THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
Ecole Centrale Lyon et INSA Lyon

École Doctorale 512
Informatique et Mathématique de Lyon
(INFOMATHS)

Spécialité
Informatique

Présentée par

Nicolas Jacquelin

Pour obtenir le grade de
DOCTEUR de L'UNIVERSITÉ DE LYON

Sujet de la thèse :

Automatic Analyses of Swimming Race Videos

Analyses Automatiques de Vidéos de Course de Natation

Soutenue publiquement le XXX, devant le jury composé de :

ABSTRACT

In top-level sport, where all participants have exceptional physical and technical skills, as well as deep theoretical knowledge of their field, the gap between the best results is minimal. The winner is determined by small details that may seem insignificant to the uneducated eye, but are actually fundamental to gaining ground on others. In swimming in particular, many finals of important competitions end up with a difference of less than a tenth of a second between the leaders. The details bringing victory can be very varied because they concern the individual physique of the swimmers, their mental and physical preparation, their understanding of the swimming style of their competitors, and many other things. Understanding them is crucial to winning: this is the role of the sports coaches. They will study with finesse what can allow their swimmer to be the most efficient.

The first step in the analysis of training and races is information extraction. In this thesis, we are particularly interested in swimming competitions. Our goal is to generate an automatic race report. This would free up an invaluable amount of time for coaches, and would also allow for extensive analysis of competitions. Such technology would also improve the detection of potential talent through the systematic analysis of all amateur competitions.

We will focus here on video analysis, as sensors and other intrusive acquisition systems cannot be used during championships. This imposes important constraints related to the recording conditions of the videos: our methods must be robust and general. Computer vision methods will be explored to get the best out of the videos. We will also explore image analysis in a less data-dependent way than usual. Indeed, this field has progressed enormously over the last decade thanks to the development of deep learning, but it depends a lot on the quality and quantity of data. Our general problem will therefore concern the extraction of information from swimming race videos using small amounts of data. This task will be divided into three parts, each one studying a specific type of information. All results, models, and resulting databases have been published online, accessible to all.

We will start by focusing on the detection of swimmers in images. This task is the most obvious to start with, because to study a swimmer on a race, we must be able to locate him. This chapter will therefore introduce a detection method specifically adapted to swimmers, as well as a dataset related to the task.

Detecting swimmers on an image is a first step, but it does not give positional information in the pool. For that, we need to register the image, that is to map each point of the image to a zone of the pool. A particularly fast and very efficient method will be explained to answer this task. Another dataset will be presented.

The third part of this thesis will concern the measurement of swimming cycles. The repetition of the movement being omnipresent during a race, its study is one of the most useful to perceive the quality of swimming. It is an excellent basis for measuring a swimmer's fatigue, efficiency, or technique. A general method to count cycles on a video will be presented. Specifically for swimming, we will also describe a way to locate the ends of cycles, in order to measure their individual duration with precision.

All this will lead to the fusion of these methods into a single tool capable of analyzing swimming races with little or no human intervention. The different models will be arranged in such a way as to get the most out of each and to reduce their individual errors. The qualities and limits of this tool will be presented, as well as the way to measure its accuracy. Our goal is to make it efficient enough for the FFN to use it during its competitions and training in order to improve performances.

RÉSUMÉ

En sport de niveau international où tous les participants ont des compétences physiques et techniques exceptionnelle, ainsi que de profondes connaissances théorique de leur domaine, l'écart entre les meilleurs résultats est minime. Le vainqueur est déterminé par des petits détails qui peuvent sembler infimes à l'œil du profane, mais qui sont en réalité fondamentaux pour gagner du terrain sur les autres. En natation en particulier, plusieurs finales de compétitions importantes se soldent par un écart inférieur au dixième de seconde entre les premiers. Les détails apportant la victoire peuvent être très variées car elles concernent le physique individuel des nageurs, leur préparation mentale et physique, leur compréhension du style de nage de leurs concurrents, et bien d'autres choses. Comprendre finement ces différences est donc crucial pour l'emporter : c'est le rôle des entraîneurs sportifs. Ils vont étudier avec finesse ce qui peut permettre à leur nageur d'être le plus efficace.

La première étape de l'analyse des entraînements et des courses est l'extraction d'information. Dans cette thèse, nous nous intéressons particulièrement aux compétitions de natation. Notre objectif est de générer un compte-rendu de course automatique. Cela libérerait une quantité de temps inestimable pour les entraîneurs, et permettrait également l'analyse exhaustive des compétitions. Une telle technologie permettrait également d'améliorer la détection de potentiels talents via l'analyse systématique de toutes les compétitions amateures.

Nous nous concentrerons ici sur l'analyse vidéo, les capteurs et autres systèmes d'acquisition intrusifs ne pouvant être utilisés lors de championnats. Cela impose des contraintes importantes liées aux conditions d'enregistrement des vidéos : nos méthodes devront faire preuve de robustesse et de généralisation. Des méthodes de vision par ordinateur seront explorées afin de tirer le meilleur des vidéos. Nous explorerons également l'analyse d'image de manière moins dépendante des données qu'habituellement. En effet, ce domaine a énormément progressé au cours de la dernière décennie grâce au développement de l'apprentissage profond, mais il dépend beaucoup de la qualité et quantité des données. Notre problématique générale concernera donc l'extraction d'informations issues de vidéos de course de natation en utilisant de faibles quantités de données. Cette tâche sera divisée en trois parties, chacune étudiant un type d'information précis. Tous les résultats, modèles, et bases de données qui en découlent ont été publiés en ligne, accessibles à tous.

Nous commencerons par nous intéresser à la détection des nageurs dans les images. Cette tâche est la plus évidente à comprendre, car pour étudier un nageur sur une course, il faut être capable de le localiser. Ce chapitre introduira donc une

méthode de détection spécifiquement adaptée aux nageurs, ainsi qu'une base de données liée à la tâche.

Détecter les nageurs sur une image est une première étape, mais cela ne nous donne pas d'information de position dans le bassin. Pour cela, il faut recaler l'image, c'est à dire savoir à quelle zone de la piscine correspond chaque point de l'image. Une méthode particulièrement rapide et très efficace sera expliquée pour répondre à cette tâche. Une autre base de données sera présentée.

La troisième partie de cette thèse concernera la mesure de cycles de nage. La répétition du mouvement étant omniprésente pendant une course, son étude est l'une des plus utiles pour percevoir la qualité de nage. Il s'agit d'une excellente base pour mesurer la fatigue d'un nageur, son efficacité, ou sa technique. Une méthode générale pour compter les cycles sur une vidéo sera donc présentée. Spécifiquement pour la natation, nous décrirons également une manière de localiser les fins de cycles, dans le but de mesurer leur durée individuelle avec précision.

Tout cela aboutira à la fusion de ces méthodes en un seul outil capable d'analyser les courses de natation avec pas ou peu d'intervention humaine. Les différents modèles le constituant y seront agencés de manière à tirer le maximum de chacun d'entre eux et de réduire leurs erreurs individuelles. Les qualités et limites de cet outil seront présentées, ainsi que la manière de mesurer sa précision. Notre but est de le rendre assez performant pour que la FFN s'en serve durant ses compétitions et entraînement dans un but d'amélioration des performances.

REMERCIEMENTS

Plusieurs personnes ont permis la bonne tenue de cette thèse, que ce soit par leurs encouragements ou leurs conseils, qu'ils soient d'ordre techniques, organisationnel, ou personnels.

Tout d'abord, je salue mes encadrants qui ont fait preuve de diverses qualité. Stefan Duffner, mon directeur, a su me guider sur les pistes techniques de ma thèse. Son expérience en vision par ordinateur m'a énormément apporté. Merci également à Romain Vuillemot, co-encadrant de la thèse, qui a mis en place et organisé le projet Neptune et m'a mis en contact avec la FFN. Ses conseils d'organisation et de planification ont permis une thèse riche en rencontres et discussions.

Je remercie aussi les membres de la FFN (Fédération Française de Natation) et particulièrement le pôle performance. Leur rencontre, ainsi que les opportunités de filmer des compétitions de niveau national, ont grandement amélioré la thèse. J'ai beaucoup apprécié ces évènements.

Je remercie également mes deux équipes au sein du LIRIS. Tout d'abord, côté École Centrale, l'équipe Imagine, grâce à laquelle je bois désormais du café **??**. Merci aussi à l'équipe SICAL, qui m'a permis d'écrire ma thèse en 6 mois à la place de 5 tout en devenant meilleur aux échecs. Les membres de ces deux équipes auront su à leur façon m'apporter de la motivation et des connaissances techniques qui ont permis un bon déroulement de la thèse.

Enfin, un remerciement général à ma famille et mes amis qui m'ont apporté leur soutien et ont toléré mon retards à des évènements et ma paresse en leur présence. Promis, je bossais.

Je remercie pour finir l'invité surprise de la thèse, le covid 19. Il a su significativement réduire mon impacte carbonne durant cette thèse en m'évitant de prendre l'avion pour aller en conférence.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

AP	Average Precision
AR	Average Recall
CNN	Convolutional Neural Network
CAM	Class Activation Mapping
CV	Computer Vision
DL	Deep Learning
DNN	Deep Neural Networks
DoG	Difference of Gaussians
FFN	Fédération Française de Natation (French Swimming Federation)
FPS	Frames per Second
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
HOG	Histograms of Gradient
IOU	Intersection Over Union
JTC	Joint Transform Correlation
KL div	Kullback–Leibler Divergence
mAP	mean Average Precision
mAR	mean Average Recall
MIL	Multiple Instance Learning
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NMS	Non-Maximum Suppression
NN	Neural Networks
OBOA	Off-By-One Accuracy
PCA	Principal Component Analysis
POV	Point of View
ReLU	Rectified Liner Unit
ROI	Regions Of Interest
RPN	Region Proposal Network

SGD	Stochastic Gradient Descent
SIFT	Scale Invariant Feature Transform
SOTA	State of the Art
SVM	Support Vector Machines
tanh	hyperbolic tangent
VAE	Variational Autoencoder
YOLO	You Only Look Once

CHAPTER 1

INTRODUCTION

Contents

At the 2022 swimming France championships, after Maxime Grousset and Florent Manaudou finished in respectively first and second place, they were asked by a sports media to talk about their race. Maxime Grousset explained how he started strongly with great first 15m, then how his finish was under-optimal. Such analysis performed intuitively by such a great swimmer (silver medal in 100m freestyle at the 2022 World championships) is extremely valuable. It allows him and his coach to know what to practice and what is mastered.

However, from inside the pool, no swimmer can have a global view of what happened to each participating swimmer. Their relative position, how what happens in each lane influences the ones it touched... Further, Grousset explained his qualitative feelings of the race, but he could not have provided quantitative data showing how ahead he was initially and how he slowed down a the end. He also could not always know exactly what lead him to slow down around the end, such as a bad swimming movement several seconds later. Further, qualitative information, such as what Grousset explained after the race, can be extracted from quantitative data. "Slow start", "good finish", or "irregular cycles", can be easily said just by analyzing the numbers. Such appreciations are often more meaningful and swimmer-friendly than a whole set of metrics displayed on a spreadsheet. One could either compare themself to absolute criteria (world records, other swimmer's performance), or personal ones (previous results during the championship, comparison to last years). Such output must also depend on the race nature, as one expects more calm regularity from a 1500m qualification race and more exploding dynamics from a 50m finals.

These quantitative and qualitative analyses can only be performed by both studying the entire race after it ended and also each frame of the video, closely. Coaches can do it manually using specific tools, but they tend to focus on a few metrics they personally prefer. Further, not each swimmer has either a coach

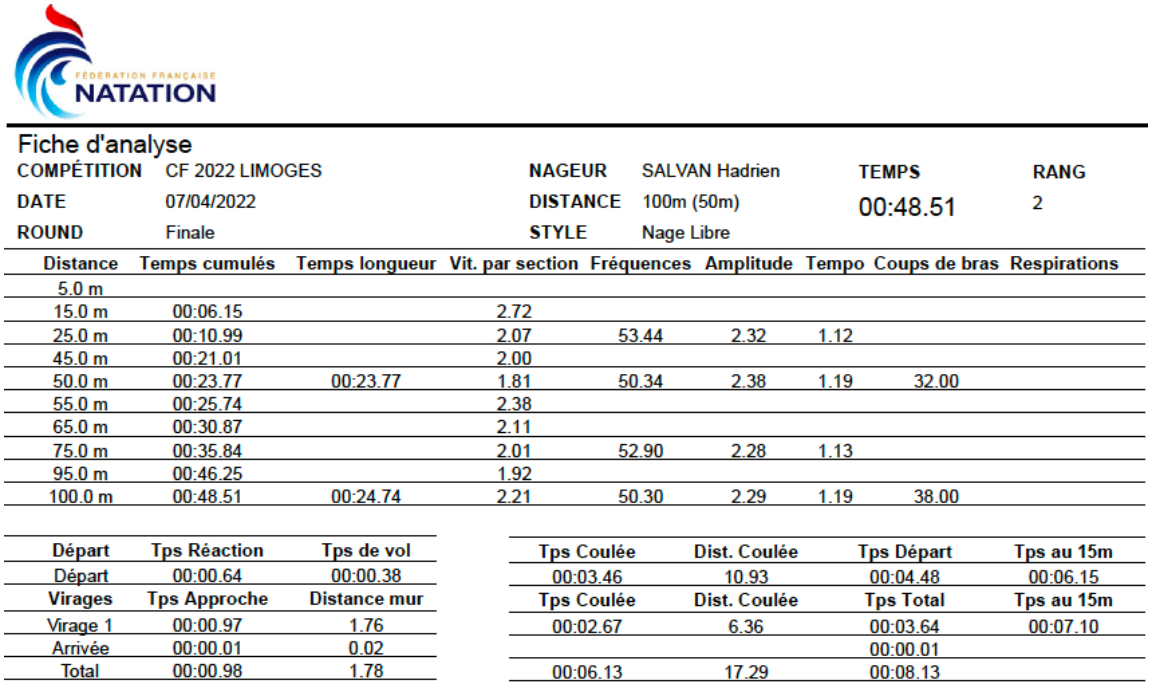
Figure 1.1 – Pool framing in different conditions. Point of View (POV) 1 gives a better view of the pool than POV 2 due to the camera placement and lighting conditions.

to do it for them, or the time and tools at disposition. Therefore, being able to automatically generate such a race summary with quantitative metrics and pixel-wise precision can be of broad interest to the swimming community. If any swimmer could have access to a summary sheet showing how they performed during a race, that could help them to progress rapidly. The experience Grousset needs to analyse his race just after he jumped out of the water could be available to anyone. The tool could also be profitable to head hunters looking for potential young talents. They cannot watch every race of every local competition, but if they could have access to their data, the task could get easier. Swimmers could be detected faster and thus get better results.

The Neptune project (Natation et Paranatation: Tous Unis pour Nos Elites) began just 3 months after this doctorate. It aimed at bringing academics and swimming professionals together to prepare for the 2024 Paris Olympics. This project is multidisciplinary and concerns computer science, but also mechanics, materials science, biomechanics, or fluids mechanics. We met members of the Fédération Française de Natation (French Swimming Federation) (FFN) on many occasions. They explained what lacked from their current analyses and the researchers focused on these topics. As such, this thesis presents the research made with their collaboration, guided by their directions and objectives. Its purpose was to create a tool to automatically analyze races with little human assistance, using Computer Vision (CV).

1.1 Using Videos

To analyze races, coaches rely on videos. Compared to other data streams, they are very convenient. A camera is transportable and can be installed next to a pool with no major requirements other than a dry area. It does not need swimmers to equip with intrusive sensors or to proceed to extensive prior calibration phases. Cameras are easy to manipulate even by non-experts due to their presence in our



Fiche d'analyse

COMPÉTITION	CF 2022 LIMOGES	NAGEUR	SALVAN Hadrien	TEMPS	RANG
DATE	07/04/2022	DISTANCE	100m (50m)	00:48.51	2
ROUND	Finale	STYLE	Nage Libre		

Distance	Temps cumulés	Temps longueur	Vit. par section	Fréquences	Amplitude	Tempo	Coups de bras	Respirations
5.0 m								
15.0 m	00:06.15		2.72					
25.0 m	00:10.99		2.07	53.44	2.32	1.12		
45.0 m	00:21.01		2.00					
50.0 m	00:23.77	00:23.77	1.81	50.34	2.38	1.19	32.00	
55.0 m	00:25.74		2.38					
65.0 m	00:30.87		2.11					
75.0 m	00:35.84		2.01	52.90	2.28	1.13		
95.0 m	00:46.25		1.92					
100.0 m	00:48.51	00:24.74	2.21	50.30	2.29	1.19	38.00	

Départ	Tps Réaction	Tps de vol
Départ	00:00.64	00:00.38
Virages	**Tps Approche**	**Distance mur**
Virage 1	00:00.97	1.76
Arrivée	00:00.01	0.02
Total	00:00.98	1.78

Tps Coulée	Dist. Coulée	Tps Départ	Tps au 15m
00:03.46	10.93	00:04.48	00:06.15
Tps Coulée	**Dist. Coulée**	**Tps Total**	**Tps au 15m**
00:02.67	6.36	00:03.64	00:07.10
		00:00.01	
00:06.13	17.29	00:08.13	

Figure 1.2 – A summary sheet filled-in by the performance division.
Frequency (min^{-1}) = #strokes/time, Amplitude (m) = 50m/#strokes, Tempo (s) = 60s/Frequency.

daily life and their similarity to our eye's behavior. Specific technique parameters can be selected by experts before the competition. Further, videos from TV streams are easy to access for free on the internet.

Apart from videos, it is also possible to analyze a race using other capture methods or intrusive sensors. Body sensors, for instance, are much more constraining although they give a precise body position. Intrusive gears are not allowed during competitions, contrary to filming material. Likewise, ultrasound or infrared systems are not adapted to pools due to the environment's properties. Sensors are constrained to their specific domain, while image and sound are exhaustive. They contain abundant relevant information related to swimmers' performance as well as environment conditions. Videos are also good communication tools to explain results to coaches, swimmers, or teams. Other data can be integrated directly onto them and visualized, either using tracking methods or simply by putting abstract data in the spatial or temporal context. Finally, videos are direct sources of data. Posterior blog posts or live social media feeds can describe a sports meeting, but the information is filtered by the author, who can focus on some points more than others. Videos are also constrained by the field of view they frame, but it is usually the most representative media of what happens in a competition. In the end, most metrics that are not videos are extracted using videos. All these reasons made us focus on videos instead of others.

