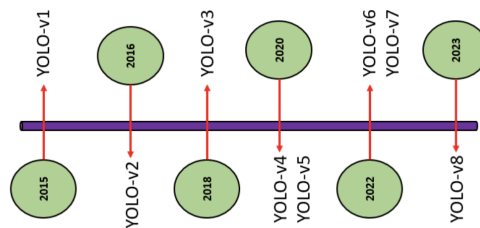


Figure 2.10: Yolo progression timeline. [29]

YOLO-V4 was essentially the culmination of a broad range of object detection methods that had been evaluated and improved in order to provide a real-time, portable object detector.

It came with the following features and upgrades:

- Use of CSPDarknet53 architecture for feature extraction.
- Implementation of Path Aggregation Network (**PANet**) for feature pyramid scaling.
- Use of the Spatial Pyramid Pooling (**SPP**) module for capturing multi-scale information.
- Leverage of the Complete Intersection over Union (**CIoU**) loss for better localization.
- Use of Multiple optimization techniques to improve performance which can be seen on fig 2.11.

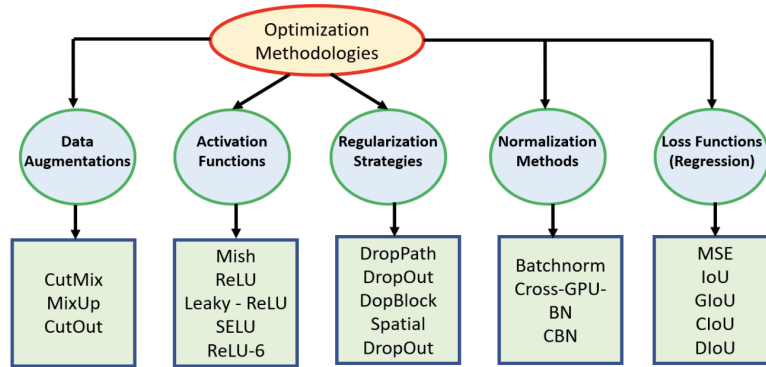


Figure 2.11: Optimization methodologies experimented in YOLO-v4 via bag-of-specials. [29]

As for **YOLO-V5**, it was the first to be entirely written in PyTorch instead of Darknet, since the latter was not meant to be used as a production environment with a significantly smaller number of subscriptions because of configurability challenges. PyTorch provided an established environment with a wider subscription base among the computer vision community and provided the supporting infrastructure for facilitating mobile device deployment. Some of the main upgrades and features of **YOLO-V5** were:

- Introduction of a streamlined architecture for faster training and inference.
- Introduction of model variants with different trade-offs between speed and accuracy shown in table 2.1.
- Focus on deployment in practical applications and ease of use.
- Independent Development from previous versions.

Table 2.1: YOLO-V5 **mAP**

Model	Average Precision (@50)	Parameters (M)	FLOPs (B)
YOLOv5s	55.8%	7.5	13.2
YOLOv5m	62.4%	21.8	39.4
YOLOv5l	65.4%	47.8	88.1
YOLOv5x	66.9%	86.7	205.7

The different variants performance can be observed in the fig 2.12, where the model is also compared to another **SOTA** detector EfficientDet [31]. We can clearly confirm **YOLO-V5** superiority:

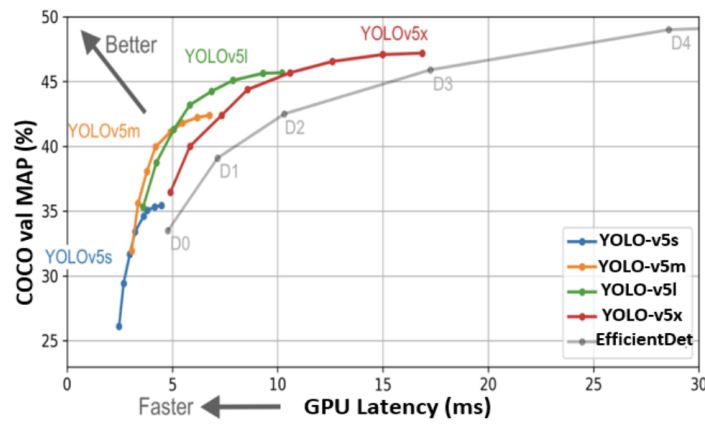


Figure 2.12: YOLO-V5 compared between its variants and EfficientDet. [29]

YOLO-V6 was released with big improvements in every aspect such as speed and accuracy compared to its previous version. It also came in different variants trading of between speed and performance as shown in table 2.2:

Table 2.2: YOLOv6 Variants and Performance

Variant	mAP 0.5:0.95 (COCO-val)	FPS Tesla T4	Parameters (Million)
YOLO-v6-N	35.9% (300 epochs)	802	4.3
YOLO-v6-T	40.3% (300 epochs)	449	15.0
YOLO-v6-RepOpt	43.3% (300 epochs)	596	17.2
YOLO-v6-S	43.5% (300 epochs)	495	17.2
YOLO-v6-M	49.7%	233	34.3
YOLO-v6-L-ReLu	51.7%	149	58.5

This incredible boost in performance can be justified by the changes made to the architecture of the model such as:

1. In contrast to its predecessors, **YOLO-V6** opts for an anchor-free approach, making it 51% faster when compared to anchor-based approaches.

2. A revised reparametrized backbone and neck, proposed as *EfficientRep* backbone and Rep-PAN neck was introduced, namely, up to and including YOLO-V5, the regression and classification heads shared the same features. It implements the decoupled head as shown in fig 2.13, inducing an increase on the performance.
3. The loss function is divided into two different functions. Varifocal Loss (VFL) is used as the classification loss and Distribution Focal Loss (DFL), along with SCYLLA-IoU (SIoU)/Generalized-IoU (GIoU) as regression loss. This helps balancing the learning signals from both positive and negative sample types, which is shown to be particularly efficient when the ground truth box boundaries are blurred.

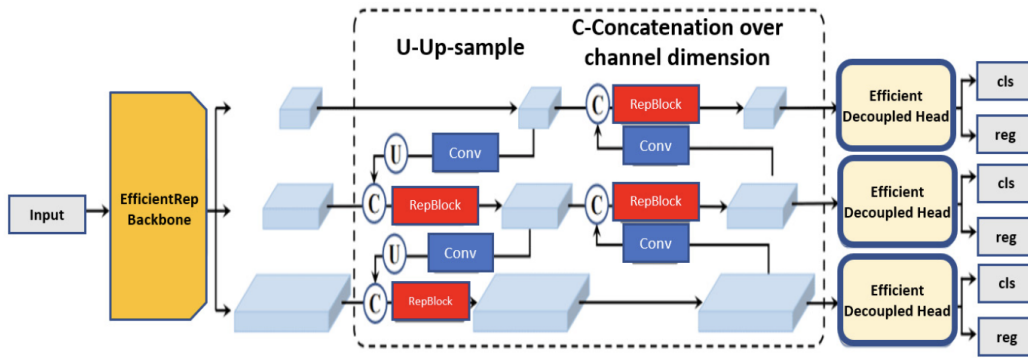


Figure 2.13: YOLO-V6 architecture. [29]

Comparing the performance of YOLO-V6 with its predecessors on the benchmark COCO dataset, we can verify its superiority, achieving a higher mAP at various FPS. The results are presented in fig 2.14:

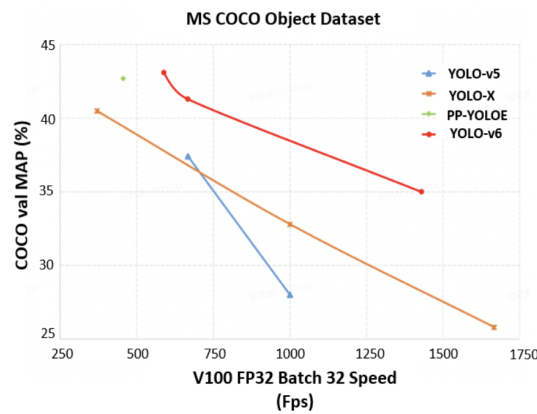


Figure 2.14: YOLO-V6 performance vs YOLO-V5. [29]

YOLO-V7 came with some architectural changes, namely, the implementation of the Extended Efficient Layer Aggregation Network (E-ELAN) on the backbone of the network, being its design impacted by factors directly related to speed and accuracy like memory access cost, gradient

path and input/output channel ratio. Another change was the implementation of compound model scaling as we can see in fig 2.15. This approach enables the width and depth to be scaled in coherence for concatenation-based networks, retaining optional network design while scaling for numerous sizes. This is because certain applications focus on accuracy while others demand speed.

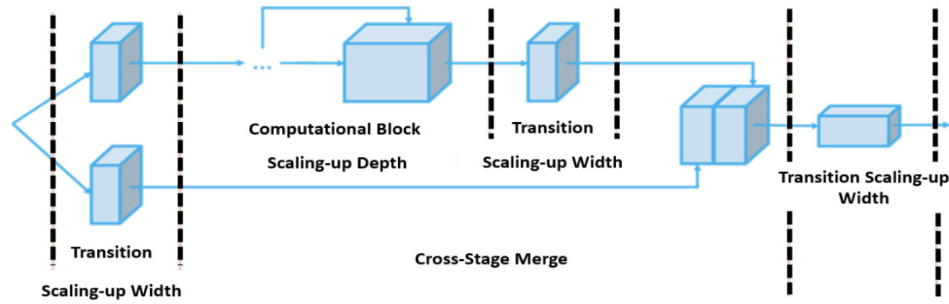


Figure 2.15: Compound scaling on YOLO-V7. [29]

The authors mentioned that the model uses gradient flux propagation techniques to identify which internal network modules should make use of re-parameterization procedures. Additionally, something to consider is that the model wasn't designed to be ran on a CPU.