1.2 Swimming Races

To analyse a race, one must first set up a capture system. In practice, coaches rely on videos and metadata published by the race organizers, such as swimmers' times. A camera is placed on the poolside at different places depending on the presence of a public in the stands, the availability of a filming cell, etc. In function of the position-induced POV, the video quality can vary significantly, as shown in Figure 1.1. A camera that is centered, meters higher than the pool, and at a good distance from it, gives the best point of view. Further, indoor pools have less lighting problems, which may cause camera saturation during filming or visibility problems in general during the analysis. Other problems can arise, such as spectators standing in front of the camera. These capture conditions must be best fitted for the later analyse. During the thesis, the question of panning cameras versus static ones was asked several times. Static cameras are easy to calibrate but are not zoomed into swimmers. The analysis is therefore more robust but limited by the numerical zoom. On the other hand, a video following the swimmers (either all of them or just a small group) is closer to the action and can thus a better understanding of the race, but this will be limited by the calibration quality, significantly harder than for fix cameras (see Chapter 4). As the situation evolved during the project, the answer changed. This results in different kinds of captured videos that cannot all be studied in the same way due to their very nature (either statics or following a group of swimmers). This thesis proposes solutions for both.

Capture is not only limited to camera placement. Framerate and resolution are also important, as well as the zoom level. They all allow different kinds of analysis. With a high framerate and an important zoom or a big resolution, it is possible to closely study the movements of each swimmer. However, it is more cumbersome to handle such massive data compared to 25 Frames per Second (FPS) 720p videos, which may be more suited for global swimmers' position and ranking throughout the race.

The FFN's performance division used the videos thus obtained to fill in summary sheets, as shown in Figure 1.2. An expert informs the position through time and the number of swimming strokes during one pool length (the individual stroke times are also measured but not displayed here) to compute the other metrics. Despite all being inferred from the same two values, they describe a race in different ways to fit the uses of different coaches. In general, it is better to seek fewer strokes of better quality, as they maintain a better speed for less energy cost. Further studies in the Neptune project quantitatively measure this phenomenon. Breathing is also an important aspect of swimming that is informed in these sheets. Although in 50m races, top-level swimmers do not breathe to maintain their speed, with 100m already this is not the case anymore. This second type of race contains a deeper strategic and resource management aspect. Indeed, with

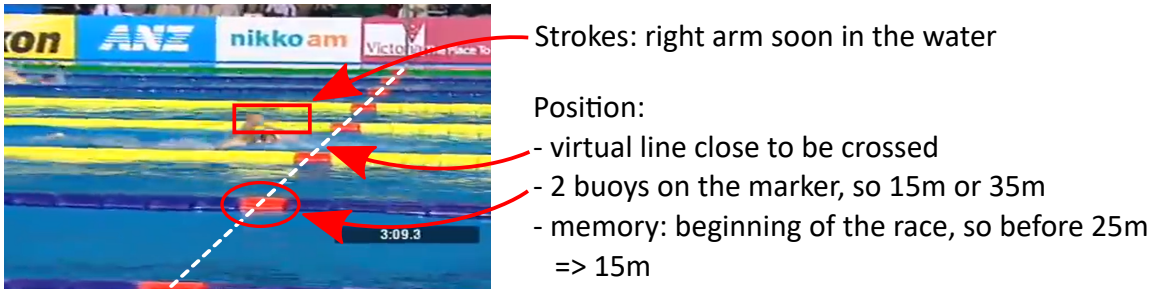


Figure 1.3 – How humans understand a pool and use their prior knowledge of the situation to infer data out of the video.

too few breaths during the first 50m, one can be the first to enter the second half of the race, but they will likely collapse before the finish.

Although these data contain a lot of valuable information, we think it is possible to use automatic CV methods to enhance them. It is possible to create continuous data describing the race more precisely instead of discrete data points. This sheet is also created for only a few races (especially finals and semi-finals) or swimmers with access to the performance division. With an automatic method, one could fill them in for each participant, and even for foreign competitions if compatible videos are available. Finally, the bottleneck of these analyses is the human time they require. Certain persons in the performance division spend their entire time filling them in, yet computers with the right algorithm can handle them if enough research is done on the topic. Further, knowing the position of a swimmer in a pool is different from knowing their position in the image. The latter enables automatic zooms on the swimmer to better study their movements.

With CV methods, it is possible to tackle many limitations highlighted here. In this thesis, we will focus on swimmers' position and strokes. We argue these two metrics can inform the most on a race quality. As in the sheet, different metrics useful for coaches can be inferred by them. The works developed to that end could also extend to other topics such as breathing detection, though no work has been done directly for this during this doctorate.

1.3 Tasks Breakdown

For the FFN, the current method to perform swimming race video analysis is using a software developed for their specific needs. This manual tool enables them to enter the different race metadata (swimmer name, total time, race ranking, etc.) and input others themselves. To enter the time of the first 50m for instance, an expert first identifies the frame where the swimmer touches the pool edge, then the frame corresponding to the start buzzer sound. These data are handled by their tool which also finds the video frame rate and computes a time. The

strokes end must be defined the same way for each analyst. For instance in crawl, it is usually when the right hand touches the water. To count them and assign them a time, the video is played (sometimes in slow-motion) and each time the stroke end event occurs, the expert presses a key. Similar actions are made to detect the breathings. The swimmer time is also marked each time they cross a length marker (5m, 15m, 25m, 35m, and 45m for Olympic pools) with the same key-press protocol. This analysis tool is completely manual and relies entirely on the expert to validate the results. The expert also needs to focus on one swimmer at a time, despite a video containing information about at least the surrounding swimmers.

To replicate the same measures as this tool with CV, one must understand the prior knowledge humans have regarding the spatial structure of a pool. One must also realize parallax deformation is easily compensated by our brain, but that an image processing system does not have such prerequisites. Finally, to count the swimming cycles of a swimmer, one must before know where the swimmer is in the pool. These information are displayed in Figure 1.3. As CV algorithms do not have such prior knowledge, they cannot study a race the same way humans do. The entire analyse pipeline is thus going to be completely different. Other tasks that are not perfectly trivial to humans must also be addressed in order to perform a complete analysis. First, capturing the entire pool may require multiple cameras, thus they must be synchronized temporally. To fill in a swimming sheet summary, it is also required to know who are the swimmers on the video. Therefore, identifying them and associating them to a given swimming lane is fundamental. Finally, to study periodicity, one must segment the swimming into different phases, because there is no periodicity during the diving, the underwater section, and the u-turn. Here is a summary of these tasks:

- **video spatial calibration**
- **periodicity estimation**
- **swimmers detection**
- cameras temporal synchronization
- swimmers identification
- swimming phases segmentation

In this thesis, we focus on the tasks previously listed in bold. Some of the others have been addressed using parallel methods not based on CV, or even not based on automatic analysis at all (see chapter 6). The others are left to future works.

1.4 Computer Vision Challenges

For our automatic model, we propose a pipeline suited to our fields of research and the current possibilities of CV. As such, we propose to start with detection. Once it is be possible to reliably detect a swimmer, only half the swimmers

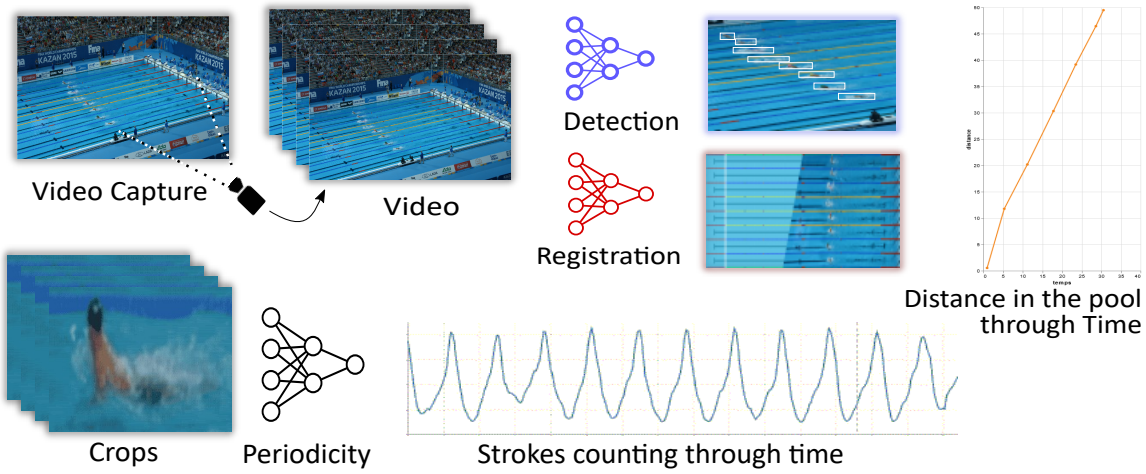


Figure 1.4 – Description of our pipeline. Each Neural Networks (NN) model represents a different challenge this thesis addresses. The direct objectives are to estimate the position through time and the periodicity of the swimmers.

positioning part is done: depending on the camera position, the framing, and possible camera movements, knowing the position of a swimmer on the image is different from knowing their position in the pool. A camera calibration step is thus necessary to resolve this. Finally, with each swimmer detected, one can extract crops of the video framed around the swimmers, and study the periodicity of these videos. This process is displayed in Figure 1.4.

These tasks are all classic CV challenges which have been tackled in other contexts. Detection is one of the main CV research topic, registration is often use in the context of autonomous driving [18], football analysis [184, 32, 106], or medical imaging [71, 151], and periodicity counting has always interested the community [126, 56]. This is not chance, as this breakdown was created to follow classic CV topics.

However, the works tackling directly swimming are rare and no pre-existing model can be used as an *ad hoc* solution. Extensive works of adaptations will be necessary to finally obtain a CV algorithm fitted for swimming. Indeed, challenges addressed by most detection algorithms usually concern daily-life videos and general problems. This enables good overall results, but causes performance drop under unusual conditions [135, 63] such as new orientation, different background, presence of specific other objects, etc. In swimming, the type of obfuscation in unique (waves, bubbles, diffraction) and the background is peculiar. Swimmers, *i.e.* the objects of interest, have large partial occlusions and considerable appearance variations due the articulated body motion, waves and other types of noise. A model specifically addressing these constraints would perform better for a given cost than a general algorithm fitted for many issues that will never occur in our context.

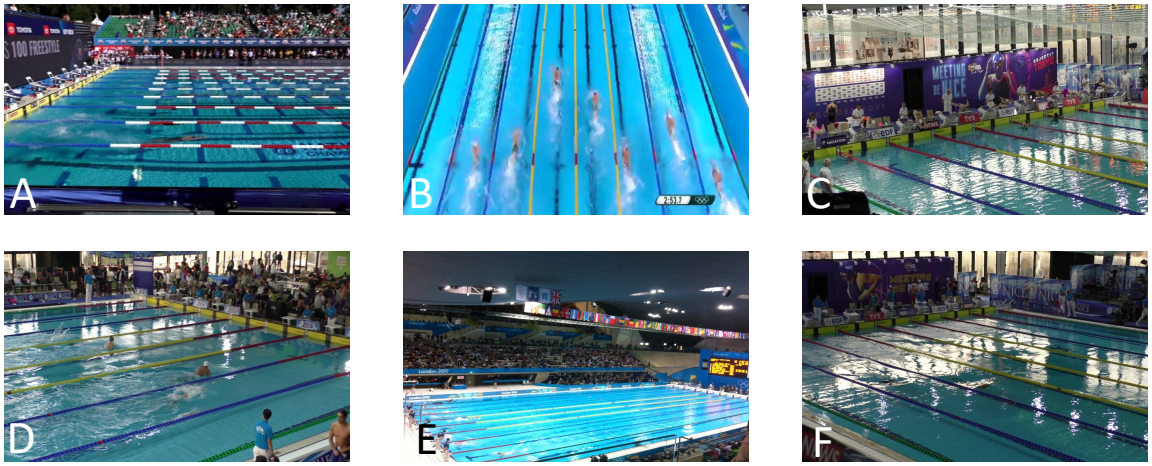


Figure 1.5 – Difficult conditions in swimming races video. Underwater swimmers are sometimes barely visible (A, F). Orientation and distance are also a challenge (B, E). Diving swimmers can be close to referees and there are still swimmers in the pool from a previous race, which creates a difficult subjects of interest choice (C). Only 2 swimmers are on image D, the others being at a different part of the pool. Different lighting conditions, like outdoor with bright sun (A), or indoor with back-lighting (C, D, F).

Further, Deep Learning (DL) algorithms rely on dedicated datasets, which must be big enough to train heavy powerful models. No public dataset of the sort exists in swimming, either for detection, registration, or strokes counting. Further, although a swimmer is a human and the human class appears in many detection datasets [129, 60], these datasets do not contain examples of humans in water. As a result, even powerful object detection models [165, 80, 220] able to detect any pedestrian, runner, or sitting person, are unable to robustly detect swimmers.

CV results also depend significantly of the video capture condition. Difficult lighting, obfuscation, motion blur, and every other perturbation can cause important performance drop. Such problems exacerbate in swimming, as examples showcase it in Figure 1.5. These conditions are extremely frequent, due to the very nature of this sport. The background is a reflecting object close to giant bay windows, the waves and bubbles can randomly hide any region of interest, the swimmers are close to the surface causing diffraction problems, etc. Further, there is no general camera placement or parameters set to avoid all these issues. They have to be handled by the algorithm.

The chapters of this manuscript each concern one sub-task of these problems. We explain the detection of swimmers in an image (Chapter 3), then the registration of an image in a field (Chapter 4). Using the detection, we further crop sub-videos around the swimmers and study these videos' periodicity (Chapter 5). Finally, a global unifying tool is described (Chapter 6), creating a synergy with the different models to increase their performance. The objectives of such a general swimming race automatic analysis tool are to:

- automatically estimate spatio-temporal data (position and swimming rate).
- reduce the time-consuming video analysis task performed by the coaches.
- enable exhaustive analysis of any swimming race.
- be non-intrusive, quick, and easy to setup for a new pool
- isolate new interesting swimming styles and strategies.
- facilitate swimmers spotting during sports events.

1.5 Contributions

Three papers have been produced during this PhD. This includes a paper on swimmer detection [103]. The proposed method handles the constraints of swimming, with data augmentation that is adapted for the task at hand and a chosen architecture perfectly suited for swimmer detection, especially as they are ever-changing shapes of varied size.

Nicolas Jacquelin, Stefan Duffner, Romain Vuillemot. *Detecting Swimmers in Unconstrained Videos with Few Training Data*. Machine Learning and Data Mining for Sports Analytics, Sep 2021, Ghand, Belgium. ⟨hal-03358375⟩.

A paper tackling the task of camera calibration has also been produced [104]. The purpose is to perform registration, that is mapping the input video to a virtual top-view where the pool (visible on the image and out of the image) forms a rectangle of known dimensions. This gives a direct correspondence of a position (in pixels) in the image with a position (in meters) in the pool. The problem is formulated as a homography matrix regression task. To find it, several points of interest are found on the image, allowing a consensus algorithm to compute the matrix.

Nicolas Jacquelin, Stefan Duffner, Romain Vuillemot. *Efficient One-Shot Sports Field Image Registration with Arbitrary Keypoint Segmentation*. IEEE International Conference on Image Processing, Oct 2022, Bordeaux, France. ⟨hal-03738153⟩.

A third paper has been written, addressing strokes counting [105]. It introduces a general periodicity estimation method for complex temporal signals. It works on regular videos with 2D images, but also on more complex series such as 4D videos (like IRMs, scanners, etc.) or more abstract signals. It returns the number of periods in a signal in an unsupervised fashion, training a model without external data.

Nicolas Jacquelin, Stefan Duffner, Romain Vuillemot. *Periodicity Counting in Videos with Unsupervised Learning of Cyclic Embeddings*. Pattern Recognition Letters, Elsevier, 2022, ⟨10.1016/j.patrec.2022.07.013⟩, ⟨hal-03738161⟩.

In addition to these papers, two datasets have been introduced along them, namely $Swimm^{400}$, published in [103], a swimmer detection dataset, and $RegiSwim^{500}$, published in [104], a swimming pool registration dataset.

Finally, a tool has been delivered to the FFN, enabling them to automatically analyse swimming race videos. It merges the ideas from these papers in a way that optimizes processing time and that gives robustness to the resulting model.

CHAPTER 2

RELATED WORKS

Contents

Solving our challenge of automatic swimming video analysis requires a deep understanding of what is image analysis or, as it is more usually referred to as, Computer Vision (CV). This domain uses different methods coming from signal processing or data analysis to understand the content of an image. A CV algorithm inputs an image and outputs information relative to it, which depends on the algorithm's nature.

The domains of application of CV are extremely varied and tend to develop with technologies. The automation of vision-based tasks is tackled with CV (pedestrian counting, action recognition) and frequently online paired with camera (video surveillance, plants monitoring, etc.). Monitoring tasks are indeed often addressed using CV because it is reliable and cheap. Outside of this domain, experts can be assisted by CV models for complex tasks to enhance their decision-making. This is especially frequent in medical imaging, where small cancer tissues are easy to miss by a human eye watching the whole scan. Regarding sports analysis, as for this thesis, it can both be seen as monitoring and experts assistance. Indeed, analysing swimmers during a race is a monitoring task: once the video is created, one can wait for the analysis to be over to get a race summary. However, CV can

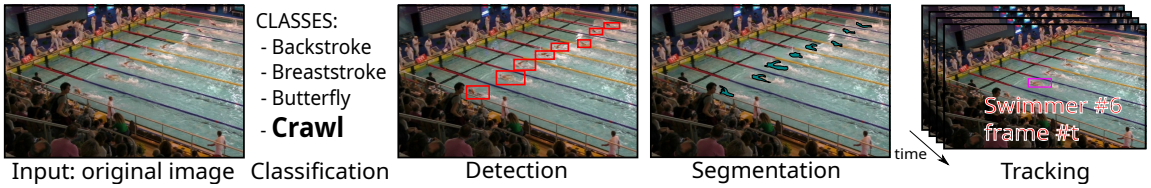


Figure 2.1 – An illustration of the different CV tasks. For classification, there is a list of possible classes present in the image. The detected classes are in bold. For the detection task, the problem is formulated as the placement and shaping of bounding boxes framing the objects of interest (here swimmers). Segmentation models output a probability mask where each pixel maps a pixel from the source image. High probabilities (*i.e.* close to 1) define the positions where an instance of interest (here a swimmer) is present, while the background is at zero. Tracking starts with the detection of an instance (either provided by a detection model or directly by a human) and uses the results of previous frames to detect the same instance on new frames.

also help coaches identify anomalies during a race or training. For instance, if the stroke rate suddenly changes for a small period, a coach might analyse the corresponding video timecode and find out an improvement area for the swimmer. As such, CV applications can be extremely broad, even with one tool, depending on the use context.

Some tasks being easier than others, one must segment high-level abstract tasks into an optimal set of correctly defined sub-tasks. Here is an overview of some challenges of the domain, each represented in Figure 2.1.

- classification: the content of the image must be identified among a list of potential classes. It can either concern objects in the image or the general definition of the scene. This is usually considered the basics of CV.
- object detection: the objects in the image must be identified and located. The positional aspect of this task is general (either a barycenter or a bounding box). This time, multiple objects can be present in the same image.
- segmentation: this task estimates the probability of each pixel to belong to a given set of classes. It can be understood as pixel-wise classification.
- object tracking: in a video, any object is selected and must be detected on each subsequent frame, even if it moves are changes. This task uses temporal information from the video to extract information on each frame.

They have different levels of complexity which involve different speeds of execution. The time constraint is present in a great amount of CV applications. Sometimes, as only a few images are to be analyzed this is not a problem, but in real-time video analysis, for example, the inference speed is critical. In such conditions, trade-offs such as speed vs precision must be made. Depending on the use-cases, one is more important than the other. Classic older algorithms described in the next section are faster as they are simpler than the newest ones, even with the slower hardware available at the time. Through time, both hardware

and software evolved in parallel, with heavier new models running on faster new machines. As a consequence, new Deep Neural Networks (DNN) models, orders of magnitude more computationally expensive than older algorithms, can nowadays run in real-time in the correct conditions.

Further than speed of execution, classic CV methods do not rely on as much data as the more classic ones. This is usually seen as the most important difference and the reason why recent methods work better. This data dependency, however, also has its drawbacks that older approaches did not have. This will be further developed in Section 2.1.2.2.

2.1 Previously in Computer Vision

CV techniques have evolved a lot throughout the years. As the new is based on the old, understanding methods predating Deep Learning (DL) and data-dependency is required to understand the newest models of computer vision.

2.1.1 Classic Algorithms

In this section, we call *classic algorithms* any CV algorithm where the user chooses the different parameters and where the task definition is extremely specific (*e.g.*: line detection). By contrast, the majority of the modelisation process of recent methods come from the data. Further, their output yield more complex tasks than, for instance, line detection, although the indirectly perform them. In general, these "classic" methods appeared before Machine Learning (ML), mostly between the 70's and the early 2000's. In this section, we describe in detail a few of them which were milestones when they were introduced. Presenting them all is impossible, so we will only focus on the ones that were used or considered in our work, for our application.

2.1.1.1 Convolution Filters

The first family of these algorithms, coming from the signal processing domain, is the 2D convolution filter. It relies on one important property of images, which is their spatial coherence: the pixels distribution in local regions contains more information than individual pixels. Therefore, processing an image as a group of spatially organized values instead of simply a group of disjointed pixels result in richer and more meaningful information. Further, as objects can be placed anywhere in the image, translation-invariant operations are often required for better analysis.

Convolution filters implement both the idea of spatial coherence and translation invariance. A sliding window (called a filter) of size $N \times N$ convolves the whole

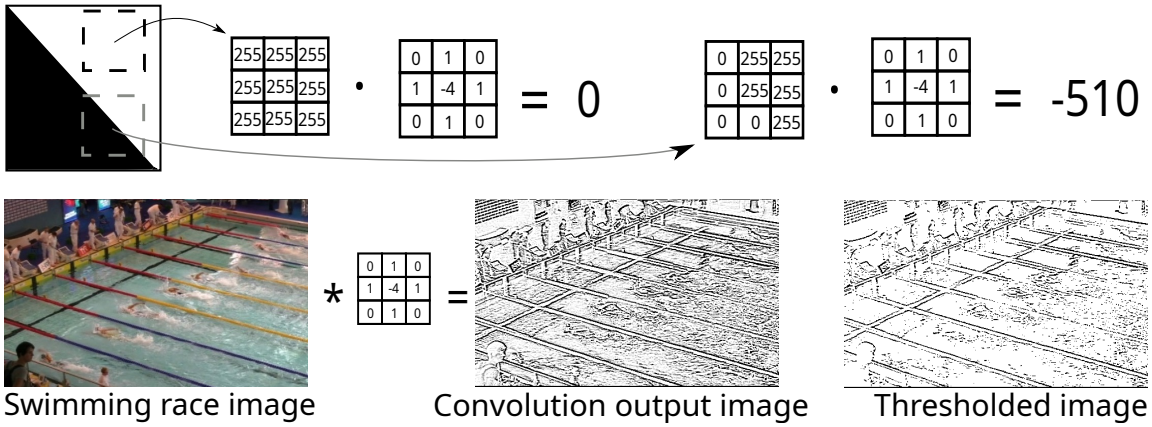


Figure 2.2 – Illustration of a 2D convolution edge-detection filters. The "*" symbol represents the convolution and the dot represents the pixel-wise matrix multiplication (*i.e.* the local behaviour of convolution) and the "•" represents element-wise multiplication. The 3×3 Laplacian filter is displayed with the result of its convolution on the image. At the top, a toy example displays the filter's behaviour without texture and with an edge texture. At the bottom, an application of the filter on a swimming race image. The line buoys are mostly well detected, as they are made of simple features with little texture. The swimmers and waves, however, are more complex and the filter cannot isolate them.

image, which results in a new image of the same dimensions. The content of the filter is key: it defines the nature of the output. A wildly used filter is an edge detector named the Laplacian is illustrated in Figure 2.2, top. In the result, extreme pixel values (*i.e.* far from 0) represent positions with sharp edges. Such a filter is intuitive and natural to interpret and understand. Using the same idea, one could create a detector of other specific shapes like that. Increasing the filter sizes allows the search for more complex patterns covering a larger region.

With 2D filters, it is possible to look for any pattern, but if the pattern is too complex, it becomes scale-dependant. To alleviate that, one can create the same filter at different scales and window sizes to have better chances to find a result. However, increasing linearly the kernel size of a filter creates a quadratic computation increase, due to the 2D nature of filters (a 3×3 filter has 9 elements, but a 5×5 filter has 25). Therefore, being scale-exhaustive is extremely slow, due to the computation time too big filters require. Filters are also orientation-dependant. It is possible to rotate their weights to look for the same pattern with some rotation, but being exhaustive requires a considerable amount of filters. It is rarely possible to be exhaustive with these filters, thus one usually uses only a few well-crafted filters. The Laplacian detects lines of any orientation. Edges are not scale-dependent, so small size kernels generally give acceptable results. Further, its output value indicates the line orientation.

After a convolution, a threshold is applied to the result to binarize the result: either a pixel represents a pattern (an edge for instance) or it does not. Such

threshold can be tricky to determine, yet it is extremely important. It depends heavily on the specific image being analysed and it is frequent to have a different threshold for different images. Further, these filters are highly noise-sensitive and patterns can be detected for no good reasons sometimes (a sharp shadow on a wall, a buggy pixel area in the camera matrix...). Some regions of high textures cannot properly be analysed by such a filter, such as the waves as shown in Figure 2.2. Indeed, this method only processes the local pixel regions, but it does not further interpret their meaning. For these reasons, 2D filters lack generalisation power. One must adapt the filter's size, orientation, and threshold depending on the context (*i.e.* the scene) that is analysed.

Despite all these problems, convolutional filters are still used nowadays for simple applications, especially for edge detection or blur (with a Gaussian filter). Edge detection has been improved with automatic cleaning algorithms, such as the Canny Filter [23]. Despite being generally more robust, it still has similar problems as the others, with unintelligible results on highly textured areas and a need for thresholds.

2.1.1.2 Hough Transform

Although convolution filters can identify local patterns, the result is still difficult to grasp for a computer. Even if some pixels are identified as edges, corners, or similar marks, one cannot identify important wider shapes. An idea that emerged in the early days of computer science (around 1960) was a way to convert certain aspects of an image into equations, much easier to manipulate than pixels.

The Hough Transform [54] was originally a method to detect straight lines on a "skeletonized" image, *i.e.* an image of edges, usually obtained with a Laplacian filter and a threshold. Afterward, this method can return an equation for any line in the image, even highly noised or partly obfuscated ones. An illustration of this method is given in Figure 2.3.

The Hough transform converts pixels in the position (x, y) space into curves in the (θ, ρ) Hough parametric space. For this operation, each pixel in the original image is converted into a sinusoid curve, following the method explained in Figure 2.3, top. criptors creation. The local grad on the same line (*i.e.* with the same (θ, ρ) pair), the corresponding position in the Hough space will be highlighted N times. As the result, one can threshold the Hough space by N to only keep the lines with N points or more. An application of the Hough transform to swimming is given in Figure 2.3, bottom.

The Hough transform is easily parallelizable, but it is not necessary as it is generally very fast. It is, however, difficult to set up: one must first extract the edges on the image and threshold the result, with all the problems and thresholds implied. Accidental lines can appear (red line in Figure 2.3), especially in the highly textured areas: if an edge detector outputs noise, many pixels can, by

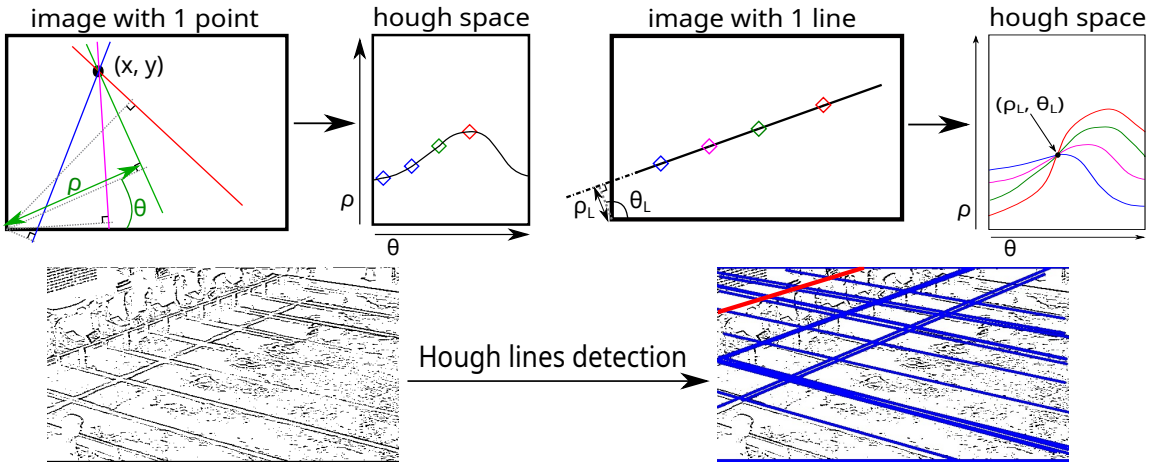


Figure 2.3 – Hough transform illustration. Top: toy examples with 1 point (left) and 1 line (right) with their respective Hough transform result. Point example: any line crossing the (x, y) position is represented in the Hough space using the angle (θ) and radius (ρ) as in the example. A point results in a sinusoid curve. Line example: the curves from each point of the line intersect in a point corresponding to the line parameters, which are not directly obtainable from the image. Bottom: Hough line detection applied to a pool (after edge detection and thresholding) to detect its buoy lines. The red line does not represent a line in the image and appears solely because of noise. Best seen in colour.

chance, be aligned. Indeed, the Hough transform does not differentiate a correct line from points across the whole image which happen to be aligned. Further, one must choose a threshold corresponding to the minimal length of a line to be considered (*i.e.* the threshold in the Hough space). This threshold largely depends on the targeted content and its scale on the image. Finally, it is common to have more than one highlighted point per line in the Hough space, each close to one another. This comes from the fact that sinusoids in the Hough space have a thickness, therefore their intersections do too. As a result, multiple close (θ, ρ) pairs may exceed the detection threshold. This results in multiple lines with very similar equation parameters (many examples of this in Figure 2.3, bottom).

Hough transform has similar problems as convolution filters: a lack of overall generalisation ability due to many context-dependent thresholds. In the past, when one had to detect lines, engineers were very careful about their image capture conditions: they avoided unwanted shadows creating lines, put everything at a calibrated distance to avoid scale problems, and were extra careful about orientation. Overall, CV was much more limited than today. As soon as an application was outside of classic calibrated tasks, it was impossible to manage all the exceptions. This problem still exists today and might be even more complex to analyse due to the complexity of DNN, but more recent methods significantly gained in robustness. This is especially the case with videos, where anything can

get closer, change its orientation, or cast shadows. For this, new methods were necessary.

2.1.1.3 Feature Matching

The two previous methods extract low-level information on the image. Such features inform on spatial properties of the objects present in the image, but they cannot provide more complex results, such as object identification. Further, these techniques have 3 main problems :

- scale-sensitivity: the same object zoomed-in can have different representations
- light-sensitivity: depending on the lighting conditions, these methods can behave in very different manners
- orientation-sensitivity: the object's orientation is crucial to any local pattern description

In the early 2000's, a few methods were developed to overcome such problems and enable deeper image understanding. Their core idea was to (i) extract points of interest in images and (ii) give meaning to these points using a semantic vector. If two vectors were similar, it meant they both represent similar patterns. If one could match the vectors of enough points like this from different images, it means the images represent a similar object. A set of vectors describing an image thus represents high-level semantic information. In general, transforming an image into a vectorial representation with semantic information is called an **embedding**. To perform embeddings, several methods were created [**surf**, **orb**, **brief**, 134], each with different properties. The most used is called Scale Invariant Feature Transform (SIFT). It performs interest keypoints detection and feature extraction.

The first step is to detect interest points, also called landmarks. They are special areas in the image containing valuable information, which have chances to be unique and discriminant compared to other areas. In practice, the highly textured regions. SIFT applies a Gaussian blur of different sizes to the image, and computes the difference between the results: this is the Difference of Gaussians (DoG). A landmark is a DoG extreme value. The image is downscaled at multiple resolutions and the process is repeated for each, giving scale-robustness to the landmarks detection. Pixel neighbourhoods around the landmarks are isolated to study the region gradient orientation, as in Figure 2.4, left and center.

After the landmark detection, SIFT computes their embedding vector, which can be considered a canonic representation of the area. This is summarized in Figure 2.4. SIFT extracts a region of 16×16 pixels around the point coordinates, rotated so that the keypoint gradient orientation always faces up (to be orientation invariant). It isolates 16 4×4 grids in it and creates a histogram of gradient direction for each of them. The orientations are discretized to only 8 possible angles to normalize the possible outcomes. SIFT also ignores the magnitude of the gradient, as they

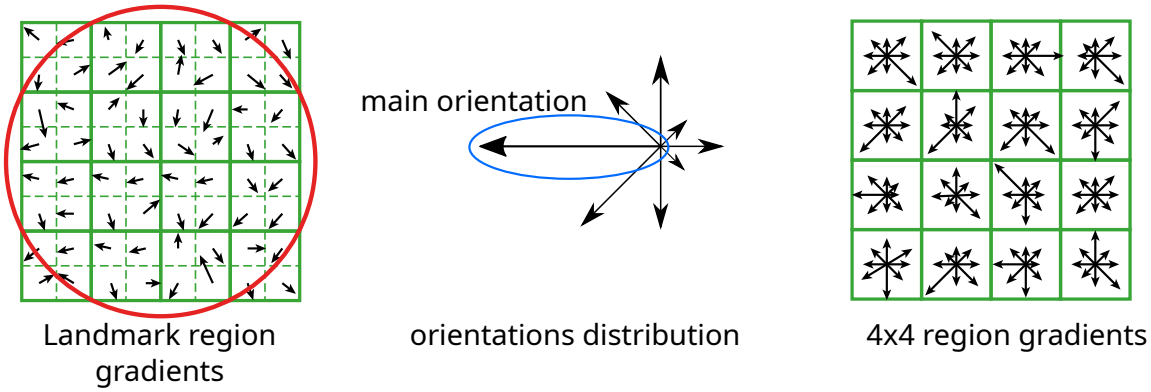


Figure 2.4 – SIFT descriptors creation. The local gradient is computed on a 16×16 region around the landmark's position. Their orientation distribution (among 8 possible angles) is computed and the dominant one is retained. Then the process is repeated for smaller 4×4 regions inside the area. These local orientation distributions are rotated accordingly to the main orientation as angle normalization. This results in $4 \times 4 \times 8 = 128$ values, *i.e.* the SIFT embedding vector.

are sensitive to lighting effects. As a result, SIFT obtains 16 histograms of 8 values. They are concatenated to form a 128-dimension vector describing the landmark's local area. Although it is not completely context-invariant, it has the main properties missing from the previous methods, as it is robust to changes in:

- rotation: the main orientation always faces up
- size: the image is scaled to several resolutions during landmark detection
- light: the orientation magnitude is ignored during the embedding vector creation

SIFT outputs detailed local pattern descriptions, but it is still not enough to describe an entire object or scene. As mentioned before, one must study different descriptors in images to be sure they represent a given concept. One uses a set of varied images representing an object and generates SIFT descriptors for each of them. The most recurring vectors in these images are saved in a list of "words" representing the object. This is called the "Bag of Words" technique [77, 155, 133, 130]. The more varied the images are in the set, the more robust the Bag of Words is, as it contains many orientations, positions, contexts, and general variations of the object it describes. One creates Bags of Words for different types of objects. Each time they want to analyse the content of an image, they extract its SIFT descriptors and compare them with the different existing bags. If one is close enough, the image likely contains the corresponding object. Not all the words have to be present to make a match, as each part of the object cannot be present in the image at the same time.