With the release of **YOLO-V8** by *Ultralytics*¹⁰, YOLO reached once again state of the art object detection and tracking performance. Although no paper is available to date, we can, after observing fig 2.16¹⁰, come with the conclusion that the upgrades made to the architecture prioritized the edge device deployment.

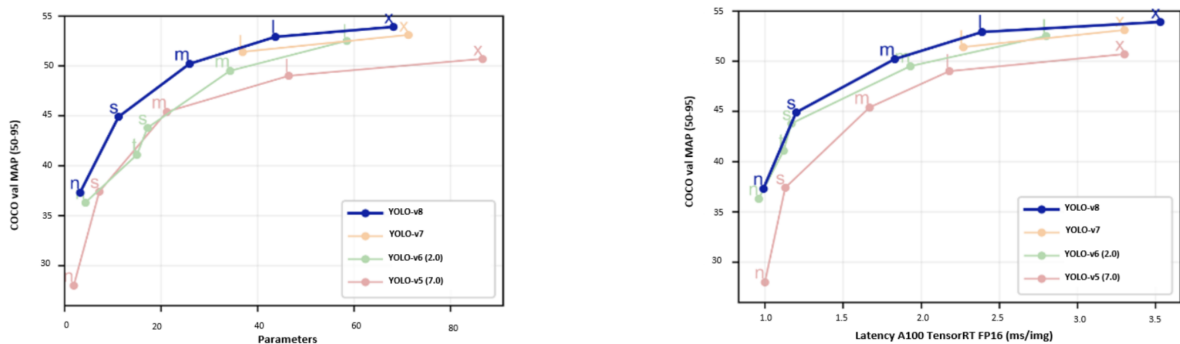


Figure 2.16: YOLO-V8 performance relative to its predecessors.

¹⁰<https://github.com/ultralytics/ultralytics>

As for the loss function for YOLO-V8, a combination of three loss measures are used:

- **Classification** loss: It calculates the error between the predicted class probabilities and the ground truth class probabilities. It uses the focal loss function, addressing the problem of class imbalance [32].
- **Localization** loss: It measures the error between the predicted bounding box to the actual bounding box. It uses the **GIoU** loss function, which is more robust and accurate than the L1 and L2 loss functions [33].
- **Objectness** loss: It measures the model's level of certainty regarding the existence of an object within a bounding box. It uses the binary cross-entropy loss function.

The final equation can be represented by:

$$\text{loss} = \text{classification_loss} * \lambda_c + \text{localization_loss} * \lambda_l + \text{objectness_loss} * \lambda_o \quad (2.3)$$

Where:

- classification_loss is the focal loss function.
- localization_loss is the **GIoU** loss function.
- objectness_loss is the binary cross-entropy loss function.
- $\lambda_c, \lambda_l, \lambda_o$ are hyperparameters that control the importance of each loss term.

2.2.1.1 Transfer Learning

An important concept when using model's like YOLO or Faster R-CNN is transfer learning, which is using a model that was used for one task as the starting point for a model on a second task. The knowledge gained from a previous task helps improving the generalization of the model [34]. A simple example of transfer learning is shown on fig 2.17:

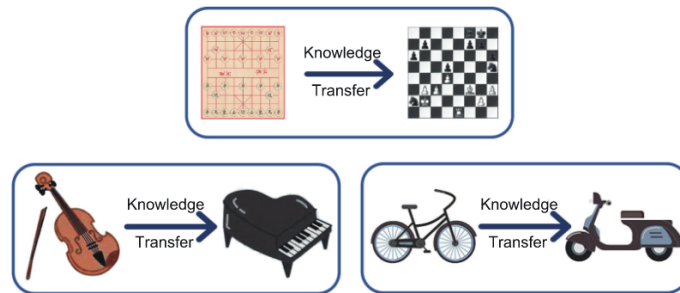


Figure 2.17: Intuitive example of transfer learning. [34]

Finally, another model that should be mentioned is Single Shot MultiBox Detector (**SSD**) [35], another one-stage method for object detection, just like YOLO. Since R-CNN and Fast R-CNN are two-stage algorithms, they have to go through an incremental pipeline to first calculate region proposals and afterwards classify each of them in the second stage. SSD, on the other hand, in order to recognize objects in a picture, streamlines the pipeline and employs a single deep neural network, fully omitting proposal generation. Multiple default bounding boxes of varying ratios and sizes make up the output space of SSD. The idea behind these bounding boxes is comparable to that of R-CNN anchors. [36] SSD framework can be seen in fig 2.18:

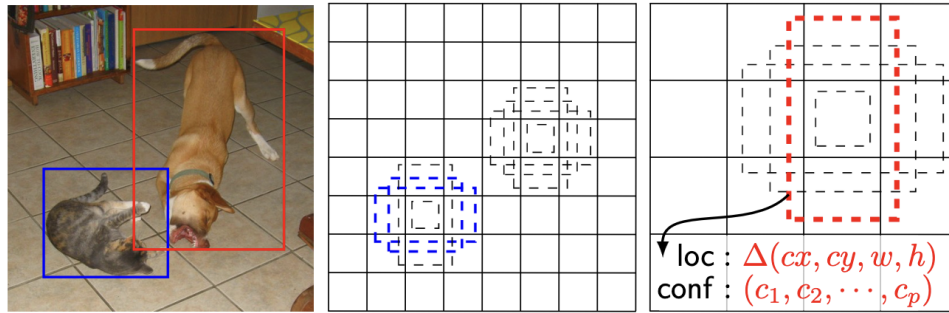


Figure 2.18: SSD framework. [35]

SSD is quite easy to train, making it simple to incorporate into detecting systems. The creation of bounding boxes and the extraction of feature maps, which are also known as default bounding boxes, are critical components of its design. The network calculates loss by comparing the predicted classes' offsets and the default bounding boxes with the ground truth values for the training samples, applying a new filter for each iteration. All the parameters are changed using the back-propagation algorithm and the estimated loss value. The overall objective loss function is a weighted sum of the **localization** loss and the **confidence** loss. In this approach, SSD is able to discover the best filter structures, which generalize the provided training samples in order to reduce the loss value and achieve high accuracy during the evaluation phase. These structures can accurately identify the object features [25]. Its architecture can be seen on fig 2.19:

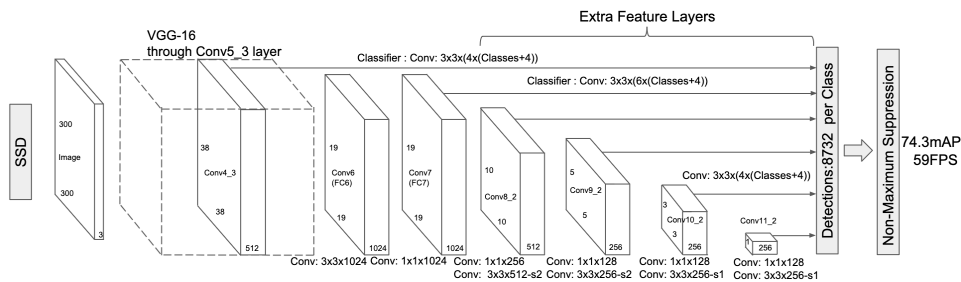


Figure 2.19: SSD architecture. [35]

2.2.2 Histogram based detectors

Histogram-based detectors are a sort of object detector that locates things by analyzing the histogram of a picture. Using histograms, you may visualize how the pixel values in an image are distributed. A histogram-based detector can ascertain whether or not an object is present in an image by comparing the histogram of the image to the histogram of a known object. [37]

They have a number of **advantages** over other types of object detectors:

- They are very efficient, as they only require a single pass through the image, making them really good for real time object detection.
- They are very robust to noise and illumination changes since they do not rely on specific features of the object.
- They can be used to detect a wide variety of objects, including cars, faces, etc...

But they also have some **disadvantages** like:

- They can be sensitive to the size and orientation of the object being detected. This is because they depend on comparing the histogram of the image to the histogram of a known object.
- They need to be trained on a large number of examples of the object being detected hence being difficult to train for complex objects.

Histogram-based detector can be used to perform Histogram-based object tracking on objects like vehicles [38], mobile robots [39] and many other objects.

The histogram distance between a target and a detected bounding box can be calculated as a metric for filtering region proposals. The distance between the target models and the detected bounding boxes of the color histograms can be defined in a variety of ways. The *Bhattacharyya* similarity coefficient can be applied to define the distance between HSV and HOG histograms respectively [40], as shown in equation :

$$BC(p, q) = \sum_{x \in X} \sqrt{p(x)q(x)} \quad (2.4)$$

Additionally, it is possible to use the Euclidean distances between the object's estimated locations and the centers of the bounding boxes just like in [41]. Another method for comparing a detected histogram with a model histogram is color histogram intersection [42]. [36]

2.3 Camera lens distortion

When light entering a camera lens is not bent in a perfectly straight line, a phenomenon known as camera lens distortion occurs. In the resulting image, straight lines in the real world may appear bent or warped. Depending on how severe the distortion is, lens distortion can even be seen in the image's center.

2.3.1 Optical distortion

Typically, the goal of rotationally symmetric optical systems is to create an image that closely resembles the object in terms of geometry. There are a few cases where geometric criteria are purposefully broken, such as with some specific systems like fish-eye lenses. Ideally, this function is carried out in accordance with perspective projection geometry.¹¹

There are three types of optical distortion:

- Barrel.
- Pincushion.
- Moustache.

Barrel distortion refers to a type of aberration in which straight lines are bent inward into the shape of a barrel. Wide-angle lenses frequently exhibit barrel distortion because the lens' field of view is significantly broader than the size of the image sensor, so it must be "squeezed" to fit. As a result, straight lines clearly curl inward, especially when they approach the frame's farthest edges. Most wide angle prime lenses and many zoom lenses with relatively short focal lengths frequently exhibit barrel distortion. The distance between the camera and the subject might affect the amount of distortion.

In **Pincushion** distortion straight lines are bent outward from the center, which is the exact opposite of barrel distortion. Due to the image magnification rising further from the optical axis and toward the margins of the frame, this form of distortion is frequently observed on telephoto lenses. This time, the field of view must be "stretched" to fit because it is less than the image sensor's size. As a result, the corners of straight lines seem to be dragged upwards.

In essence, **Moustache** distortion combines pincushion distortion and barrel distortion. In the middle of the frame, straight lines appear to be bent inside, then they appear to be curved outward towards the corners. Because of its complicated qualities and potential for discomfort, mustache distortion is frequently referred to as a "complex" distortion. Many vintage lenses and some contemporary lenses exhibit mustache distortion.¹²

¹¹<https://www.image-engineering.de/library/image-quality/factors/1062-distortion>

¹² <https://photographylife.com/what-is-distortion>

An example of these three types of distortion can be seen on fig 2.20¹²:

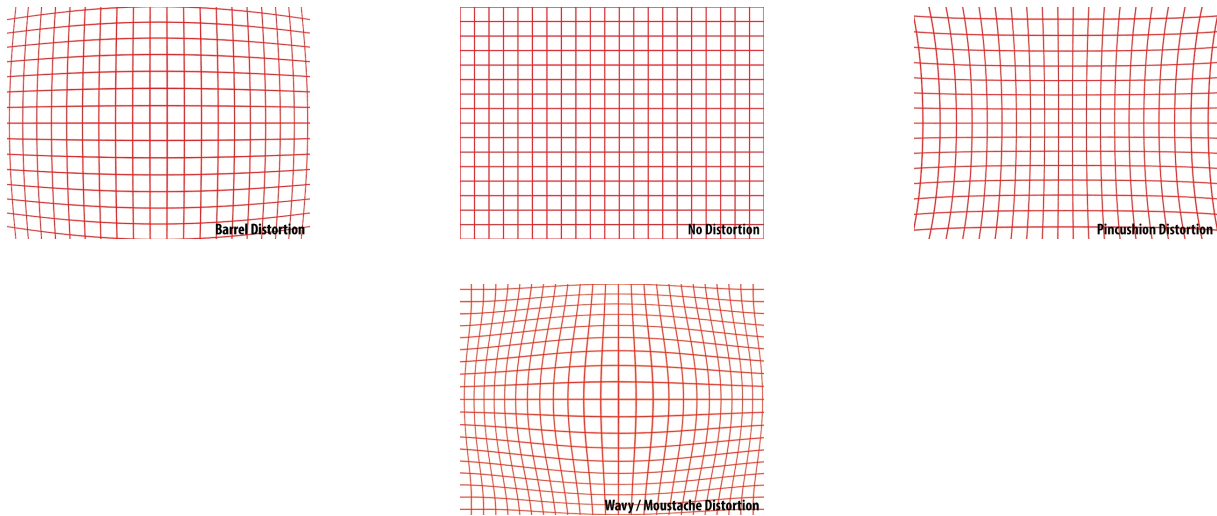


Figure 2.20: Different types of optical distortion.

2.3.2 Perspective Distortion

Perspective distortion is another sort of distortion that is frequently found in pictures. It is not a lens error since, unlike optical distortion, it has nothing to do with the optics of the lens. If the subject is too close to the camera, it may appear excessively huge or distorted when compared to the things in the background because three-dimensional space is projected onto a two-dimensional image.¹² This effect can be seen on fig 2.21¹³:

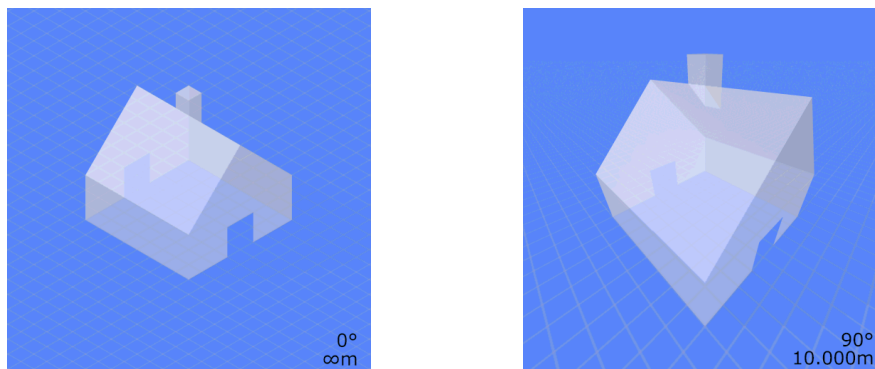


Figure 2.21: Perspective distortion by adjusting the angle of view and the camera distance.

¹³ https://en.wikipedia.org/wiki/File:Camera_focal_length_distance_house_animation.gif

2.3.3 Rectilinear vs Curvilinear Lenses

Some lenses, such as *fisheye* lenses, are optically engineered to be curvilinear, which means they provide straight lines without bending them (similar to how humans see). Especially at the boundaries of the frame, rectilinear lenses typically stretch objects to make them appear straight. Contrarily, curvilinear lenses drastically distort images by curving straight lines rather than stretching anything. An example of these types of lens can be seen on fig 2.22¹²:



Figure 2.22: Curvilinear vs Rectilinear lenses.

2.3.4 Lens distortion correction

Deep Learning has been used by researchers to develop techniques to fix these distortions in recent years.

In [43], they used CNN's to predict the displacement field between distorted images and restored images after training them on a sizable synthetic distortion dataset. They have first designed a network to first predict the model parameter p^B . This parameter is then used to generate the flow $F = M^B(p^B)$ in the network. This network, called *GeoNetS*, designed for single-model distortion estimation was only able to capture a specific distortion type with a distortion model at a time. They have created a second network, *GetNetM*, for multi-model distortion estimation in order to overcome these restrictions. It is hard to train a multi-model network with the model restrictions, though, because the distortion model and the parameters p^B can differ significantly between kinds. The network is trained to simultaneously classify the distortion type and regress the distortion flow without model limitations.

The loss function for the GeoNetS network was the distance between the estimated distortion flow and the ground truth flow:

$$*\theta^B = \arg \min \mathcal{L}(\mathcal{N}^B(I; \theta^B), F) \quad (2.5)$$

$$N^B(I; \theta^B) = M^B(n^B(I; \theta^B)) \quad (2.6)$$

where the component used to regress the distortion parameter is represented by n^B , a sub-network of N^B . Their loss function was determined endpoint error which was the Euclidean distance between the anticipated flow vector and the ground truth, averaged across all pixels. As for the GeoNetM they have used the endpoint error loss L_{flow} in the regression branch, and a cross entropy loss in the classification branch. The total loss can be defined as follows:

$$*\theta = \arg \min(L_{flow} + \lambda L_{class}) \quad (2.7)$$

The while system can be seen in fig 2.23, including both networks' architecture:

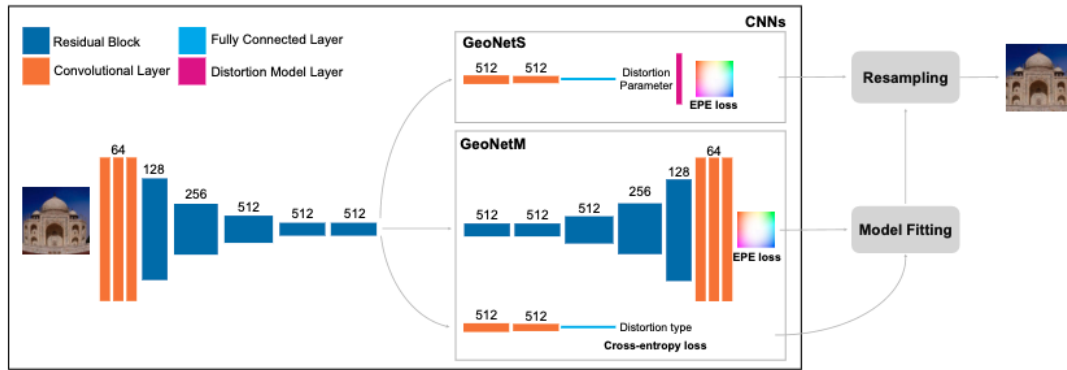


Figure 2.23: GeoNetS and GeoNetM system architecture. [43]

2.4 Court detection

Court detection is a CV task that consists on finding and locating courts in images and videos. For a range of applications, such as sports analytics, court detection is crucial.

The process of court detection can be approached in a variety of ways. Making use of a template matching algorithm is one typical strategy. The input image is compared to a template image of a court as part of template matching algorithms. The method will return the location of the court in the input image if the two photos are sufficiently comparable.

Using a machine learning algorithm that has been trained on a dataset of photos and videos that contains labels for the locations of courts is another popular method of court detection. Once trained, the system may be used to find courts in fresh pictures and movies.