This combination of descriptors and Bags of Words has been the State of the Art (SOTA) in CV until the early 2010's. The descriptors are robust to many variations and the bag of words adds robustness to obfuscation and provides

detection. With this technique, one relies on data to create the description of an image. This idea of aggregating information from a wide source of examples has proven very effective in the domain of CV. Although it was used for a long time with histograms analysis or pixels intensity threshold, data-oriented algorithms gained popularity in the mid 2000's with this method and others (Support Vector Machines (SVM) for instance). It is called Machine Learning, and it is the main focus of the current methods in the domain.

2.1.2 Going Further with Machine Learning

Before the beginnings of ML, CV techniques were less relying on data. Some of the previous methods were automatically finding parameters to optimize a solution, but it was never at a big scale: the statistical models trained on data could not digest too much information. This restricted the performance a model could reach. Further, an important amount of algorithms were based on human biases. Edge detection or line equations, blurs, colour manipulations, all these methods relied on human ideas and human direct perceptions.

With ML, an important part of the intelligence can directly be found in the data. A human could at most create heuristics biased towards what they focus on, but it is often less comprehensive than data-driven algorithms.

Contrast and shapes being easier to describe than texture [137], SIFT was conceived similarly. If one generates SIFT descriptors for a face and studies what they represent [130], the ones centered around the eyes, the mouth, and other important regions, will be kept, as what they describe seems important to human vision. However, doing so would miss important descriptors on the cheeks and the forehead, because our eye is less focused on these regions as they lack in texture [201]. Using varied object representations and ML can alleviate these problems.

ML can be defined as the algorithms which, given a set of inputs and outputs, choose the parameters of a model to map them (*i.e.* this definition also applies to optimization, which has many similarities with ML). In our case, however, the input/output pairs are of different nature (an image and a label, for instance) and therefore the model and matching algorithms have to be complex. Also, despite having fewer human biases and being more powerful than many previous algorithms, ML has its problems and biases too [61]. Further, it depends on data (quantity and quality) to function accurately. Finally, the nature and complexity of the model itself are determinant of the quality of the results. In this section, we will detail these aspects individually, applied to a specific approach of ML, namely the Neural Networks (NN).

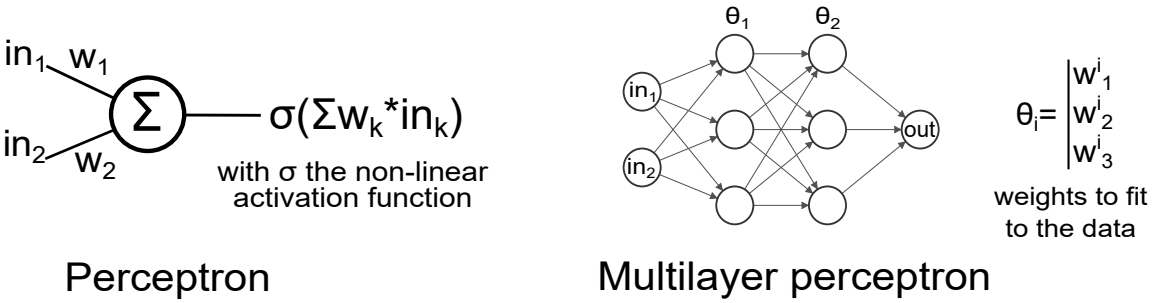


Figure 2.5 – The perceptron and a Multilayer Perceptron (MLP) architecture with 2 layers.

2.1.2.1 Training Neural Networks

In the mid 50's, Rosenblatt conceptualized the perceptron [173], represented in Figure 2.5 left, as an elementary processing unit, only performing an addition, a multiplication, and a non-linear operation. Combining many, organized in layers, resulted in a MLP architecture, as shown in Figure 2.5 right. The output is computed layer after layer, each inputting the output of the previous one, following Equation 2.1:

$$out(X) = z_n(X) = z_n(z_{n-1}(...z_0(X)...)) \ , \ z_i(X) = \sigma(z_{i-1}(X) \cdot \theta_i) \ , \qquad (2.1)$$

z_i and $theta_i$ being respectively the output and the weights of the i^{th} layer, σ the activation function. Such model is theoretically able to approximate any function. The more layers (*i.e.* the deeper the network), the higher the complexity of the problems it can solve.

This is the core idea of ML, as it revolves around a key concept in the domain: the difference between a formula and a model. The first one integrates as many influencing elements as possible, using prior knowledge of the environment. This results in an explicit formula with highly interpretable parameters. If enough elements are handled, the system is predictable with great precision. The number of parameters depends only on the system and the equations used to manage it. However, it is not trivial to find, especially with high-level notions: the formula inputting an image and outputting an object class is far from being understood. On the other hand, a model does not result in interpretable and limited parameters. Instead, it gives an approximation on a specific range and can be made of any number of parameters, often significantly more than the formula. Most importantly, a model is obtained using data and learning.

Given an input X, an output Y and a NN f, the problem is to find the NN parameters θ to map X and Y. At the start, the output $\tilde{Y}$ is different from Y: the model's parameters, called weights, must be adjusted. This is the role of the gradient descent algorithm [3]. With gradient descent, one computes the error E between Y and $\tilde{Y}$ and changes the weights following the error's gradient. This

results in a new, less incorrect model, and the operation is repeated until the error is low enough. There are many existing error functions, which must be (i) derivable (so must be f) and (ii) decrease as Y and $\tilde{Y}$ get closer. Formally, this follows:

$$f_\theta(X) = \tilde{Y} \neq Y \ , \qquad E(Y, \tilde{Y}) \xrightarrow[\tilde{Y} \to Y]{} 0 \ , \qquad f_\theta \leftarrow f_\theta - \alpha \times \frac{\partial E}{\partial \theta} \ , \tag{2.2}$$

with α the learning rate, a coefficient ($1e^{-3}$, $1e^{-4}$...) defining how much the weights will change from their original value in the gradient direction. Its value must be carefully chosen. If it is too large, the model might never converge towards a good solution, the optimal being distant of less than its value. On the other hand, a too small learning rate can trap the algorithm in a local minimum.

Several variations of the gradient descent algorithm have been proposed, such as the Stochastic Gradient Descent (SGD) [110] or Adam [112]. SGD provides a faster gradient computation for little precision loss. Adam adds gradient direction momentum throughout the steps to increase convergence speed.

Despite being suitable for many ML optimisation problems, gradient descent is not an *ah hoc* solution for NN. Although it is straightforward to compute the error for the output layer, there is no direct way to know how changing the weights of an intermediate layer affects the final output. The solution to alleviate that is called back-propagation. It was developed in the late 80's [174] and substantially improved during the next decade [122, 125]. Back-propagation computes the error's gradient through the layers using the chain rule, following Equation 2.3 for the layer $l \in [1, n-1]$:

$$\frac{\partial E}{\partial \theta_l} = \frac{\partial E}{\partial \theta_n} \left(\prod_{k=l}^{n-1} \frac{\partial z_{k+1}}{\partial z_k} \right) \frac{\partial z_l}{\partial \theta_l} \ , \tag{2.3}$$

$$\begin{aligned} \frac{\partial z_{k+1}}{\partial z_k} &= \frac{\partial z_{k+1}}{\partial (z_k \cdot \theta_{k+1})} \frac{\partial (z_k \cdot \theta_{k+1})}{\partial z_k} = \sigma'(z_k \theta_{k+1}) \cdot \theta_{k+1} \ , \\ \frac{\partial z_l}{\partial \theta_l} &= \frac{\partial \sigma(z_{l-1} \cdot \theta_l)}{\partial \theta_l} = \sigma'(z_{l-1} \cdot \theta_l) \cdot z_{l-1} \ . \end{aligned}$$

Intermediate layer's weights θ_i are optimized using optimizations of layers $i+1$ through n. This explains the name "back-propagation", as the original error gradient is propagated to each layer from end to start.

Before updating the weights, the gradient of multiple input/output pairs is computed, to parallelize the back-propagation. However, back-propagating the error of the entire dataset requires a lot of memory and is not very efficient as it allows only one update of the weights for the whole dataset. To use data more optimally, the back-propagation is computed by batch (*i.e.* subsets of the data), before updating the model. This allows a more frequent update of the

weights, thus faster convergence. Further, using batches reduces the risk that the individual gradients have opposed values, which would result in very small or null adjustments.

The choice of the batch size can be critical according to [108, 98]. The first shows the relationship between batch size and learning rate, proving their inter-dependency. It concludes by stating that with large learning rates, smaller batch sizes are the best to obtain the best model. It argues that for a given problem, it is preferable to start with a low batch size (*e.g.* 32) and a small learning rate ($< 1^{-3}$) and try increasing the batch size until performance decreases. On the other hand, [98] showcases how batch size influences a training's speed and stability. In general, the bigger the batch size, the more stable the training. The more there are elements in a batch, the less it is subject to data noise. As variance is reduced with bigger batches, the test model is also more stable by the end of training between epochs, contrarily to training with small batch sizes. However, a small batch size allows significantly faster training and fewer epochs to reach optimal performance. Both works further explain an important aspect of batched training: it creates a trade-off between the model's specificity and generality. A too specific model can be obtained with too small batch sizes because the gradient will correspond to only a sub-part of the dataset. Each batch will change the model in too different ways to adapt each time to too different data. As a result, the model may never converge to stable optimal weights. A too general gradient adaptation often results in no strong decision by the model (*i.e.* all the outputs have the same probability). Indeed, if the gradient of each input/output pair is calculated and averaged, it might result in a very small vector, as many elements may have opposed gradients.

After a pass of the whole dataset, one epoch is complete. It is usual to do several of them (hundreds, thousands...) to use every bit of information in the data, even if most information is learned during the first few epochs. The earliest iterations fit the model to the nature of the data, but the model's problem solving is processed afterward, with small variations of the weights. Intuitively, it is because a model needs to understand what an image is before telling what it contains.

2.1.2.2 Data: Solution and Problem

The weights of a NN are adjusted to fit the data. Therefore, instead of human vision biases, the models are based on data biases. The problem's different possibilities and variations must, therefore, be present in the data. If one wants a human detector and feeds only images of men to train a neural network, they cannot expect the model to detect women [204]. Such bias is obvious, but this is not the case for all. Still for the same task, one must find images of persons of every age, in every posture, under every lighting condition, etc. Similarly,

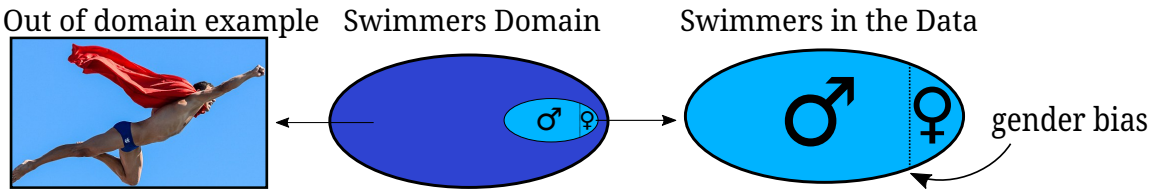


Figure 2.6 – Illustration of domain definition and data bias. The swimmers domain in the data (right) only represents a small portion of the entire swimmers domain (center). For a model trained on this data, the farther an image is from the training domain, the less likely it will be identified as a swimmers. For instance, Superman in swim briefs represented here will hardly be identified properly if the domain only contains classic images of swimmers. The domain thus needs to be as wide as possible for a given class. Further, data biases appear when the classes are unequally represented. In this example, there are more examples of males than of females, which causes problems for the future model's representation.

it is also important to have a variety of non-human objects that look like one, such as statues, photos, and monkeys, in every variation. This task is obviously not feasible, as there are infinite variations and possibilities. This results in two consequences: data biases and domain definition, illustrated in Figure 2.6.

The data domain can only represent part of the entire reality. Although models have generalisation capacities, anything outside of the domain risks not being fit for the end model, giving unpredictable results. This limit is very important to understand why sometimes NN work very well in experimental conditions, but not in real life: their data is not comprehensive enough of reality. Biases, on the other hand, exist because not all data elements can have the same representation in the dataset [141], as in Figure 2.6, right, with an over-representation of male images. In consequence, the data misrepresents female characteristics (size, standing, hair length, clothes...). During training, the average gradient will be pushed (*i.e.* biased) towards male attributes, as there are statistically much more of them in the batches. The resulting model will be more imprecise with images of females.

False correlations might also appear in the data. In [170], Ribeiro *et al.* trained a model to classify huskies and wolves. Figure 2.7 shows how the model considers the task: it only pays attention to the backgroafterwardsnow is visible, it considers it's a wolf. After observation, they understood that in the training dataset, each wolf image contains a snowy background. This correlation in the training dataset has no meaning in reality. This is the "shortcut-learning" problem [76]: if there is an easy-to-detect feature in the training set (usually low-level features, such as textures and colours), the model does not train further. The gradient tweaks the weight to get a more accurate precision on this specific feature. Again, if getting all possible contexts of each and every class was feasible, this would not happen, as this snow/wolf correlation will not appear in the dataset. But it is not currently possible.

(a) Husky classified as wolf (b) Explanation

Figure 2.7 – Raw data and explanation of a bad model's prediction in the "Husky vs Wolf " task. From [170]

Data is often considered the most critical part of ML. It always has biases, whether easy to explain or not, and its domain cannot represent the entire reality. Effective methods exist to alleviate these problems with domain-specific data augmentation [214], but the problem cannot be completely ignored. In the end, it is always important to know the dataset limitations, as they often define the final model's.

2.2 Convolutional Neural Networks

Although data is critical to training a model, certain NN are able to extract more or less pertinent features from them. To correctly analyse images, which concerns us in this thesis, simple MLP architectures are not very efficient. Images are spatially structured in a way that a model analysing them must be translation invariant. A MLP inputs the same area of an image and the same image shifted by a few pixels with two different input neurons. This might result in significantly different outputs, despite the same general input. In 1989, inspired by the pioneer work of Fukushima [144, 73, 72], LeCun *et al.* [121] proposed to merge 2D filters with NN learning algorithms to automatically learn the coefficients of a convolution kernel. This was the first of a whole new NN type, extremely well fitted for image analysis, called Convolutional Neural Network (CNN).

As seen in Section 2.1.1.1, 2D filters extract features from an image, resulting in what is called a feature map. A CNN does this iteratively, where each new input is the previous output. A layer is composed of several filters, therefore output has several feature maps (represented as its depth). They represent the manifestation of the different kernels, at the same spatial position in the image, as shown in Figure 2.8. The first layers are very similar to handcrafted 2D convolution filters. They detect low-level features on the image, such as colors, edges, corners, etc. At each layer, the features of the previous layer are combined, and through this

hierarchy, more and more abstract visual characteristics such as complex textures and shapes are thus extracted. Around the last layers of a CNN, the expressed features are often understandable by a human, as they react to the elements composing the object they were trained to understand. For instance, with human detection, the last feature maps can describe faces, legs, hands, or clothes.

This section explains in more detail the use of deep models applied to CV. This domain requires specific architectures and elements to perform optimally. One can select from a toolbox of multiple elements to construct a model, but they need to correctly manage them to obtain better results. Depth is also critical. Before recent improvements, it was seen impossible to go "too deep" and the use of shallow networks was the norm. This section also provides explanations on how this was alleviated.

2.2.1 Deep Learning

Looking at [121], one of the earliest CNN architectures, there are only 3 hidden layers. In [122], LeNet-6 architecture has 6. However, 2016's ResNet-152 [87] contains 152 layers. Such very deep model is part of what is today called DL, that is ML applied to deep NN. There is no exact definition for what "deep" precisely means. However, comparing the number of weights in the models clearly shows an increasing tendency over the years, up to the recent Natural Language Processing model GPT-3 [20] and its 175 billion parameters.

This section explains the different challenges regarding depth and NN size. Data, processing power, and architecture, this question concerns many parts of the domain.

2.2.1.1 Better Abstraction

Understanding how CNN convert images into concise, abstract, and meaningful data is the key to grasping the interest of deep models. Remark the following is true for DL architectures in general. Convolution kernels describe local patterns, so the abstract visual area squared in red in Figure 2.8, left, is represented by the (quantitative) red vector in the layer output. Suppose the filter #i corresponds to a vertical edge detector and that the value of channel #i at position (x, y) is high in the layer output. In this case, the model understands there is a vertical edge at position (x, y). All the other kernels of the layer represent a feature and their presence can be quantified by looking at their index in the vector. This is **abstraction**: transforming abstract local features on the image into concrete numbers in a vector. The red vector is an abstract representation of the image's red square area.

Going further, suppose at another position in the layer output, no dimension corresponding to edge detectors contains a high value. On the next layer, a filter

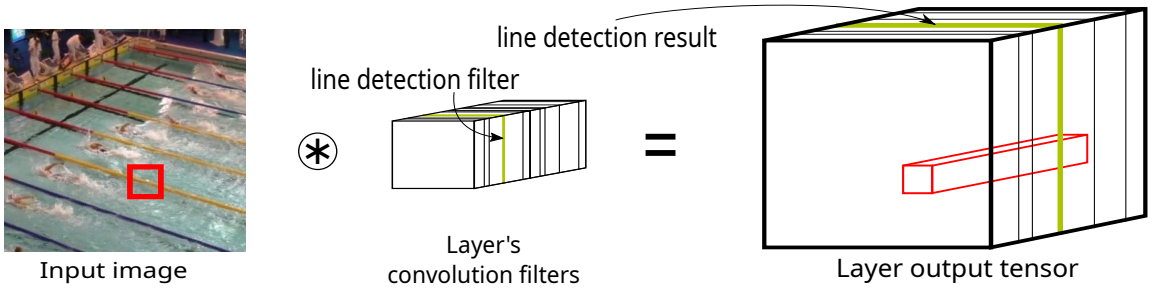


Figure 2.8 – Stacked convolutional filters forming one layer of a CNN. The "⊛" symbol represents convolution between the image and the filters. The input of the next layer is this layer output. The red square contains visual features, not easy to understand for a computer. The corresponding red vector, on the layer output, contains these information in a more understandable form for machines.

can pay attention (*i.e.* give high weighting) to these dimensions and be activated (*i.e.* output a high value): this would be a "low contrast detector", which is slightly higher-level than edge detectors. This behaviour propagates throughout the layers: low-level features are combined to compute higher-level ones. As such, each layer inputs the features collected by the previous layer and extracts more complex and meaningful features. The more there are layers, the higher-level the feature.