Another simpler approach is to utilize something like Hough transform to detect the court's lines and perform homography to determine how well a projection of the lines of a reference court overlap with the a binary image of the frame.¹⁴

¹⁴<https://blog.ml6.eu/improving-tennis-court-line-detection-with-machine-learning-90f82dcd1d>

2.4.1 Using Hough transform and Homography

A rather simple approach could be the one presented in [44], where they used Hough transform to detect the court's lines.

They use a self-adaptive threshold to first transform the video stream's frames to grayscale images, and then from those to binary images. The binary image is then subjected to edge detection to produce an edge image. The next step is to choose a Sobel operator to identify edges in the horizontal and vertical axes, respectively. To detect the lines on the edge image, they finally use the Hough transform. Other irrelevant lines are also found near edges and are classified as noise lines. They used two formulas based on line end-points to filter out these lines, as follows:

$$\lambda_1 * w \leq x_{li} \leq (1 - \lambda_1) * w \quad \text{and} \quad \lambda_2 * h \leq y_{li} \leq (1 - \lambda_2) * h \quad i = 1, 2 \quad (2.8)$$

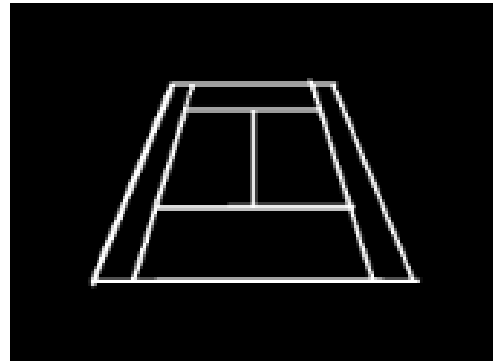
$$|x_{l1} - x_{l2}| \geq \lambda_3 w \quad \text{and} \quad |y_{l1} - y_{l2}| \geq \lambda_4 w \quad (2.9)$$

Where w and h indicates, respectively, image width and height, x_{li} and y_{li} expresses horizontal and vertical coordinates of line l , and $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are proportion parameters.

The effect of these equations can be see on fig 2.24:



(a) Before equations.



(b) After equations.

Figure 2.24: Application of the Hough transform to detect the lines of the court. The noisy lines got filtered after the application of the equations. [44]

2.4.2 Model based approach

In [45], they offer a method to track the court from a tennis video and estimate the homography required to transform the image coordinates of the video frame to actual coordinates. Firstly, the frames containing the court are detected by applying a court view detection process, in which they then proceed with the initialization step. To find straight lines in the image, they begin by segmenting the scene and extracting every pixel of the court lines. A set of interest points are then found and with the use of RANSAC algorithm they estimate the geometric transformation between the image coordinate and the model. For each transformation they estimate the error projection between the white lines in the image and the projected model and they try to optimize this error by decreasing it. Once the court has been identified for the first frame, the camera's motion is checked. If the camera is moving, the interest spots are tracked using the Lukas-Knades tracker, and Ransac calculates the homography for each frame. Finally, a model-based correction method is used to minimize the projection error utilizing the Levenberg-Marquardt (LM) algorithm in order to guarantee good performance. But with a fixed camera, the model-based correction technique is only necessary. The system's can be visualized in fig 2.25:

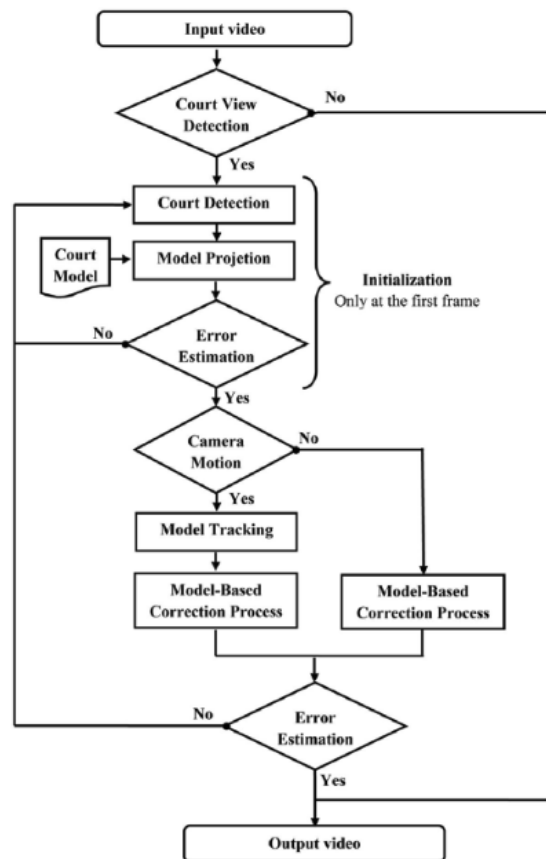


Figure 2.25: Illustration of the proposed solution. [45]

To evaluate their results they have three sets of the tracking evaluation metrics based on frame-based metrics provided by Bashir and Porikli [46]. These metrics are Tracker Detection Rate (**TRDR**), False Alarm Rate (**FAR**) and accuracy and are defined, respectively by:

$$TRDR = \frac{TP}{GT} \quad (2.10)$$

$$FAR = \frac{FP}{TP + FP} \quad (2.11)$$

$$Accuracy = \frac{TN + TP}{TF} \quad (2.12)$$

where:

- **TP**: represents the number of frames where the court is correctly detected.
- **TN**: represents the number of frames where the system correctly detects there is actually no court.
- **FP**: represents the number of frames where the court wrongfully detected the court.
- **GT**: represents the ground truth.
- **TF**: represents the total number of frames.

The system obtained an impressive **TRDR** and Accuracy of 0.9733 and 0.9831, respectively. Some examples of its detection can be seen on fig 2.26:

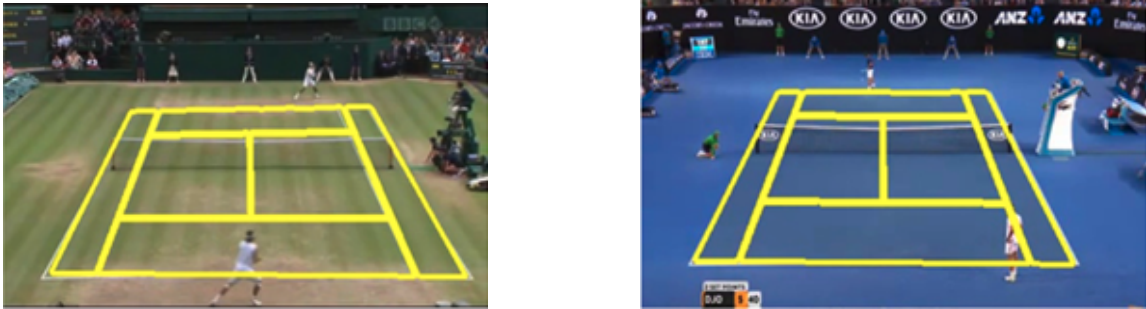


Figure 2.26: Some of the system's detection on different courts with different conditions. [45]

We can explore many solutions to perform court detection but overall, it's a challenging task:

- A court can be played in a variety of lighting situations, from direct sunlight to dim interior lighting. A court detection system may find it challenging to precisely identify the court boundaries as a result.

- The system may have trouble determining the court boundaries due to the shadows cast by players, equipment, and other objects.
- Distinct sports may employ different court markings, which can change over time and become faded or worn. The system may find it challenging to precisely identify the court boundaries as a result.
- The accuracy of the system might also be impacted by the camera angle being used to view the court. For instance, it may be challenging for the system to precisely identify the court boundaries if the camera is not perpendicular to the court.
- The system also must be able to operate in real time, meaning it must be able to quickly and precisely identify the court's boundaries.

2.5 Ball tracking

Ball tracking consists in constantly identify the ball's position and trajectory across the court in a video sequence. It can be performed in different ways such as using a color-based object tracking algorithm, or a motion-based tracking algorithm. We can also include deep learning models to make the tracking even more precise. A state of the art network called TrackNet [47] has shown to have really good results when tracking the ball.

They created a system made up of a single-frame version and a multi-frame version, that by training on the dynamic behavior of the trajectory, it can accurately track swiftly moving and small objects.

Their dataset consisted on frames from the 2017 Summer Universiade, with a resolution of 1280 x 720 and a frame rate of 30 frames per second. The characteristics of each frame are as follows: "Frame Name", "Visibility Class", "X", "Y" and Trajectory Pattern". This can be seen in fig 2.27:

```

      ⋮
0008.jpg, 2, 727, 447, 0
0009.jpg, 1, 735, 457, 0
0010.jpg, 1, 722, 433, 1
0011.jpg, 1, 707, 403, 0
      ⋮
0029.jpg, 1, 555, 220, 0
0030.jpg, 1, 550, 218, 2
0031.jpg, 1, 547, 206, 0
      ⋮

```

Figure 2.27: Frames' label files. [47]

Additional 16118 frames were gathered to increase the variability of the training dataset. These images were taken from nine videos that were shot on various tennis courts, including hard courts, grass courts, etc... The model was anticipated to recognize the tennis ball at various courts by learning diverse circumstances, increasing robustness.

A **CNN** is the first component of its architecture, which is then followed by a deconvolutional neural network. A heatmap representing the position of the object is produced over a series of frames and by learning the trajectory pattern, moving object identification with several input frames can be improved. A probability detection heatmap is generated and a 2d Gaussian distribution centered on the tennis ball serves as its ground truth. The networks' architecture can be seen on fig 2.28:

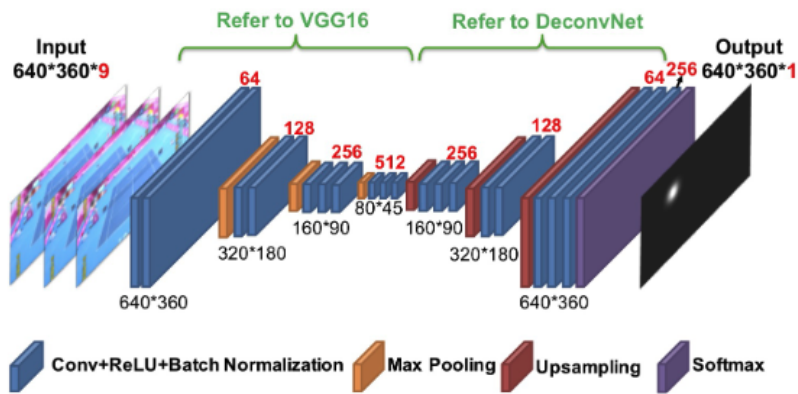


Figure 2.28: TrackNet architecture. [47]

The heatmap is expressed in terms of the ball center (x_o, y_o) :

$$G(x, y) = \left[\left(\frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_o)^2 + (y-y_o)^2}{2\sigma^2}} \right) (2\pi\sigma^2 * 255) \right] \quad (2.13)$$

where the first factor is a Gaussian distribution with variance of σ^2 and the second simply scales the value between 0 and 255. It can be seen in fig 2.29:

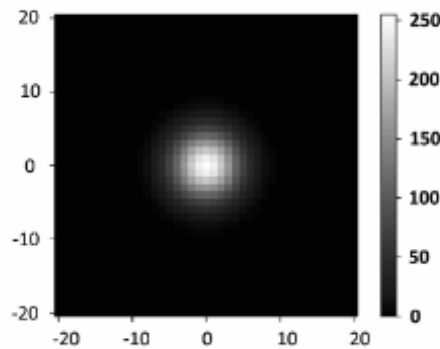


Figure 2.29: Heatmap on the ball. [47]

Training TrackNet’s dynamic object tracking capabilities can be further enhanced by successive input frames, especially for small, blurry or occluded objects. The performance in terms of precision, recall and F1-measure are defined, respectively by:

$$Precision = \frac{\# TruePositives}{\# TruePositives + FalsePositives} \quad (2.14)$$

$$Recall = \frac{\# TruePositives}{\# VC1 + VC2 + VC3} \quad (2.15)$$

$$F1 - measure = \frac{2 * (Precision * Recall)}{Precision + Recall} \quad (2.16)$$

where VC corresponds to the visibility class on a certain level. TrackNet using 2 consecutive frames as input obtained a precision of 99.8%, a recall of 96.6% and a F1-measure of 98.2%, beating all other **SOTA** models such a as the Archanas’s algorithm.

Chapter 3

Research Design and Results

In this chapter, the work and the solutions implemented to tackle the different problems involved in creating statistics for a Padel game will be presented. The main problem will be described as a whole and what resources were available, how was the data creation process and how were surpassed some of the main challenges, with a possible solution for the integration of all the pieces of the puzzle into one system. The results will also be presented.

3.1 Available hardware

The hardware available, from cameras to computational power available was an important factor as it would constraint or limit any possible work.

3.1.1 Cameras

Starting with the available cameras, there were two cameras per court, one in each side, as illustrated in fig 3.1¹:

These cameras are from *Pro Vision* and have the following relevant specs:

- **Sensor:** 1/3" 2MP Sensor.
- **Selectable Output:** 1080P AHD/TVI/CVI/CVBS.
- **Lens:** 2.8-13.5mm Mega-Pixel Vari-Focal.
- **IR Range:** 25m (3 Array LED).
- **3D-DNR/BLC.**

¹<https://afpcourts.com/en/padelcourts/adidas-fx/>

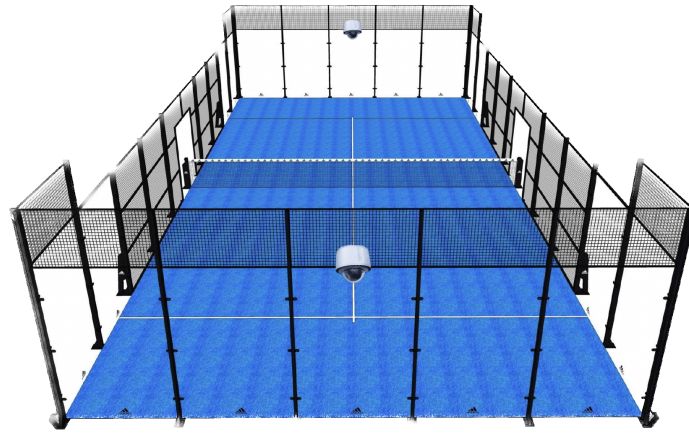


Figure 3.1: Camera placement.

These cameras produced some distortions as they had wide angle lenses. This distortion will be later on corrected as the lines of the court have to be straight in order to detect them.

3.1.2 Computational Resources

A MacBook Pro M2 was used to develop all the work and had the following relevant specs:

- 8-core CPU
- 10-core GPU
- 16-core Neural Engine
- 8GB RAM

3.2 Data

Data was the most important thing for the well execution of the system as it would impact its performance.

In order to create a model to detect the different strokes, images of these padel strokes being executed were needed, so a training session with some padel players was set up. There were sets of two minutes of non-stop stroke execution for each of the movements. This way, the accessibility of the data would be more organized and straightforward, consuming less time to create them.

An important detail taken in consideration was that two of these players were professional ones while the other two were amateurs. This lead to different techniques making it possible to optimize the training and validation sets so the model could generalize more. If it could detect

the amateur techniques while being trained with the professional ones, this would suggest it would generalize well.

The session resulted in a 30 frames 1080p video of 10 minutes which were split into multiple videos, each containing the execution of only one stroke. This was important as it would consume less time when creating the actual data. An example of one of the frames extracted from the video can be seen on fig 3.2:



Figure 3.2: Example of one of the extracted frames.

3.2.1 Pose estimation

After gathering the frames, they were processed to include the body pose keypoints, which would be used to create a skeleton, that would be an image feature on the training set. The model used was presented by *LearnOpenCV*² and is available on [Github](#).

The model consisted of 10 VGGNet layers that would create features maps, and of a two step multi-stage CNN. The model's architecture can be seen on fig 3.3²:

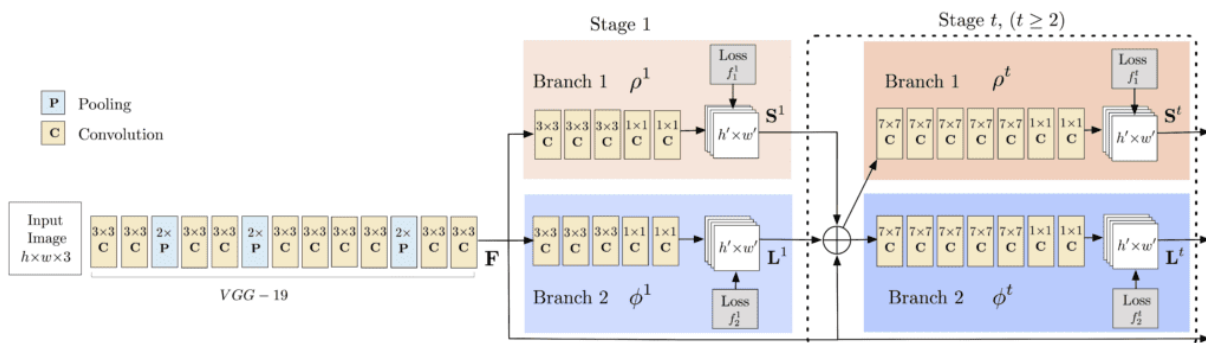


Figure 3.3: Pose Estimation model architecture.

²<https://learnopencv.com/multi-person-pose-estimation-in-opencv-using-openpose/>

2D Confidence maps of body part locations were predicted on the first step. This produced a grayscale image which had a high value at locations where the probability of a certain body part is high. An example of the confidence map for the left shoulder can be seen in fig 3.4²:

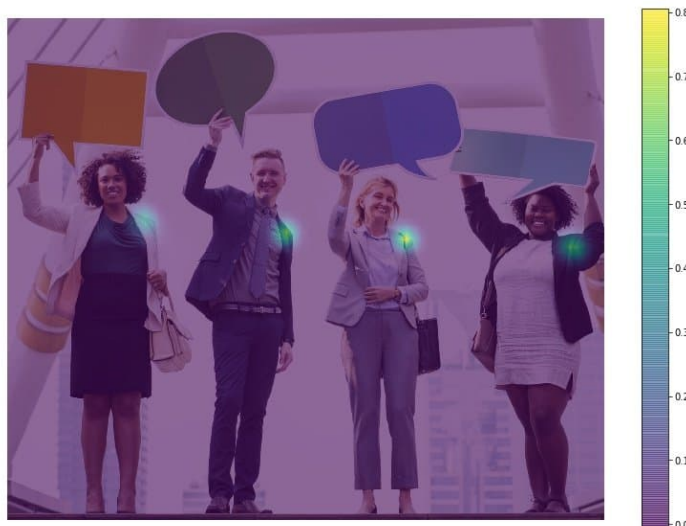


Figure 3.4: Confidence map for left should in a certain image.