In fact, although the first layers are often texture oriented and the highest revolve around almost understandable concepts, it is difficult to explain exactly what happens in the intermediate layers. Visualisation tools [150] are able to give an intuitive understanding of the features, but it is not clear yet how depth improves abstraction. Further, the notion of abstraction is not quantifiable and thus troublesome to grasp.

Another answer is brought by [162], explaining ML models as Fourier function approximators. The Fourier transform of any complex function contains high frequencies. Further, they prove that approximating rapidly oscillating sinusoids requires more and more layers with the frequency increase. Therefore, approximating complex functions requires some depth in order to be precise enough.

2.2.1.2 Previous Limitations

As deep CNNs can powerfully abstract images into understandable data for computers, one can wonder why such models have not been used before. The general idea of adding layers to increase the representation is present since at least 1965 with [102], but DL started several decades later. The question has a multimodal answer concerning data, implementation, and processing power.

Despite having only 3 layers, the first CNN was very slow, as shown on this early digits identifier model from 1993 [124]. The computer was state of the art, the model very well implemented, however several seconds were required to analyse

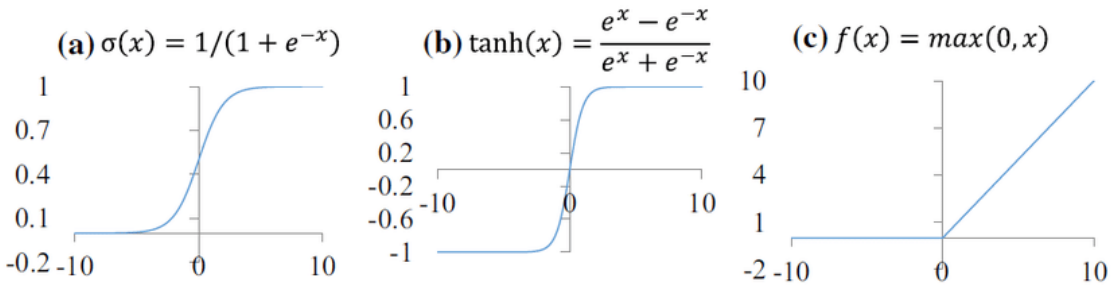


Figure 2.9 – Sigmoid (left), hyperbolic tangent (tanh) (center) and Rectified Liner Unit (ReLU) (rights) activation functions. The two firsts saturate when moving away from the origin, thus resulting in a weak gradient as their absolute value gets bigger. ReLU has a higher derivative for any positive value, enabling a better gradient back-propagation. Scheme extracted from [193].

a handful of numbers. This was due to hardware limitations, as NN requires powerful processing units to input an image and output a result. This was already slow and difficult to set up, so no one had the resources to train significantly deeper models in the 90's. To alleviate this problem, more powerful machines offered a solution. However, the bigger revolution came with the implementation of CNN inference on Graphics Processing Unit (GPU)s in [94], which claimed acceleration of 8 to 17 times. Indeed, a CNN can benefit from GPUs due to their parallelization power. As filters of a layer are convolved across the image with no interaction with each other, they can all be processed individually in the different cores of a GPU. This enabled to speed up drastically training and inference and is still used and improved upon today.

In principle, the deeper a network, the higher its abstraction powers. However, if shallow models can be completely trained with few examples, deep ones require significantly more, with as many label. As we will see in Section 2.3.2, we eventually found ways to alleviate this. Though, when NN were not as advanced as today, this was a problem. One needed to not only assemble a big set of images, but also to attribute them a label (*i.e.* an output value) to train a model on them. Even MNIST dataset [49] (1998, 70,000 images) and Letter Dataset [70] (1991, 20,000 images), which provided tens of thousands of images each, were not big enough for deep models with the current standard. The situation unlocked in 2009 with the release of Imagenet [48] and its 1.28 million images (back then) and thousands of classes.

The amount of training data and the speed increase of GPUs are well-known early limitations of DL. However, one last element has to be considered to finally enable the effective training of the models used today: the ReLU activation function. Before 2011, the activation functions (the non-linear function in between layers) were either the sigmoid or the tanh, illustrated in Figure 2.9 (left). Both these functions simulated the biological neurons, which pass a tension only if a certain

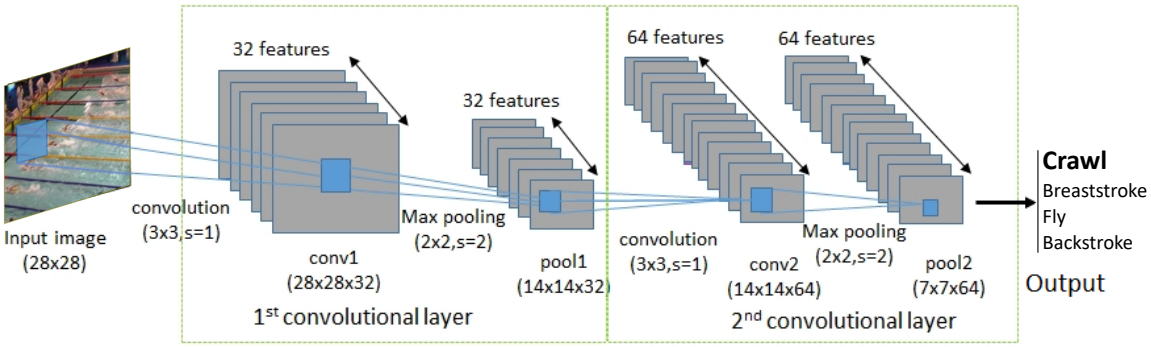


Figure 2.10 – A CNN architecture with 2 convolutions, each followed by a Max Pooling operation. The task at hand is classifying the swimming style of the input image.

threshold is reached. Their problem is that except around the origin, their asymptotic behaviour results in a very small gradient. Back-propagating it throughout the layers results in quick gradient vanishing: layers too far from the output could not be trained efficiently. ReLU [145], introduced in 2011 and illustrated in Figure 2.9 (right), proposed a much better solution for this problem. As the positive part is completely linear, the gradient is meaningful and is proportional to the error. The negative part is null, which behaves similarly to biological neurons, which do not pass tension under a certain threshold. The function is not derivable in 0 which could create problems to compute the gradient, but in practice, if the input is precisely 0 (very unlikely in float32 precision), one can decide to either apply the positive or negative behavior of the function.

In the end, it was the convergence of the GPU implementation (2006), Imagenet release (2009) and using the ReLU activation function (2011) which enabled the implementation of AlexNet [117] and its 60 millions parameters over 8 layers (2011-2012). Further evolution of deep models will be discussed in the next sections, but the main technical advances allowing the emergence of DL were achieved using these techniques.

2.2.2 CNNs Components

Different elements with various roles and importance compose a CNN architecture. Their importance is crucial to understanding how deep architectures are suited for given problems. This section will provide explanations for these elements, detailing what exactly composes a CNN. Figure 2.10 provides an example showing these different elements.

2.2.2.1 Receptive Fields

For computation purposes, the 2D filters present in a given CNN layer all have the same size. This size is called the receptive field of the layer and it is represented

in blue in Figure 2.10. Having a big receptive field enables to compute features on a big part of the input, therefore they contain more information. Following this logic, LeNet has (5×5) receptive fields for all its layers, and AlexNet has (11×11), (5×5) and (3×3), just to cite them. This is especially true around the beginning of the network, where the features are not complex yet: the bigger the filter, the more context it can give to a region.

In 2014 however, Simonyan *et al.* suggested in [185] to limit the receptive fields to (3×3) convolutions, which is the smallest size to capture the notion of left/right up/down and center. They argue that combining 2 successive (3×3) convolutions result in a (5×5) overall receptive field. One can obtain any receptive field size just using (3×3) convolutions. As this involves more layers, it also involves more ReLU activations, which increase the complexity of the representation with non-linear operations. Moreover, the number of parameters is significantly reduced: 3 successive (3×3) convolutions contain $27 \times C$ parameters for a receptive field of (7×7), while a single (7×7) contains $49 \times C$ (C being the number of channels on the layers). For a given amount of data, the fewer the parameters to train, the more each can be optimized (without considering over or underfitting). The more layers there are, the more abstraction the network can make. Therefore, increasing the number of (3×3) has two very important benefits. Szegedy *et al.* [191] even suggest going further, replacing one (3×3) convolution by a (3×1) and a (1×3), but it did not appear to significantly change the result, and the idea has not been broadly used.

It is also possible to use (1×1) convolutions, but they do not provide spatial understanding. Instead, they are used to linearly combine local features (followed by the activation), as in [87]. They offer a computation-wise cheap way to increase the network's complexity, having only $(1 \times C)$ parameters. In practice, they are used around the end of the architecture, once spatial features have been extracted and all that remains is to combine them for the task at hand. Another use of these (1×1) convolutions in [190, 177], is to reduce the number of channels thus reducing the number of parameters. In the paper, they compare it to "features distillation", where only the most important features manifold of the previous features is kept. This is known as linear bottlenecks, as illustrated in 2.13 and further explained in 2.2.3.2.

The most efficient existing architectures used today [87, 185, 191, 172] follow this rule of thumb: a big receptive field for the earliest layers (*i.e.* (5×5)), then several classic (3×3) to complexify the features and bring spatial information to the representation, then finally (1×1) filters to assemble these features so that they are suited for the given task. In Section 2.2.3.2 this will be nuanced, but the core idea will remain.

2.2.2.2 Spatial Sampling and Abstraction Increase

In Figure 2.10, in-between the convolution layers occurs a downsampling operation: the pooling. It extracts only one value per region (usually the maximum value, sometimes the average) for each channel. Also, the successive layers have an increasing number of channels, as in Figure 2.10 where there are 32 channels for the first layer and 64 for the second. These two parameters (spatial downsampling and abstraction increase) act together towards the same goal: to only keep the interesting features of the input.

As the network computes deeper and deeper information throughout its layers, it is interesting to get as many high-level features as possible for a more pertinent representation of the input. However, this also increases the data to save in memory [147]. It is not rare to have hundreds or thousands of channels. With a (224×224) pixels input, 1024 output channels, and a computation in float32, the tensor size is $32 \times 224 \times 224 \times 1024 > 1.6$ Go per image. Although recent GPUs can handle such data, it would be very resource-intensive, especially when each intermediate layer output has to be kept in memory for gradient estimation. Furthermore, even with parallel computing such layer takes a long time to be processed, forbidding real-time analyses [148]. Finally, most data in this support are in fact either useless (around unimportant areas of the input) or redundant (each neighbour pixel encoding almost the same information).

Max Pooling prevents these problems by only keeping the most relevant information. In a small region (usually (2×2) pixels), knowing whether a channel is activated or not matters more than knowing which exact pixel activated it [123]. The same article explains how the spatial organization is important for CV models: the feature's relative position with each others is the most important. Further, for global tasks looking for one result in the image (classification mostly), one does not care about where the elements are, only *if* they are present. For spatially precise tasks (detection, segmentation...), the channels tend to encode the spatial information, so reducing the tensor size does not change the result in too significant ways, up to a certain limit.

Furthermore, pooling enables a natural receptive field increase of filters [185]: with a (3×3) convolution applied just before a (2×2) downsampling, the surface described by each spatial element on the support size is doubled. If in the end, the output is $(1 \times 1 \times C)$, each channel encodes something about the whole image, which is very powerful. Combining pooling and channels increase converts local pixel distribution into global semantic meaning.

Instead of the Max Pooling operation, it is also possible to do convolutions with a stride of $N > 1$. The stride is the step between each convolution operation, so a stride of 2 for instance means only one of every 2 pixels will ever be the center of a convolution. This too reduces the output size. This method has some interests, as the filters will learn to handle downsampling by back-propagating through

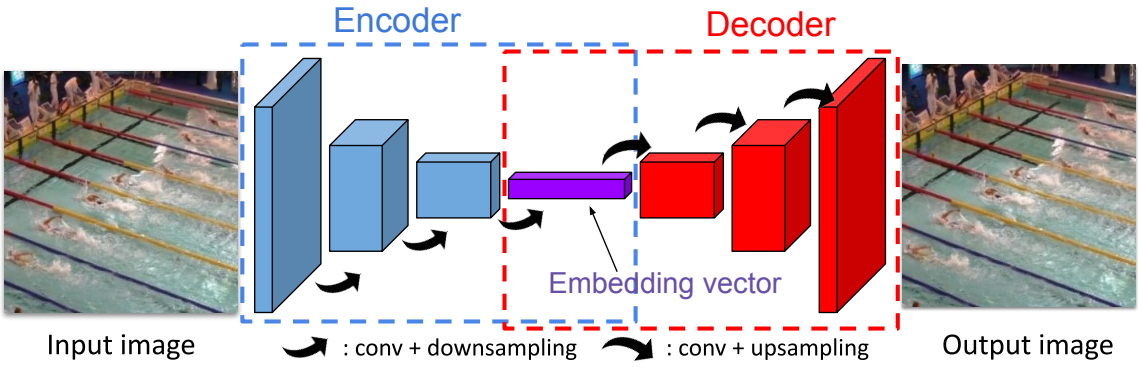


Figure 2.11 – An encoder-decoder architecture. As the input and output are the same, it is an autoencoder. The length of a block represent its number of channels. It is remarkable that the bottleneck of a linear autoencoder converges into the Principal Component Analysis (PCA) representation of the data.

them. Further, it is faster, as only part of the input is processed, while some of it is done for nothing as pooling discards them. Pooling is significantly more frequent, though, and one can argue it is more powerful as it compares the output of each position, contrarily to important strides.

2.2.3 CNNs Architectures

CNN components can be combined in different orders, with different parameters, forming an architecture. Depending on their nature, they can accomplish different objectives with different performances. To complete a task, one must choose between them all and eventually adapt them to fit precisely a problem. In this section, the main architectures used in this thesis will be described.

2.2.3.1 Encoder-Decoder Architectures

The encoder-decoder architecture, illustrated in Figure 2.11, can have different objectives, mostly related to image-to-image translation (out of scope for this thesis) or unsupervised training with autoencoders.

The first part of an encoder-decoder is the encoder, which has the most basic use of a CNN: encoding the information, *i.e.* transforming pixel distributions into a vector with semantic meaning of smaller size than the input. The resulting dimension reduction can be used for a broad variety of contexts, as encoders are usually only the first part of the network. For a classification task, fully-connected layers or (1×1) convolutions are added at the encoder's end to output a vector the size of the class numbers. For detection tasks, one adds a "detection head" (see Section 2.4) on top of the encoder. Note that encoders usually reduce the

data width and height, but this is not always the case, as in [80] where almost no pooling is applied to preserve as much spatial information as possible.

Encoders convert images into semantic vectors and decoders do the opposite. They are the architecture to generate images or pixel distributions with CNNs. They input a semantic vector that is converted into an image. The values present in the vector entirely define the output image in a similar fashion as the output vector of an encoder is defined by the input image. The elements composing decoders are similar to the ones in encoders, but they use upsampling instead of downsampling. This operation can be achieved in two main ways: either statistical interpolation algorithms (bilinear, nearest neighbour, etc.) or using transposed convolution layers. Regular convolution operations input an area of multiple values and output only one. They can be expressed as matrix multiplication to speed up the process. Transposed convolutions do the opposite and can be expressed as the matrix multiplication of the input with a transposed convolution matrix.

An autoencoder [93, 92] is an encoder-decoder architecture where the output is equal to the input, as illustrated in Figure 2.11. The model is trained to reconstruct the original image after a data compression [90]: the bottleneck (*i.e.*: junction of the encoder and the decoder) contains less data than the input due to the pooling layers. Different reconstruction losses can be used, mainly the L_1 and the Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i}^{n}(\phi(X_i) - X_i)^2 \qquad L_1 = \frac{1}{n}\sum_{i}^{n}|\phi(X_i) - X_i| \ , \tag{2.4}$$

X_i being an element of a batch of size n, ϕ being the function representing the autoencoder, hence $\phi(X)$ is the reconstructed image. These losses are complementary [85]. The MSE converges faster at first because quadratic functions penalize more big errors, but give less weight to small errors inferior to 1. L_1 has the opposite behaviour and both can be used at once, the MSE to quickly reduce important errors, the L_1 to make smaller adjustments. Such losses are not indicators of the quality of an image: a reconstructed image can have a low MSE compared to the original yet be blurry. The perceptual loss [156] addresses the problem, weighting high-level features instead of pixel-wise comparison. It relies on a trained model with frozen weights ψ, which outputs an embedding vector. One compares (usually with cosine distance) the embedding vector of the original image and the reconstructed image. Formally, this follows:

$$Perceptual\ Loss = \frac{1}{n}\sum_{i}^{n} cos(\psi(X_i),\ \psi(\phi(X_i))) \ , \tag{2.5}$$

As it is not possible to reconstruct lost data, the model focuses on encoding and reconstructing the visual features and colours that are the most represented in the dataset. As such, an autoencoder model trained on faces will perform

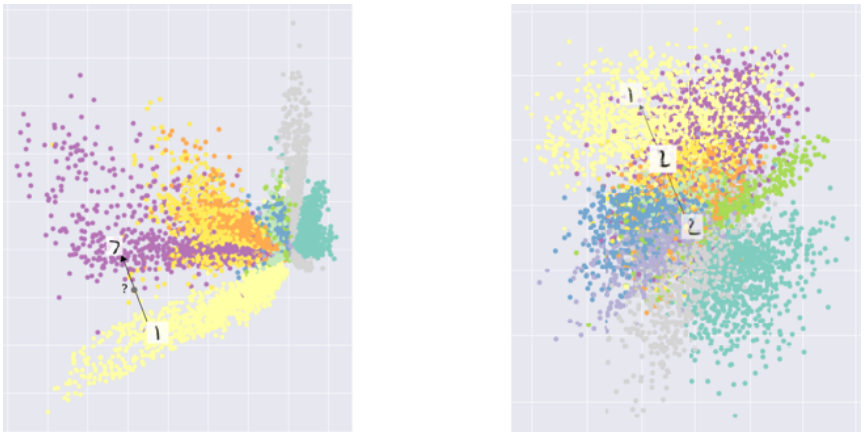


Figure 2.12 – 2D PCA visualisation of different autoencoders trained on MNIST. Colours represent classes. Left: an autoencoder with a non-continuous manifold in the latent space. Right: a VAE with a dense continuous manifold. Figure from [107].

poorly if fed pools, because the specific visual features they contain are not present in the original dataset. However, even well-reconstructed features can be too broad for a specific task. Indeed, an autoencoder trained for swimming races will be likely to reconstruct swimmers, but also spectators, stands, or the poolside, as they are a significant part of the training images, even if they are not interesting for race analysis purposes. Further, with distinct subgroups of images (*e.g.*: X images of pool A, Y images of pool B, etc.), the autoencoder will separate regions in the latent space (a distinct region per pool), as in Figure 2.12 left. If the latent space is sparse and not continuous, the features extracted from new pools will poorly describe them. A method to circumvent this problem is to use a Variational Autoencoder (VAE) [113]. Such model forces the continuity in the latent space by adding a regularization term on the distribution of the latent vectors: the Kullback–Leibler Divergence (KL div). This function measures the difference between 2 distributions. It can be used as a loss function to force the distribution of the bottleneck output at the bottleneck to be close to a multi-variate normal distribution. The exact implementation of a VAE is out of the scope of this section, but the result is a dense and continuous manifold at the bottleneck (as illustrated in Figure 2.12 right), hence a better encoder generalization.

2.2.3.2 ResNets

One inconvenience of the back-propagation algorithm is that the gradient is less and less significant after each layer: the output layer has a very precise gradient, but the input layer's is diluted and very indirect. As a result, the deeper an architecture, the less its first layers are trained. This is called *vanishing gradient* and it has two major drawbacks: (i) it slows down training by requiring more iterations to update the first layers enough and (ii) it increases the risks of overfitting, as

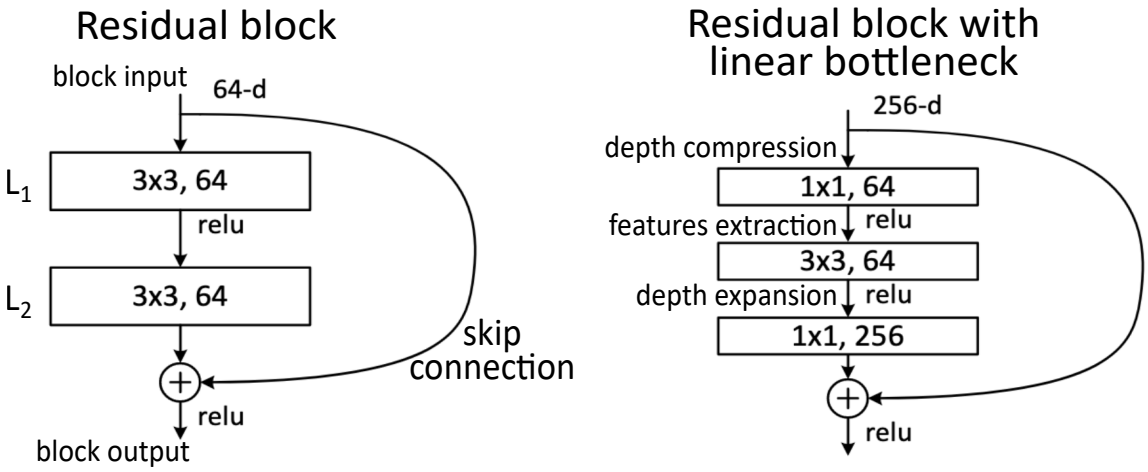


Figure 2.13 – Elementary residual blocks. Variations can be further applied to them, but the core feature is the skip connection, propagating gradient directly from the block output to its input. On the right, an illustration of the linear bottleneck, useful for deeper networks. The (1×1) convolutions create depth compression (256-d to 64) and expansion (64-d to 256). In between, a regular featurerepresentsion with (3×3) convolution is done with only 64 channels instead of 256.

shortcuts can be found to overcome the slow learning. To reduce it, it is necessary to add more and more data, but the needs for data increase too quickly, and very deep architecture are not feasible. The VGG-19 [185] architecture, with 19 layers, was considered very deep when it was introduced, and the authors mentioned the extensive experiments they had to make to reach convergence. Increasing the amount of data is thus not scalable to increase the depth of architectures.

A solution was proposed in [87]. In their paper, He *et al.* introduced the idea of one layer connected to multiple parts of the network, at multiple depths. Due to these "skip connections", part of the gradient is now directly propagated from one layer to any other. In Figure 2.13 left, the gradient back-propagates from the output to the input in two ways. First, the long way, through L_1 and L_2. The gradient has started fading away arriving at the block input. However, with the skip connection, the output's gradient also back-propagates directly to the input with no fading. Training is therefore more efficient, as even the first layers have a significant gradient.

This technique enabled very deep architectures. The most efficient way to make them is using successive feature extraction blocks, as in Figure 2.13. One can stack several depending on different trade-offs, in particular accuracy/speed, as deeper architecture are more accurate but slower. Variation of the ResNet architecture with 18 up to 152 layers [87], and all the other architectures that followed [191, 189, 99, 177, 192], are made of these blocks. Such models have a better use of data and processing power, as Figure 2.14 showcases. For the deeper ones (>50

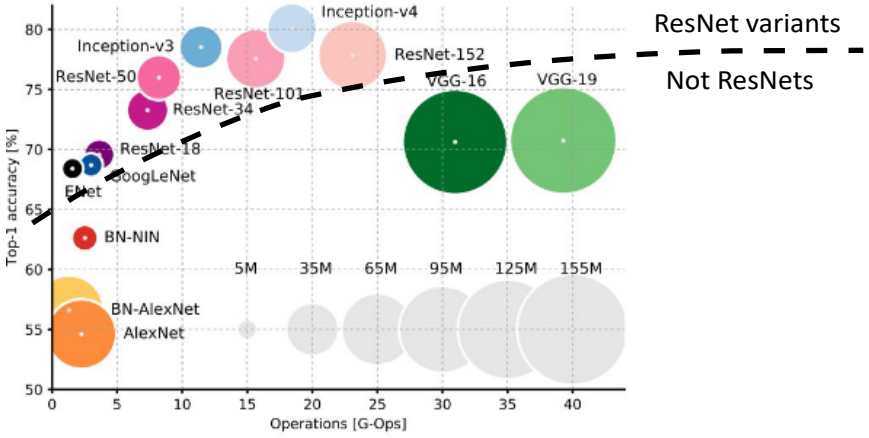


Figure 2.14 – Classification accuracy on the popular Imagenet CV benchmark [48] as a function of the number of operations (x-axis) and number of parameters (disc area). Above the dashed line are only ResNet variants. Despite significantly fewer parameters and operations, the ResNets perform substantially better than the other architectures. Scheme extracted from [24].

layers), linear bottlenecks are used to reduce the number of parameters by locally reducing the number of channels, as explained in 2.13 right.

2.2.3.3 Unet

Unet [172] is the combination of the ResNet and encoder-decoder architectures. It is composed of an encoder-decoder with a residual connection between blocks of the same depth at both sides of the network, as shown in Figure 2.15. The difference with encoder-decoders is that residual connections enable a direct propagation of the image's spatial content to the output. The architecture presented in Figure 2.15 is the original Unet, but as for ResNets, it is possible to create variants by adding deeper blocks, linear bottlenecks, changing the number of channels in the layers, etc. As long as it is an encoder-decoder with skip connections, it can be considered a Unet-like architecture.

The encoder extracts deep features describing the image. As the deep tensors are upsampled in the decoder, they are stacked with slightly lowwer-lever features but higher spatial precision. As a result, Unet is extremely powerful to perform segmentation tasks, where both semantic meaning and pixel precision are required.

Unet was originally developed for biomedical image segmentation, where data is often lacking. Such fully convolutional architecture can be trained with small amounts of data due to the multitude of skip connections. Indeed, usually, the farther a layer is from the output, the lower the gradient. With Unet though, the earliest convolution blocks are directly linked to the latest ones symmetrically.

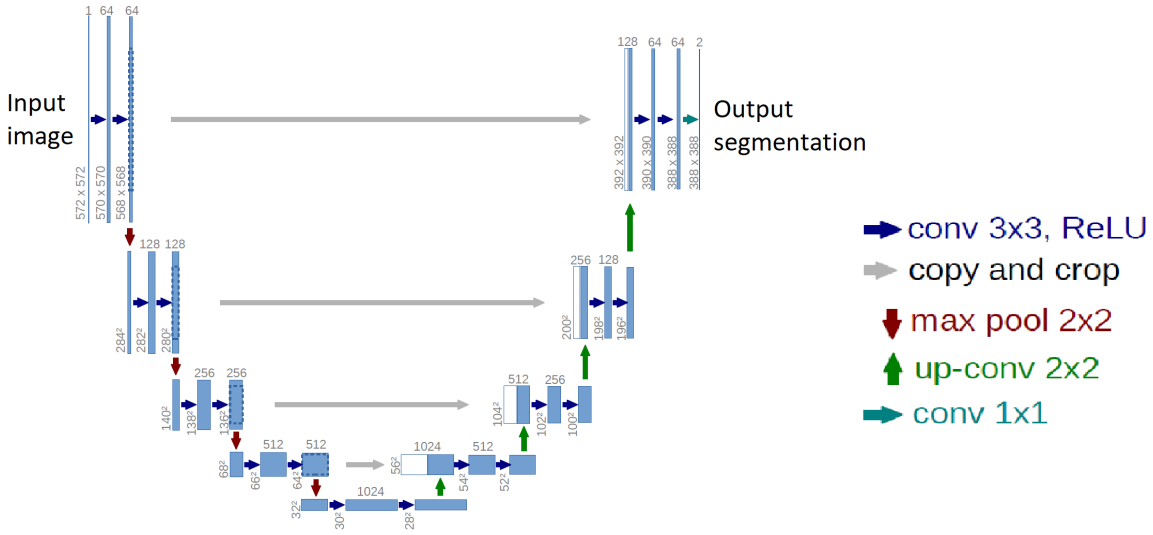


Figure 2.15 – The UNET architecture, from [172].

Therefore, this architecture has a gradient flow which enables fast gradient back-propagation through each layer.

2.3 Data and Supervision

Data is necessary to train a ML model, but its nature and amount depend on the available resources. Although it is hard to quantify, studies show data wrangling in general represents an enormous time of a ML model development (50-60% in [45, 46]). To obtain the best model out of the available data, different algorithms have been proposed, each tackling a specific configuration of data. In this section, different issues around data and training will be discussed, as these two important aspects of ML are entangled: data serves the training algorithm, but the training algorithm defines the data needs.

2.3.1 Computer Vision Datasets

Data has already been introduced as one of the key elements in the training of DL models. We already mentioned its possible biases and the limits of its domain. These paragraphs explore in more detail what data is, how it is obtained, and exactly how annotation is considered before the training process.

2.3.1.1 Data for Computer Vision

In the context of CV, a dataset is a collection of images that will be input into a model to train it. Depending on the context, each image can be associated with a

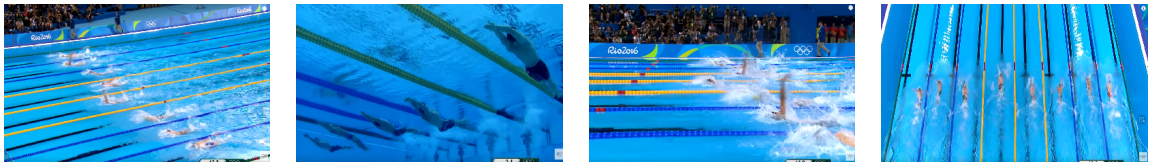


Figure 2.16 – Different views from a classic TV stream. Apart from the leftmost, they are very different from what coaches are used to.

label. The vast majority of labels are either a set of classes [**cifar**, 48, 49], bounding boxes [**coco**, 60], or segmentation maps [21, 74, 41], because they represent the 3 main challenges in CV. The first one is a vector the size of the number of classes. Each present class is at 1, the others at 0 (although specific variations can be made [210, 81]). Bounding boxes can be encoded in several ways depending on the method itself and they will be presented in detail in Section 2.4. Segmentation maps, finally, are matrices of the shape of the image, each pixel belonging to an object of interest on the original image set to 1. There can be several classes, represented by as many different 2D matrices which can overlap (*i.e.* different matrices can have the same pixel to 1) or not depending on the task.

Creating these target outputs is a manual, time-consuming process. The shortest is the classes, the second the bounding boxes, and the slowest the segmentation. It is complicated to estimate the annotation duration of each as it depends on the subject, but it is a different order of magnitude each time. For instance, to create Figure 2.1, the author needed 5s to create the classes, around 30 to create the boxes, and 3 minutes for the segmentation. Although it is indicative and not representative, it showcases how different the annotation time is for each data, even with the same image. In practice, creating a detailed segmentation map is very long. The widely used COCO segmentation dataset [**coco**] does not contain pixel-perfect annotation, but short straight segments defining the edge of the objects. Such approximation reduces enormously the labelling time without changing the data significantly.

2.3.1.2 Data Gathering

The labelling process is very long, but gathering images can be too. With the Internet [207], it is now easier than ever to create these collections of data. It is not always straightforward due to many problems (copyrights, modified images, small resolution, etc.) but it enables fast images collection gathering. In our context of swimming, this was an issue: although many swimming races (mostly from TV streams) exist online, they are subject to copyright. Further, TV way of filming is extremely constant, with only a few different shot angles, and only part of them exploitable for deep analysis. In Figure 2.16, the shots are from the same race, where the camera angles change regularly in significant manners, making very difficult the continuous analysis of a race. These angles are made for TV,

with a huge emphasis on dynamism and individual swimmers (with close-ups) instead of constrained angles with a wide view, adapted to analysis. As a result, gathering data from TV streams is not an optimal solution in our case.

In this thesis, we preferred to use videos from an online database of swimming race video and races analyses [66]. The majority of these videos are private and the access was kindly permitted by the Fédération Française de Natation (French Swimming Federation) (FFN). The races were filmed in varied conditions and using several camera positions (due to the pools' constraints at the time of the competition). There are all the swimming styles and both genders represented equally. From these hundreds of videos, we selected a dozen to represent in similar proportion the obvious classes (gender and style) to avoid class biases.

2.3.1.3 Data Cleaning

Once raw data is gathered, it is important to "clean" it. Data cleaning means removing every element that is not suited for training or testing data [101]. Unclean data can comprise multiple occurrences of the same image, modified data (photomontage or images with marks for instance are omnipresent in VOC [60]), too small images, unusual ratios (there is a (500x32) pixels image in Imagenet [48]), etc.

This process is essential to have the best model in the end, as unclean data can reduce the performance of a model by a significant margin [75]. Indeed, such data can present unique features towards which the computed gradient will be biased, during training. If said features are not representative of the final use case, this part of the gradient will only create divergence, resulting in a less efficient model. Multiple occurrences of the same image in a dataset also cause biases. An image presented N times will have N times more weight than the others during training and the resulting model will be biased towards its specific features. If said image is rare and contains valuable information, specific weighting can be given to it. Though, this is rare and such operation is made after data cleaning, with a good understanding of the usable data at hand.

Finally, data cleaning is also done after annotation to make sure the labels correspond to the data. This can be done with visualisation tools [29] or using crowd-sourcing methods [48] for big datasets, or manually for small ones.

2.3.1.4 Orders of Magnitude of Computer Vision Datasets

The more data there is, the better the resulting model. This rule of thumb is approximated for many reasons. A dataset can contain an enormous amount of poor-quality data, which would result in inefficient training. Likewise, the data has to be varied and represent the entire domain. Also, a model has its own limitations defined by its architecture and number of weights, limiting its ability to learn an infinite amount of nuances. However, the rule of thumb can generally

be followed if these elements are correctly handled. In fact, with the most recent architectures [197, 52] and their ability to digest huge amounts of data, this rule of thumbs seems to find no upper limit (see the different test sample numbers in [20]). Training with thousands, tens of thousands or even millions of images is frequent. Indeed, before the explosion of deep models, one of the most used detection datasets was Pascal-VOC [60] which counts 20,000 images in its 2012 version, with 20 different classes. To train deep models, this is undersized. It can still be used as a benchmark nowadays, but it is less considered than other datasets of higher orders of magnitude because a significant proportion of current methods cannot work with that amount of data. One of the most used datasets in CV is Imagenet [48] which has 14 million images, but not all are used for every case. For instance, the most highly used subset distribution is the 2012-2017 ILSVRC classification and localization dataset, which contains 1.5 million images "only". It is made of 1000 classes, splittable into 20,000 sub-categories. COCO [**coco**] is also a massively used dataset. It counts 200,000 labelled images with bounding boxes and segmentation masks of 80 classes (91 for COCO-stuff [22] which uses the same images). As an image can show multiple elements, the total number of annotated objects is 1.5 million. COCO also contains more than 100,000 images with no labels. These two datasets are representative of nowadays' orders of magnitude. State-of-the-art models depend on their size to function and could often not work with less data.

In our case, no dataset of the sort exists in swimming and it is not possible to create a comparable set during this thesis (it could be, but this would be outside of the scope). As such, this thesis will focus on better using few elements of data rather than over-sizing a model that will have enough data anyways.

2.3.2 Training: Different Levels of Supervision

Training a deep CNN can be done in several different ways which give significantly different results. Classic methods map the input/output pairs in the data, but such pairs do not always exist. Further, straightforward methods can be improved with prior or posterior training on other data. For a given task, the optimal training algorithm depends on the exact nature of the available data and the nature of the expected output. Though, both are often different, either by design or by constraint. The following paragraphs will describe the challenges associated with NN training in general and how they are associated with data. A scheme illustrating this co-dependent relationship between data and training algorithm is shown in Figure 2.17.