As for the second step, a set of 2D vectors of Part Affinities would be predicted, which would encode the degree in which keypoints were associated. In fig 3.5² we can visualize this between the left shoulder and the neck:

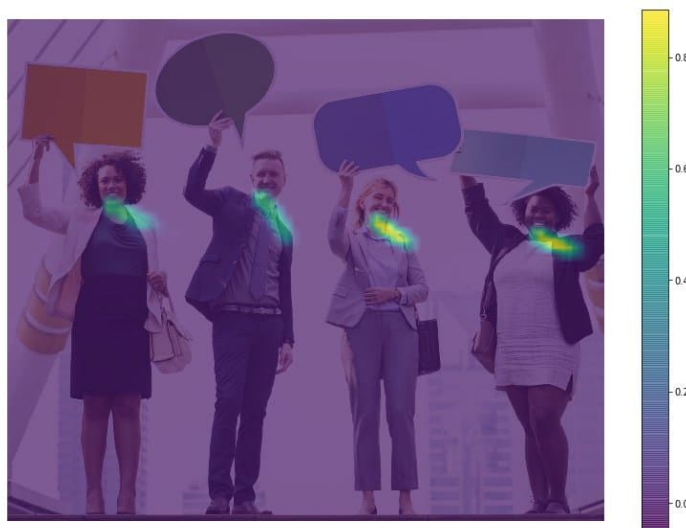


Figure 3.5: Part Affinity maps for left shoulder and neck.

Applying the model into one of the actual frames would lead to the creation of people's skeletons if the respective keypoints were detected. This effect is show in fig 3.6:



Figure 3.6: Openpose model output.

3.2.2 Annotations process

After processing all frames with the output of the OpenPose model, an annotation tool was searched in order to create the annotations that would be part of the dataset. A tool called [CVAT](#) was found to contained all the necessary basic features necessary to create data, in its free version. It could process large amounts of frames (a single video), multiple classes could be defined to create annotation for them and also, it could export the annotations in multiple formats, including YOLO format.

After creating the annotations for the **right volley** and **smash** strokes, as show in fig 3.7, a process to split the data into training, validation and test was made and ended up with 189 images for train, 51 for validation and 50 for test. As mentioned previously, the training set was based on professional players while the validation and testing ones where based on amateur players. This was the dataset was created and was ready to be used for model training.



Figure 3.7: Creating annotations with CVAT.

3.3 Player detection and tracking

The process of detecting the players, tracking them and identify all the strokes executed will be explained in this section.

3.3.1 Model

The model chosen was **YOLO-V8** as it was State of the Art (**SOTA**) for object detection in real time and also had integrated trackers which would facilitate the posterior work. Also, the variant chosen was the **V8s** as we prioritized speed over performance as the model would be deployed in a real time scenario. At the time the model was trained, there was no integration of Metal Performance Shaders (**MPS**) on the *Ultralytics* YOLO framework so it was done using CPU, however as for inference, MPS was utilized for faster processing speed. MPS presents a new device to map computational graphs and primitives from Machine Learning (**ML**) onto the incredibly effective MPS Graph architecture.³

A pre-trained version of the model was used to perform transfer learning, initializing the weights of the model according to its originally trained classes such as person, bicycle, car, etc... Although more classes could be trained, only 2 classes were used for fine-tuning the model, **right volley** and **smash** as it was enough to test the system. As for the training itself, the most important hyperparameters and their corresponding values are shown in table 3.1b as well as the results of the training on the table 3.1a:

Epoch	precision	recall	mAP50	mAP50-95
1	0.323	0.793	0.439	0.307
2	0.860	0.782	0.897	0.751
3	0.920	0.845	0.936	0.809
4	0.888	0.860	0.955	0.838
5	0.948	0.862	0.980	0.877
6	0.969	0.965	0.990	0.882
7	0.727	0.946	0.976	0.823
8	0.958	0.981	0.992	0.811
9	0.960	0.887	0.980	0.873
10	0.942	0.941	0.988	0.757
11	0.958	0.974	0.988	0.841

(a) Model's performance on the training data.

Hyperparameter	Value
Model	Yolov8s
Epochs	300
Patience	5
Batch	16
Imgsize	(1182, 673)
Device	CPU
Workers	8
Optimizer	SGD
AMP	True
IOU	0.7
LR0	0.01
LRF	0.01
Momentum	0.937
Weight decay	0.0005

(b) Model's hyperparameters.

³<https://pytorch.org/docs/stable/notes/mps.html>

The training took 11 epochs to finish by early stoping since there was no significant improvements on the last 5 epochs. Also, not much tuning was made on these hyperparameters as most of the default values provided already a decent performance atleast on the training data, with no signs of overfitting. The model's confusion matrix can be seen on the fig 3.8:

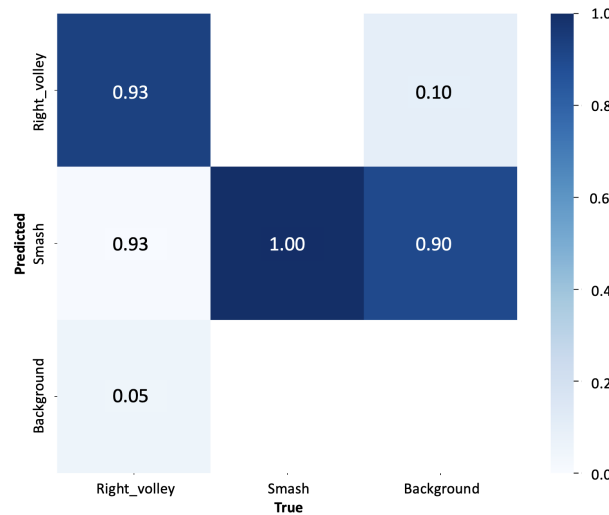


Figure 3.8: Confusion matrix for the model's performance on the training data.

We can verify that the model detected correctly **Right Volley** 93% of the times, detecting right volley roughly 10% of the times when there was no stroke being executed, accounting some false positives. As for the **Smash**, it predicted smash when it was actually right volley 1% of the times. It had 100% accuracy when detecting smash but also detected smash 90% of the times when there was no smash, meaning it was detecting a lot of false positives.

When testing the model on the testing data, it came up with a mAP50-95 of 0.79 for the right volley and 0.633 for smash. It was not the best performance but it was decent. The confusion matrix for the testing data can be seen of fig 3.9:

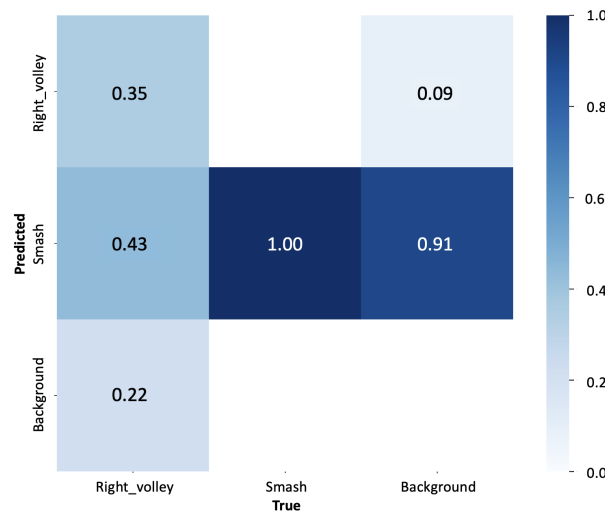


Figure 3.9: Confusion matrix of the model when performing inference on the testing data.

We can acknowledge that a lot of false positives smashes were being detected and right volley was not being detected 65% of the times. Future work would imply working around the training data in order to fix this issues.

Finally, after finishing the training, another model was used to track the players, this time, the original trained YOLO-V8s which could detect persons. The idea was to use our trained model to detect the strokes and the original model to track the players. The tracker utilized was **ByteTrack**, which has state of the art performance and is already integrated in the YOLO framework. Afterwards, the detections were filtered by a confidence level of 97% to avoid false positives. An example of the tracker in action can be seen on fig 3.10:

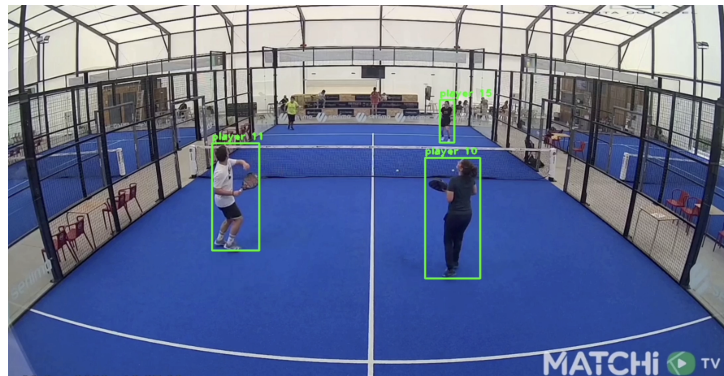


Figure 3.10: ByteTrack detecting the different players.

After putting the two models together the outcome can be seen in fig 3.11:

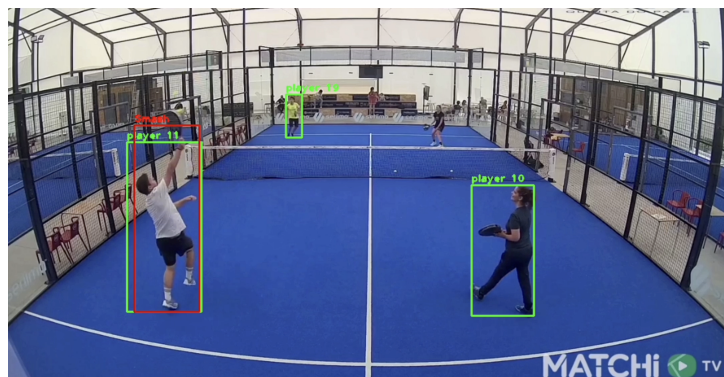


Figure 3.11: The system detecting the players and the executed strokes.