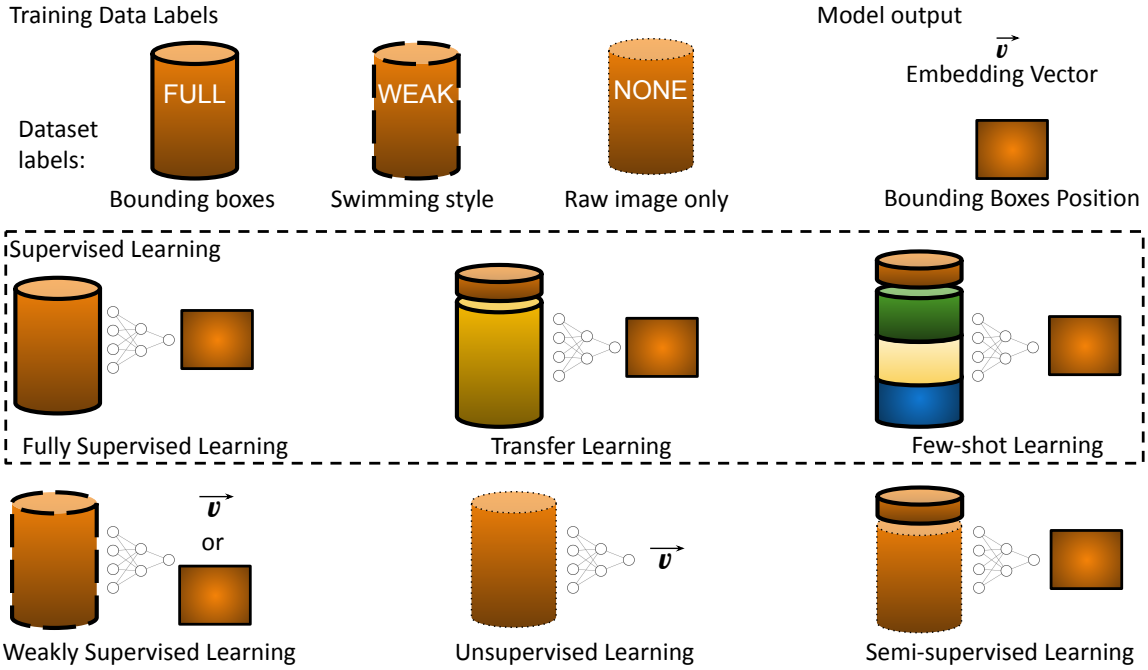


Figure 2.17 – The different levels of supervision. Different dataset colours represent different domains. There are different levels of label, here represented by the cylinders' edge. Although each type of supervision has a different complexity of training data, they all aim at performing the detection task. Note that for transfer learning, the "big" domain is distinct but close to the target domain.

2.3.2.1 Supervised Learning

Supervised training is the easier to grasp supervision level. It means the desired outputs are present in the data. Though, it can be more complicated than simply training a model from scratch on the dataset. In real life, it is frequent to have a small dataset for one's specific application, which is not enough to get acceptable results. To circumvent this issue, transfer learning [27, 16, 15] is frequently used. With this approach, one relies on a big dataset to train a first model. The dataset needs to share similar visual features with the final task at hand. Once the first model is trained, one freezes the feature extraction layers and uses a smaller specialized dataset to only train the last layer on it. This is illustrated in Figure 2.21, Phase 2. For instance, if one wants to classify the swimming style in images, they can gather a few images of each. However, it can be too limited to train a CNN on this small dataset. One can use a model pre-trained on another task involving swimmers, such as detection. One only keeps the encoder of the model, freeze its weights, and only train classification layers added on top of it. This works because the features extracted on the initial domain are similar to what the new task needs. The limit of this method is that the original and end dataset domains must be close enough. If the features extracted by the encoder are too different from what the end model needs, it will work poorly. Sadly, a swimming pool is a very specific environment, with very peculiar features related to how water and light interact. Our preliminary tests showed that transferring knowledge from a daily-life dataset (Imagenet, COCO, etc.) brings limited priors and that retraining all the layers is necessary.

An even more extreme case of this "annotated data lacking" problem is when one has only a handful of images per class (< 10). In this case, transfer learning is very limited because this amount of data is not enough to train the end layers of a model. This problem is called few-shot learning and it uses completely different techniques from transfer learning [62, 219, 67, 212]. A more formal description of it is how to create a model given N classes of objects with K samples each if K is small. Although data is lacking, this is still considered supervised training as one has a direct mapping of desired inputs and outputs. Indeed, with more efficient learning algorithms, this would not be different from regular supervised learning. For instance, one can desire to identify the swimmers in a race. They can crop images framing only the individual swimmers and train a model to identify them. If a new swimmer S, never seen before enters a competition, one desires to find them in other races. However, the swimmer S is not in the dataset, and there are only a few images of them. In this case, the better solution would be few-shot learning. To address few-shot learning, one must create a model with priors on a general domain. Then, the few elements of data will add knowledge to this prior in order to accomplish the task. This definition is extremely generic, though, because the existing methods addressing few-shot learning are very diverse. A

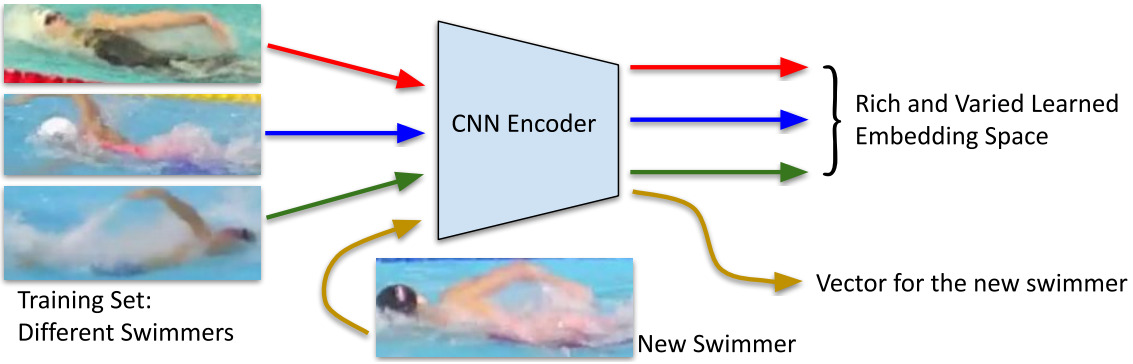


Figure 2.18 – An illustration of one-shot (the most extreme case of few-shot) learning applied to swimmers identification, inspired by [200]. The images from the new swimmer are embedded by the model. Afterward, each new swimmer image is compared to the embedding vector of the different swimmers, including the new one.

widely used few-shot classification method [200] consists in training a model with a sufficient amount of data on a wide amount of classes similar to the class one is interested in. This creates a model producing embedding vectors describing accurately the new class. The few available images of this class are fed to the model and the output vectors are kept. Then, one compares the output of new images with these vectors: if they are close enough (according to a metric and threshold defined by the user), it means the image features the class of interest. In [62], the authors compare the result with a dataset they introduce containing 1000 classes, but few images (hundreds per class), with the results from a model trained on COCO and its 80 classes (each with dozens of thousands of instances). The results (Figure 8 of [62]) show that the more there are classes in the base dataset to train the embedding model, the better the model generalizes for new classes. For our task of identifying swimmers, one can gather images of many of them to train a model in a fully-supervised fashion. When a new swimmer S, who has never been seen before, appears, the model can output a vector for one of their images. If the base dataset contains enough swimmer variations, the resulting vector will describe S in a discriminant manner. The distance to other swimmers' embedding vectors will be important while the distance to the known embedding vectors of S will be small. This idea is explained with Figure 2.18.

Few-shot learning is also addressed by other approaches, such as meta-learning [67, 212], which proposes to retrain a small model for each new class. This model outputs a vector that weights the output feature map of a bigger model. In the resulting feature map, the characteristics of the new class will be highlighted so that the last layers of the main model identify the class correctly.

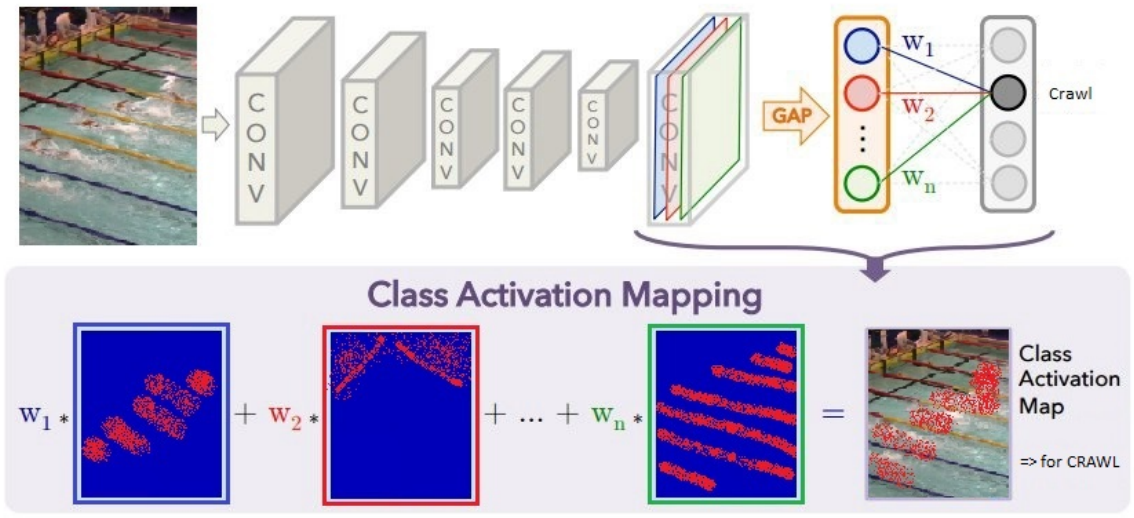


Figure 2.19 – The CAM pipeline explained (Figure inspired from [224]). The selected proxy task is to detect the swimming style. Each feature map of the last convolution layer's output is weighted by the coefficient it assigns to the crawl class (w1, w2, etc. for crawl). In our example, the weights to the 2^{nd} (red) and n^{th} (green) feature maps are low compared to the 1^{st} one's (blue), which roughly segments the swimmers.

2.3.2.2 Weakly Supervised Learning

The classic tasks of CV (classification, detection, segmentation) can be hierarchized by order of complexity, each inferior level being a subpart of the superior. This complexity can also be measured by the time required to annotate data accordingly. The challenge of weakly supervised training is to use a level of annotation during training and to output a higher level [128, 168, 36, 224]. The main interest for this is annotation time: labelling a classification dataset takes only a fraction of the time required to label a detection dataset. Likewise for detection with respect to segmentation. It can also be used for uncertain data. For instance, rare forms of pathologies exist where doctors are only sure an organ is malfunctioning, but do not know which cells are responsible for it. In such conditions, labelling a sick part is not possible but classifying sickness is trivial, as the patient is sick. Weakly supervised learning can thus be used to tackle the problem [211, 209]. A huge variety of methods address this challenge. We will explain one of them which is massively used: Class Activation Mapping (CAM).

CAM, introduced in [224], proposes detection or segmentation based on image-level annotation (*i.e.* class). This method relies on the fact that the features responsible for a classification result are localized in an image, thus in a feature map too. A CNN is trained on a classification task, with a global average pooling layer at the end to spatially reduce the feature map size, succeeded by a fully connected classification layer. Once training is complete, during inference, the

architecture is modified. The feature maps before the average pooling are kept and weighted by the coefficient associating them to a given class in the classification layer. The mean of the resulting heatmaps highlights the regions responsible for the classification. In the absence of strong biases, this corresponds to a segmentation of the class's instances in the image. This is illustrated in Figure 2.19. It is recommended to have few pooling layers before the final global pooling, as it reduces the precision of the final segmentation heatmaps. Improvements have been made on the CAM based algorithms. An NVIDIA team showcased limitations to the method in [168], and solutions to circumvent them. First, these algorithms do not separate close instances of the same class, which are merged into a big unique bounding box. The proposed solution is to use Multiple Instance Learning (MIL) to divide a region of interest into multiple sub-regions if necessary. In the article, the authors also highlight the fact that CAM only highlights the discriminant parts of a class (the head of a dog instead of its entire body, for instance). They address this problem following [36] which proposes "attention-based dropout". This method detects the regions responsible for the classification and replaces them with grey patches to force the model to find other discriminant areas.

Metric learning is also a way to create an encoder model with weak labels. Although it does not directly gives the expected end-result (detection for instance, if weak labels are classes), it is able to generate robust encoders suited to the data. It relies on a distance loss between embedding vectors. Several losses of the sort exist, such as the contrastive loss [83] or the magnet loss [171]. In this thesis, we focus on the Triplet Loss, defined as follows:

$$Triplet\ Loss(A, P, N) = \max(0, d(A, P) - d(A, N) + \alpha)\ , \tag{2.6}$$

where $\alpha \in \mathbb{R}$ is the margin, d is a distance function (traditionally euclidean or cosine), A is the anchor, P is the positive and N is the negative. The purpose of the triplet loss is to make the distance between the embeddings of A and N larger than the distance between the embeddings of A and P up to a minimum distance defined by α. The model only learns to position the input in the embedding space with respect to the other available inputs [206]. The other losses of the same nature also learn a relative position of the data in the parametric space, hence the general name "metric learning". With classification labelled data, one extracts 2 images of any class (the positive and the anchor) plus an image of another class (the negative). The model groups images of similar classes in the latent space and isolate these groups [216]. As shown in Figure 2.20, training a model on a classification task using softmax activation [19] in the end creates similar groups. However, the end distribution is very different: metric learning spreads its manifold on a wider space and there is no hard frontier between classes.

Weakly-supervised learning applied to swimmers detection has been experimented during this thesis, using CAM. We used swimming style classification,

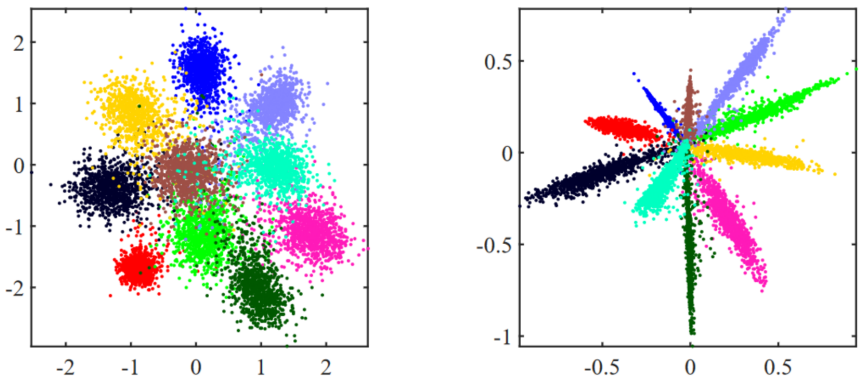


Figure 2.20 – 2D PCA of embedding vectors from encoders trained on the MNIST dataset. The colours represent the different classes. Left: the model is trained using metric learning. Right: the model is trained with an additional classification layer with softmax activation. Extracted form [96].

which is fast to label as all the frames from a race belong to said class. The results were promising but quickly became obsolete once we created a labelled detection dataset (see Section 2.3.2.5).

2.3.2.3 Unsupervised Learning

When no labelled data exist, it is still possible to create generic embedding vectors of input images. To achieve this, unsupervised methods are required. Contrary to the other levels of supervision, where the model fits a task, unsupervised models are adapted to the input data itself. This means a CNN model trained in an unsupervised fashion will represent the image in general, without focusing on task-specific properties. Each information present in a significant part of the image dataset will be encoded in an embedding model. In practice, this method has important limits, as it does not directly output information such as a class, object position, or segmentation maps. However, it is often combined with other methods (transfer learning, clustering, ...) to finally achieve this. The following describes two methods of unsupervised learning: autoencoders [93] and representation learning [33, 34]. Generative Adversarial Networks (GAN) are also a powerful method of the domain, but they have not been studied further during this thesis.

By definition, an autoencoder is an unsupervised learning model. Its main interest is the data reduction and abstraction at its bottleneck. The encoder part can be used as an embedding model for the type of images it was trained on [**ae_as_encoder1**, 42, 143]. The model can output a feature map with spatial data, which can be converted into an embedding vector with a global pooling layer.

Autoencoders can be trained with a noisy input and asked to reconstruct the denoised image [199]. This helps the model to learn a better representation of the

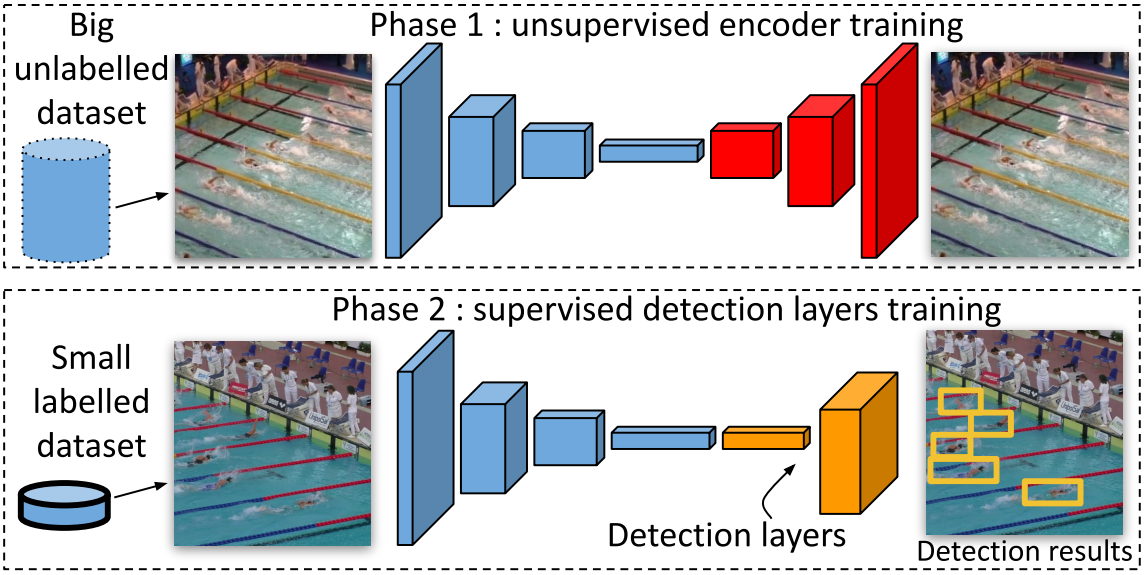


Figure 2.21 – A semi-supervised pipeline applied to detection. Phase 1: an autoencoder is used to train an encoder on a big unlabelled images dataset. Phase 2 (transfer learning): the encoder's weights are frozen and detection layers are trained on a small labelled images dataset.

content featured in the input instead of just reciting the content of an image. Such added artificial noise can be varied: blurr, (small) grey patches, colour changes, salt and pepper, etc... One must however be careful with it, as it may learn to miss relevant information for the end-task. For instance with detection, small regions can be interesting to keep. If the autoencoder is trained to reconstruct too important blurs, it may dismiss small regions to only focus on the general aspect of a region. On the other hand, specific augmentations can be used to push the training in the intended direction, such as zooming out to learn the importance of small pixel regions. Also, VAE having more densely uniform latent space, they are generally preferred for unsupervised feature extractions [169, 114]. They offer more general and adaptable features to further end-tasks.