The system was identifying the players and the strokes with a decent accuracy. However, some false positives would should up when detecting the strokes. This was a consequence of creating annotations for each stroke including a lot of amplitude on the movement. Since some of the strokes are similar, this lead to an unwanted detection of these false positives.

Another possible issue was the tracker losing some player because of occlusion or because of him leaving the camera view. This could be fixed by using a re-identification algorithm by extracting information relative to each player, like the clothes, to fix a detecting id to it, this way even if the player left the camera field of view the tracker could associate it with a previous id.

3.4 Court detection

As the players and strokes were being identified, another important piece of the system was the detection of the system. This was important because we need to know when there was a point, and for that we needed to locate the ball on the court.

3.4.1 Fish-eye distortion correction

Since our cameras had some distortion, this needed to be corrected. This was because a **Probabilistic Hough** lines transform would be used to detect the lines of the court.

A very recent developed toolkit called *Defisheye*⁴ was used to correct the distortion. It had a command-line interface for handling massive amounts of images and a graphical user interface created especially for visual analysis to find the ideal lens parameters to remove the distortion.

In table 3.2 are presented the parameters chosen after analysing different values for the cameras used followed by a brief description of each of them:

Parameter	Value
dtype	equalarea
format	fullframe
fov	180
pfov	120

Table 3.2: Parameter's values.

- **dtype**: Type of fisheye lens: linear (equidistant), equalarea (equisolid), orthographic and stereographic.
- **format**: Format of the fisheye lens image: circular (image fills a circle that spans the minimum of the width or height) or fullframe (image spans a circle that spans the diagonal dimension)
- **fov**: Fisheye field of view (aperture) in degrees.
- **pfov**: Perspective field of view (aperture) in degrees.

⁴<https://github.com/duducosmos/defisheye>

Before applying the tool, the image needed to double in size otherwise the tool would chop a good part of the image and the effect would not be the desired one. An example can be seen on figs 3.12a and 3.12b:



(a) Original image.



(b) Image after padding.

After applying the tool, the end result had straight lines instead of curved ones and can be visualized in fig 3.13:



Figure 3.13: Frame without distortion.

Although the result is not perfect and has less quality over the original, it was enough to detect at least some of the most important lines. The court is almost as visible as the original and the players can all be seen and detected.

The idea was to process each frame to remove its distortion before even pushing them into the system, however this would bring challenges when included in a real time context. As these processes are mainly graphical, an optimized solution would be required. Using the CPU, it took 1 second to process all the operations to correct the distortion of a single frame.

3.4.2 Detecting the court

The process of detecting the court would begin by turning the frame into a gray-scale frame to remove additional information that would not be necessary. A bilateral filter would then be applied followed by a Canny edge detector. This way we would obtain a frame only with the edges of all the objects in it. The produced effect can be seen in fig 3.14:

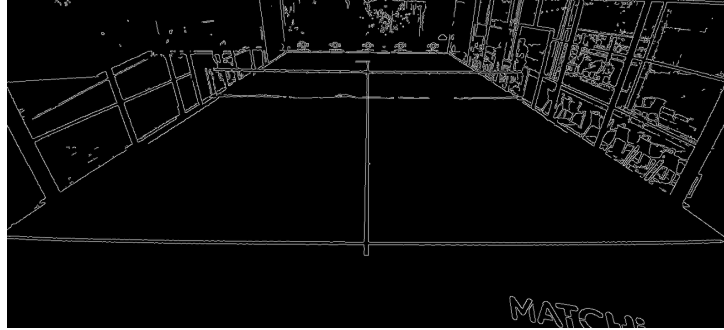


Figure 3.14: Frame after applying canny edge detector.

The lines were then made thicker with the use of a dilation filter and a probabilistic Hough lines transform was applied to the frame in order to detect lines, as shown in fig 3.15:

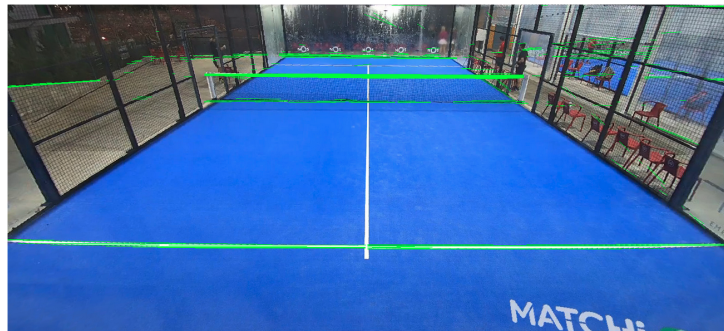


Figure 3.15: Probabilistic Hough lines transform on the frame.

Vertical lines were filtered and only lines with a length higher than 40px were left. After leaving the frame with much less noise, the lines were merged. At first the solution presented in [48] was implemented, but the results turned out to be difficult to interpret, so a simple algorithm was found to merge these lines according to the minimum distance and angle between them⁵, as show in fig 3.16:



Figure 3.16: Frame after merging lines.

This way we ended up with the first line of the court being detected while the court's back-end line was not, this being a consequence of reduced resolution, even though it wouldn't be identified on the original frame either. Facing this challenge, a possible solution was to identify the 2 points in which this first line intersects the walls which was obtained by the lines coordinates, and having a template matching algorithm that would connect these 2 points with the court's back-end ones. The first two points can be seen on fig 3.17:



Figure 3.17: The two points identified by the first line detected.

⁵<https://stackoverflow.com/questions/45531074/how-to-merge-lines-after-houghlinesp>

Using a template matching we could get the coordinates of the back-end points, as show in fig 3.18:



Figure 3.18: Frame after performing template matching.

After having the coordinates of the four points, we could then calculate the homography matrix. This matrix would describe how points in one image are transformed to be located in another image from another perspective. This would be useful to obtain the coordinates of the ball into an aerial view of the court. This aerial view would correspond to the real coordinates. An example of this can be seen on fig 3.19:

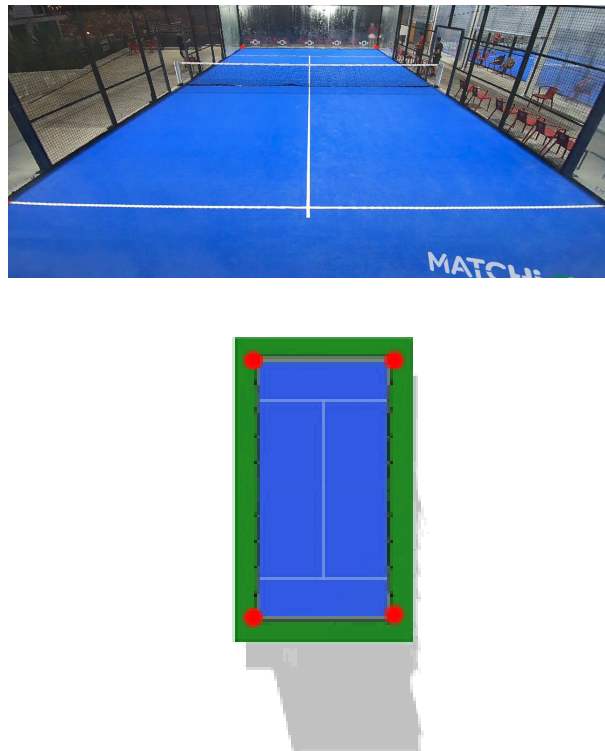


Figure 3.19: The four corresponding points between the images.

This process was based on [49] work and would provide a way of converting the coordinates of the ball on the inputted frames to coordinates on the above plane, making it possible to check

when there would be a point.

3.5 Ball trajectory

An histogram based algorithm was tested first but it didn't detect the ball as intended, a direct consequence this being a really small object and in some frame being occluded or ghosted. The main approach would be to use **TrackNet** to identify and track the ball on the court but it couldn't be implemented.

3.6 System architecture

An important topic to discuss is how would the system work in a deployed scenario, and there are some possibilities regarding the implementation. With a focus on real time execution, the system would have to be optimized to reduce delays with fast response between traded information. If two or more gpus were available, we could use one of them to process the player tracking and detection and the other one to process the ball trajectory. Since the latter would be done using a deep learning system, a lot of computational power would be required.

There could be one database which would track what happened for a specific frame in a specific time, which player did it, etc... This would imply lots of writing calls to the database, and meanwhile, another calls for reading were being made, to check for newer happenings. The statistics would then be made according to the information on the database. The diagram presented in the fig 3.20 exemplifies the flow of this process:

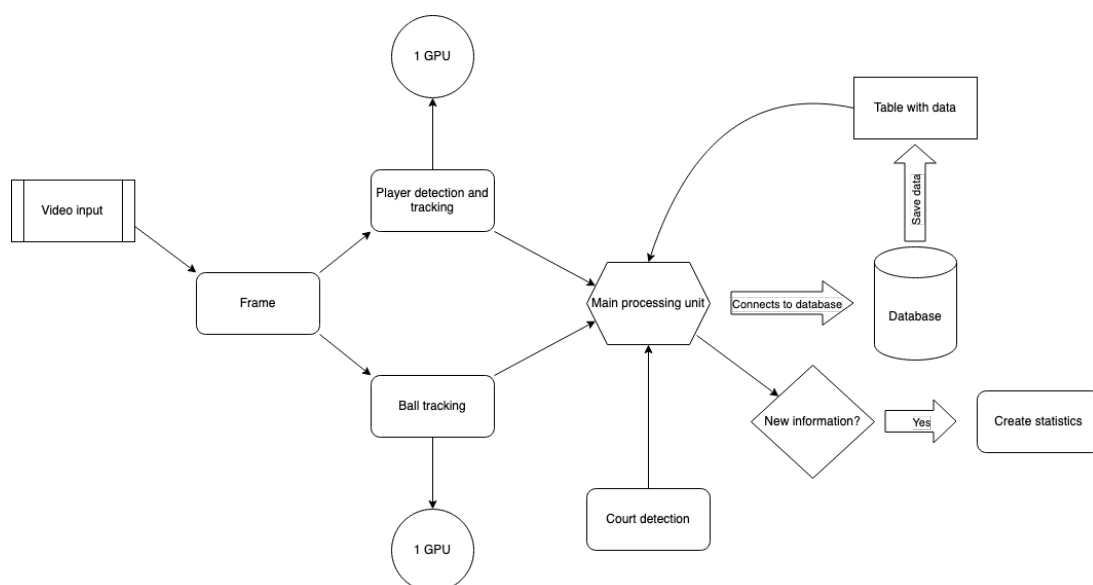


Figure 3.20: Flow of the system.

Chapter 4

Conclusions

In this thesis a system that could provide real time statistics on a game of padel was designed, with some of its core features being implemented. It showcases that the creation and processing of the necessary data is an extensive and time consuming process since it is not available everywhere.

The player detection and tracking was successful, but with more data it could probably achieve a better performance, avoiding false positives and being more precise.

As for the court detection system, the correction of the distortion proved essential to the detection of the court's lines. However, the fact that this process involved a lot of computational power, it could possibly be impractical on real time scenario. Also, with the reduction on resolution, the use of a template matching was indispensable as there was no way to detect the furthest lines with the Hough transform. With that in mind, calculating the homography matrix would set the conversion between the frames' coordinates into another image perspective coordinates. This would then lead to the the gathering of the ball's position in real coordinates.

Overall, the creation of the system proved to be a challenging goal as the hardware available was not the best, the extraction and creation of the data consumed a lot of time, and other factors such as to how to deploy it, even though it was not deployed.

4.1 Future work

Future work would include finishing the ball tracking with TrackNet, and also another approach to detect the player strokes.

Encoding the information of the pose estimation such as the angles between the different keypoints of the human body and length of the line that would connect them, we could possibly train a rather simpler model such as an XGBoost to learn that a set of combinations between keypoints would correspond to a specific stroke. This would make it possible not only to inference in a much faster way but also would provide the ability to characterize the strokes in their different moments, such as the beginning, the moment when the racket touches the ball, etc.. This would open a new way of providing players the analysis of their execution in a more precised way.

In summary:

- Ball tracking implementation with TrackNet.
- Creation of newer annotations with less amplitude of movement.
- Implementation of another approach to detect the different strokes. The idea consists in using vectors to store the information retrieved by the pose estimation model, and use it to create tabular data. The vector would contain information on the length of the body's identified keypoints, the angles between them, etc... transforming the problem into a classification problem with the following advantages:
 - It would certainly be faster than any CV approach as we could use, for example, an XGBoost model to predict.
 - It would also provide a much more precised way of distinguish the different strokes, avoiding the appearance of false positives.
 - It would introduce newer possibilities, like technique analysis on the different levels of amplitude, etc...

Bibliography

- [1] Stephanie A. Kovalchik. Player tracking data in sports. *Annual Review of Statistics and Its Application*, 10(1):677–697, 2023.
- [2] NEIL Owens, C Harris, and C Stennett. Hawk-eye tennis system. In *2003 international conference on visual information engineering VIE 2003*, pages 182–185. IET, 2003.
- [3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.
- [4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: High quality object detection and instance segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 43(5):1483–1498, 2019.
- [5] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *Proceedings of the European conference on computer vision (ECCV)*, pages 466–481, 2018.
- [6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.
- [7] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. pages 4510–4520. IEEE Computer Society, 12 2018.
- [8] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, et al. Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4974–4983, 2019.
- [9] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VIII 14*, pages 483–499. Springer, 2016.

- [10] Junjie Huang, Zheng Zhu, Feng Guo, and Guan Huang. The devil is in the details: Delving into unbiased data processing for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5700–5709, 2020.
- [11] Feng Zhang, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. Distribution-aware coordinate representation for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7093–7102, 2020.
- [12] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5693–5703, 2019.
- [13] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S Huang, and Lei Zhang. Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5386–5395, 2020.
- [14] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE transactions on pattern analysis and machine intelligence*, 2016.
- [15] Mohammadreza Javadiha, Carlos Andujar, Enrique Lacasa, Angel Ric, and Antonio Susin. Estimating player positions from padel high-angle videos: Accuracy comparison of recent computer vision methods. *Sensors*, 21, 5 2021.
- [16] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016.
- [17] David Acuna. Towards real-time detection and tracking of basketball players using deep neural networks. *Neural Information Processing Systems*, 2017.
- [18] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.
- [19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [20] Yingying Wang, Yibin Li, Yong Song, and Xuwen Rong. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences*, 10(5):1897, 2020.
- [21] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

- [22] Dulari Bhatt, Chirag Patel, Hardik Talsania, Jigar Patel, Rasmika Vaghela, Sharnil Pandya, Kirit Modi, and Hemant Ghayvat. Cnn variants for computer vision: History, architecture, application, challenges and future scope. *Electronics*, 10(20):2470, 2021.
- [23] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.
- [24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [25] Shrey Srivastava, Amit Vishvas Divekar, Chandu Anilkumar, Ishika Naik, Ved Kulkarni, and V Pattabiraman. Comparative analysis of deep learning image detection algorithms. *Journal of Big data*, 8(1):1–27, 2021.
- [26] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [27] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.
- [28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [29] Muhammad Hussain. Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection. *Machines*, 11(7):677, 2023.
- [30] Tausif Diwan, G Anirudh, and Jitendra V Tembhurne. Object detection using yolo: Challenges, architectural successors, datasets and applications. *multimedia Tools and Applications*, 82(6):9243–9275, 2023.
- [31] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.
- [32] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [33] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019.

- [34] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.
- [35] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.
- [36] Suping Shi. Comparison of player tracking-by-detection algorithms in football videos. *Master’s Thesis in Computer Science, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology*, 2020.
- [37] Wei-Ting Lee and Hwann-Tzong Chen. Histogram-based interest point detectors. In *2009 IEEE conference on computer vision and pattern recognition*, pages 1590–1596. IEEE, 2009.
- [38] Yin Yankun, Du Xiaoping, Chu Wenbo, and Wuniri Qiqige. A color histogram based large motion trend fusion algorithm for vehicle tracking. *IEEE Access*, 9:83394–83401, 2021.
- [39] Cheolmin Choi, Jungho Ahn, Seungwon Lee, and Hyeran Byun. Disparity weighted histogram-based object tracking for mobile robot systems. In *Advances in Artificial Reality and Tele-Existence: 16th International Conference on Artificial Reality and Telexistence, ICAT 2006, Hangzhou, China, November 29-December 1, 2006. Proceedings*, pages 584–593. Springer, 2006.
- [40] Tianzhu Zhang, Bernard Ghanem, and Narendra Ahuja. Robust multi-object tracking via cross-domain contextual information for sports video analysis. In *2012 IEEE international conference on acoustics, speech and signal processing (icassp)*, pages 985–988. IEEE, 2012.
- [41] Wei-Lwun Lu, Jo-Anne Ting, Kevin P Murphy, and James J Little. Identifying players in broadcast sports videos using conditional random fields. In *CVPR 2011*, pages 3249–3256. IEEE, 2011.
- [42] Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991.
- [43] Xiaoyu Li, Bo Zhang, Pedro V Sander, and Jing Liao. Blind geometric distortion correction on images through deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4855–4864, 2019.
- [44] Yu-Zhen Zhang, Qing Dong, Jian-Yu Wang, and Yue-Wei Dai. Court view shots detection based on hough transform and svm. In *2011 International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping*, pages 1–4. IEEE, 2011.
- [45] Manel Farhat, Ali Khalfallah, and Med Salim Bouhlef. A new model based approach for tennis court tracking in real time. *International Journal of Signal and Imaging Systems Engineering*, 11(1):9–19, 2018.

- [46] Faisal Bashir and Fatih Porikli. Performance evaluation of object detection and tracking systems. In *Proceedings 9th IEEE International Workshop on PETS*, pages 7–14, 2006.
- [47] Yu-Chuan Huang, I-No Liao, Ching-Hsuan Chen, Tsì-Uí Ík, and Wen-Chih Peng. Tracknet: A deep learning network for tracking high-speed and tiny objects in sports applications. In *2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–8. IEEE, 2019.
- [48] João Manuel Ribeiro Silva Tavares and Armando Jorge Monteiro Neves Padilha. A new approach for merging edge line segments. *Proceedings RecPad'95, Aveiro*, 1995.
- [49] David Wennerblom and Andrey Arronet. Padel court detection system. *Master's Thesis in Computer Engineering, School of Engineering, Jönköping University*, 6 2023.