A specific use-case of metric learning, called representation learning, can also be applied to unsupervised learning despite the lack of weak labels. To do so, one creates a pair of anchor and positive using data augmentation on one image [33, 34]. If said data augmentation does not change the content of the image in a discriminant manner (for the final end-task), both the non-augmented and augmented images have the same content. The negative can be any other image. This method forces the model to learn what similarity is in the dataset and how to alleviate data augmentation transformation. Therefore, the resulting encoder is robust to noise and learns a good representation of the dataset. As Figure 2.20 left shows, the embedding space is well covered: there are no major "holes" in it, contrarily to the right image where most of the space does not represent an

image. This means that although pertinent features are extracted from the images, no distinct prior classes have been defined: the result can be used for extremely diverse end tasks.

2.3.2.4 Semi-Supervised Learning

Semi-supervised learning [58] is the combination of supervised learning and unsupervised learning, illustrated in Figure 2.21. It uses both labelled and unlabelled data, the first in a significantly smaller amount than the latter. If the task is too complex for the small available labels, raw supervised training is not sufficient. Semi-supervised learning relies on a smoothness assumption [30] stating that if two elements are in the same cluster in the latent space, their output should have a close output in the end-task (although the notion of output proximity is complex to estimate with the detection task). There also is a manifold criterion, stating that the high dimension data information lies in a lower dimension manifold. Using unsupervised learning, one can create an encoder to reduce the input data dimension and still find the required information in the output. Said output being lower dimension, fewer parameters need to be trained to perform the end-task starting from it. Therefore, less data is required.

One can train an autoencoder on a given dataset and add layers on top of this encoder to perform the end task. The latent space of VAE being more convex, it guarantees a better smoothness assumption than raw autoencoders [51, 113]. It is possible to separate the unsupervised and supervised learning [169]. Doing so, the training is in two distinct steps: (i) train the autoencoder without the labelled data and (ii) use the (few) labelled data to train additional layers for the end-task in a fully supervised fashion. One can also merge these two steps, using both labelled and unlabelled data at once [114]. The model trains as a normal VAE with unlabelled data, but with labelled inputs, the learning is both made of a reconstruction loss and the end-task loss. It is also possible to fine-tune the entire network once the layers have satisfying weights, to get a more specific encoder. These methods using VAE have direct equivalents with GAN instead [142, 149].

Another frequently used approach, named self-training, relies on pseudo-labels [69]. A model is initially trained in a fully-supervised fashion using the available data. One then inputs the unlabelled data and keeps some. A new model is retrained using these pseudo-labels, and so on several times. If one retrains a model after the first iteration using all the pseudo labels (without dismissing the uncertain ones), this results in a model of the same precision as the original one [31]. Instead, it was proposed to use a fixed proportion of the best new elements [28] to improve results each time. Recently, curriculum learning (learning strategies starting with easier data to harder data) principles were used to improve on the idea [28, 217]. Pseudo-labels are also often used in other domains, such as weakly-supervised learning, as it does not require extra annotations [33, 34].

Semi-supervised learning has limitations. In [156], it is shown how small spots of rare colours are never reconstructed by autoencoders with only a reconstruction loss. Their bottleneck size space being limited, they prioritize the likeliest distributions. This directly translates in our swimming context, where a swimmer is small in a pool, and made of a significantly different colour distribution than water. An autoencoder prioritizes the reconstruction of the water and waves, as they represent a more important area of the image and thus allow a better reconstruction loss reduction. The amount of gradient focused on the swimmers during training is less important than for the uninteresting water. The resulting encoder is therefore very imperfect to detect swimmers, as it never learnt to focus on them. Further, training a generative model to use its encoder as basic for semi-supervised learning is not easy to optimize, as the reconstruction and the end-task need different features learning. Indeed, it was shown that a good semi-supervised learning encoder actually needs a bad generator [43].

2.3.2.5 Limits

Although the presented methods circumvent the lack of data, they present important limitations. First, the majority of works about non-fully supervised learning address classification, as it is significantly easier to train a model for this task than for detection. In the case of metric learning and few-shot in particular, the resulting embedding vector describes the image globally. Similarly to CAM, it is still possible to end the architecture by a global average pooling layer during training and to remove it afterwards. However, there is no insurance that the resulting model will highlight the elements one is interested in for the end task. This is the bet of transfer learning: despite the A and B domains looking similar to the human eye, a CNN model trained on the domain A might not detect similar features in domain B.

Further, the performance of supervised learning is significantly better than all the other forms of learning. For instance, the fully-supervised SOTA on the COCO benchmark reaches an AP-50 of 80.8 [220]. Weakly supervised learning SOTA [168] reaches only 24.8. With few-shot learning, in 1-shot, the SOTA is only 12.5 and in 30-shot it reaches 35.0 [219]. Finally, whereas the fully supervised mean Average Precision (mAP) (the AP-50 was not available in the paper) SOTA reaches 63.1, semi-supervised SOTA [218] reaches 26.1 with 1% of labelled data and up to 34.9 with 10% of labelled data. These SOTA methods are also computationally more expensive in training than basic supervised learning algorithm such as Faster R-CNN [167]. Their goal is not to compete against supervised learning, but their scores show that having more labelled data currently gives better results than any other method.

2.4 Object Detection

The task of object detection aims at placing bounding boxes around objects of interest. There can be any number of boxes per image, depending on its content. It is also possible to detect a given instance throughout a video: it is called tracking. The difference between an instance and an object is that an instance is unique, it is an individual of a class. On the contrary, there can be multiple instances of the same object. In a pool, a given swimmer is an instance of said class, and there can be up to 8 different instances of swimmer at once. In the case of tracking, the temporal aspect of the input is handled by the model to limit the field of research: the object cannot teleport, change too much colour or shape, vanish, etc. The object is generally selected by the user on the first frame and then tracked during the remainder of the video.

Before deep learning, detection methods relied on region proposals algorithms [196, 5, 57], which isolate regions of the image using heuristics on texture, colour, and edges. They use iterative processes to merge neighbour regions if necessary or refinement on filtered images. Such processes are extremely time-consuming (several seconds for a single (255×255) pixels image) and they have the usual human biases. Random forests were also frequently used [1]. They input several patches from an image and study their distribution using multiple binary trees. These trees determine whether the patches belong to a given object and where the object is with respect to the patch. They vote for a possible location of the object, resulting in a presence probability heatmap, allowing detection These method is also slow, even the fastest ones barely being "real-time" [9, 182] (on respectively 2005's and 2013's hardware, however). Finally, SVM used with a feature extractor such as Histograms of Gradient (HOG) is able to perform object detection [136, 154, 223]. The SVM is trained to classify the features extracted by HOG as one of a given set of classes (or background). Sliding windows of different sizes scan an image, each time being fed to a HOG extractor and the features to a SVM, resulting in detection.

Recent approaches can be divided into two groups: real-time and not real-time. The first offer obvious speed advantages, but they usually lack precision. Most of them are one-shot detectors, which means a model inputs an image and outputs a result directly. The other methods are more robust, but usually more memory greedy. Further, they are usually multi-shot detectors, *i.e.* multiple sub-regions of the image are analysed individually, which explains the additional time.

In this section, different aspects of object detection will be discussed. Two of the main types of algorithms will be explained, but also important algorithms revolving around detection.

2.4.1 Around Object Detection

The newest object detection techniques are complex and present unique ideas. However, they often rely on similar pre or post-processing algorithms. They will be explained first in order to get a better understanding of the detection algorithms once they are explained.

2.4.1.1 Encoding Bounding Boxes

Bounding boxes are rectangles that describe the location and size of distinct objects in an image. They can be encoded in several ways. Usually, methods either regress two opposed corners [119], the width/height and the center [167], or the width/height and one corner [163, 165]. In every case, the box is defined by 4 coordinates. These different methods each present different properties (discussed further in this section), but to our knowledge, no ablation study has been proposed to determine the best among them. Further, the different papers using them generally introduce other new concepts, and the choice of the box encoding does not seem to be significant compared to them.

Objects can have any height and width, but if the size of an object is too different from what the model has been trained on, the method will fail. This issue is likely to happen, as depending on the distance between the camera and the objects, or their orientation, they can have almost any size and aspect ratio. A small error on a small bounding box is more important than on a big bounding box, therefore normalisation methods thus must be used. Also, a region can contain both zero, one, or several elements. Detection algorithms must handle these different possible issues. To address them, [167] proposed the idea of anchors, illustrated in figure 2.22. It consists in comparing each bounding box to a set of fixed-size rectangles and taking the closest one (*i.e.* with the best overlapping). The width/height adjustment is done around the width/height of this box: this is a normalisation. In [164], the author used k-mean clustering on the training dataset to obtain the best-fitted anchor sizes: the mAP increased by 7% compared to handpicked box sizes. The number of anchor boxes also influences the results, as more sizes mean a smaller standard deviation for the different boxes attached to a given anchor. More boxes also increase the inference time and the amount of necessary data, so there is a trade-off to define for a given application.

A box can also be associated with a confidence (or objectness) score, *i.e.* the probability that an object is contained in it. If it is under a threshold, one does not consider the box. An algorithm can thus generate too many boxes, each associated with a score, to increase its chance to detect every object.

Other methods do not rely on anchors. As expressed before, CornerNet [119] computes the coordinates of two diagonally opposed points. More precisely, it associates each pixel of an image to either the top-left or the bottom-right value,

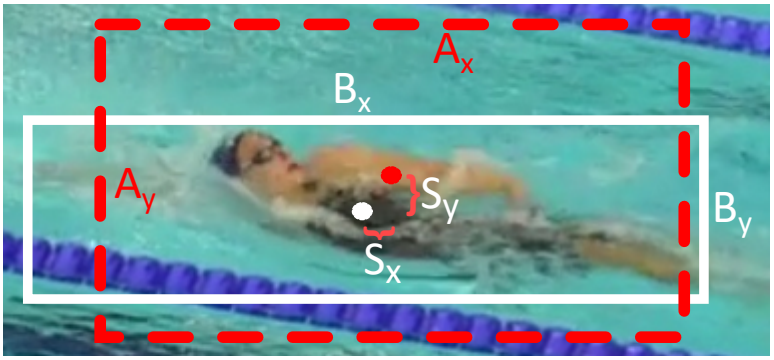


Figure 2.22 – Encoding a bounding box with respect to an anchor box. Here, the selected encoding uses width/height and distance from the top-left corner of the image to the box center. Here, the anchor is centered on the image and there is a shift of (Sx, Sy) with the bounding box. As the anchor is not exactly the same size as the box, a refinement regression is required.

plus an embedding vector. If a pair of corresponding points have a similar enough embedding vector (according to a similarity threshold), they form a bounding box. CenterNet [226] extends this idea by adding the center, an object being formed by a triplet. ExtremeNet [138] detects the 4 corners of a box, which is not necessarily a rectangle in the end. RepPoints [35] expresses an object as a set of points in the image, then determines the spatial properties of the obtained points cloud to define bounding boxes. It is also interesting to mention [59], which regresses a fixed number of boxes (100 or 200) using the corner and classifies each box as an object or background: it circumvents the challenge of having any possible number of objects on the image by oversizing on purpose that possible number. Another idea is to perform detection as a segmentation task, outputting a heatmap per class [224]. This heatmap is thresholded and the smallest rectangle around each remaining blob is used as a bounding box. The limit of this method is that if two elements of the same class are too close, they will have one big box around them both.

2.4.1.2 Non-Maximum Suppression

Depending on the method, multiple bounding boxes may be generated for the same object. In this case, using an Non-Maximum Suppression (NMS) algorithm prevents having too many false positives. This problem and the expected solution are illustrated in Figure 2.23. We precise that overlapping boxes from two different classes are not an issue: NMS only concerns boxes from the same class. The main selection criteria are the bounding boxes' objectness score (*i.e.*: the probability it represents an existing object, more on that in the next section) and the boxes overlapping. To measure this last element, the Intersection Over Union (IOU) metric is used, along with an IOU threshold.

Being only a supplement of the algorithm, NMS is rarely studied in detail and is generally not the core contribution of a paper. Although it can be learnt with ML [97], the majority of existing methods rely on a handcrafted greedy method with a threshold. It works as follows. First, one sorts the bounding boxes by confidence score into a pile of candidates. The hypothesis is that even if the box with the highest score overlaps with many others, it will be kept, as the model considers it to be the most secure box. The stack of accepted boxes begins with only the highest score box, while the stack of candidates contains all the others. One extracts the first from the (sorted) list of candidates and compares it with all the accepted boxes. If its IOU with one of them is beyond a defined threshold, it is rejected, if not it is added to the list of accepted boxes. This goes on for all the candidates.

This greedy simple approach is far from perfect. The definition of a threshold is arbitrary and impacts significantly the result. Depending on the context, this threshold must be different, although it is rarely changed in the applications. Nearby elements occluding each other naturally have overlapping bounding boxes, however NMS algorithm often disregards one of the boxes. As hidden objects generally have a lower confidence score, they are filtered out. On the other hand, a single object can be associated with two boxes that don't overlap too much and be considered as different instances by the algorithm. This also does not filter out duplicates caused by the neighbour. One frequent error with current detection models is they detect an animal *and* its head, as two different instances of the same element. As the two detections overlap significantly - one being part of the other - it could be expected that the NMS removes the incorrect one. However, it generally does not, because the head occupies a much smaller area than the entire body.

Obvious false positives are not affected by NMS and good positives are frequently thrown away. Apart from the learning methods mentioned before, Soft-NMS [17] has been proposed as an iterative improvement. It uses the same general algorithm as greedy NMS with a small adjustment. Instead of removing a candidate box overlapping too much with an accepted box, the objectness score of the object is penalized in function of its IOU with the accepted boxes. Therefore, there is no IOU threshold anymore. In practice, if the overlap is too important, the end confidence score ends up lower than the acceptance threshold: the box is discarded. Otherwise, if after comparison with all the accepted bounding boxes the confidence score is still above the acceptance threshold, the box is kept. The score penalty function follows a Gaussian with a standard deviation $\sigma = 0.5$ in the article (the method is little sensitive to this parameter), as follows:

$$Score \leftarrow Score \times e^{-\frac{IOU^2}{\sigma}} . \tag{2.7}$$

This method improves by a significant margin the pre-existing models without further training or threshold (to be precise, it removes the IOU threshold), which

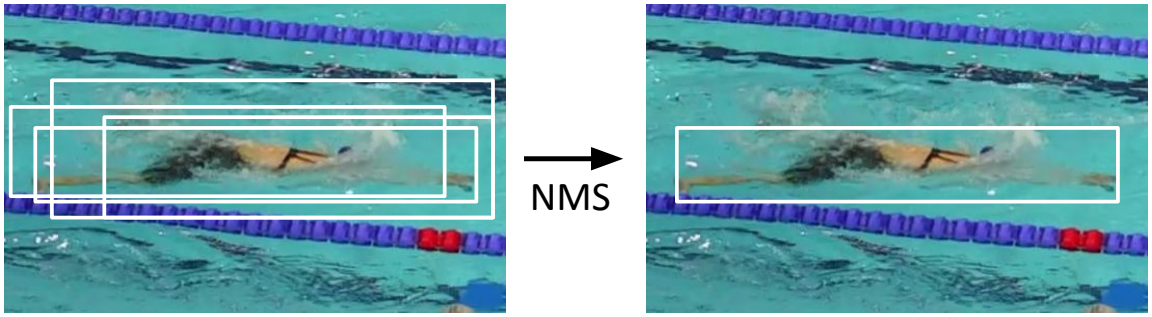


Figure 2.23 – The problematics and result of NMS.

is remarkable. Though, this algorithm will not be critical for our pool context, as the elements we want to detect are not overlapping at all (the swimmers, the markers, etc.).

2.4.2 Multi-Shot Object Detection

Multi-shot object detection algorithms input several times either an image or a selected crop. This allows refinement over an initial detection, thus great precision. The current SOTA algorithm is DINO [220], reaching an Average Precision of 63.1 with the heaviest backbone (Swin transformer [132], 218 M parameters) and Imagenet pre-training, and 51.2 with a ResNet50 and no pretraining. It is based on different modules, each responsible for a given task: feature extraction with the backbone, query selection, matching, etc... In fact, this model aggregates the ideas from several previous papers [26, 188, 227, 132], each proposing a module increasing ever so slightly the performance. It also proposes a new module (here, contrastive denoising). It is also based on multi-scale features, acting similarly to the different size sliding windows used with SVM and HOG.

This method of searching at multiple scales and creating several queries, each separately analysed by a refinement module is the foundation of multi-shot object detection algorithms. Detailing DINO requires detailing the several methods it is based on, as well as the methods they are based on, and so on. Instead, we chose to present in detail the original method that inspired them all.

The R-CNN (Region-based Convolutional Neural Networks) family of algorithms brought very powerful new ideas for object detection. They decompose the task in two stages: (i) detecting Regions Of Interest (ROI), (ii) validating/adjusting said region. The different algorithms [80, 79, 167] each proposed new concepts for these stages by refining pre-existing methods. In this section, we will mainly discuss the last one (Faster R-CNN) with mentions of the others. It consists of 3 main elements: (i) the backbone, that is the CNN model used to extract features from the image, (ii) the Region Proposal Network (RPN), and (iii) the classifier, as illustrated in Figure 2.24.

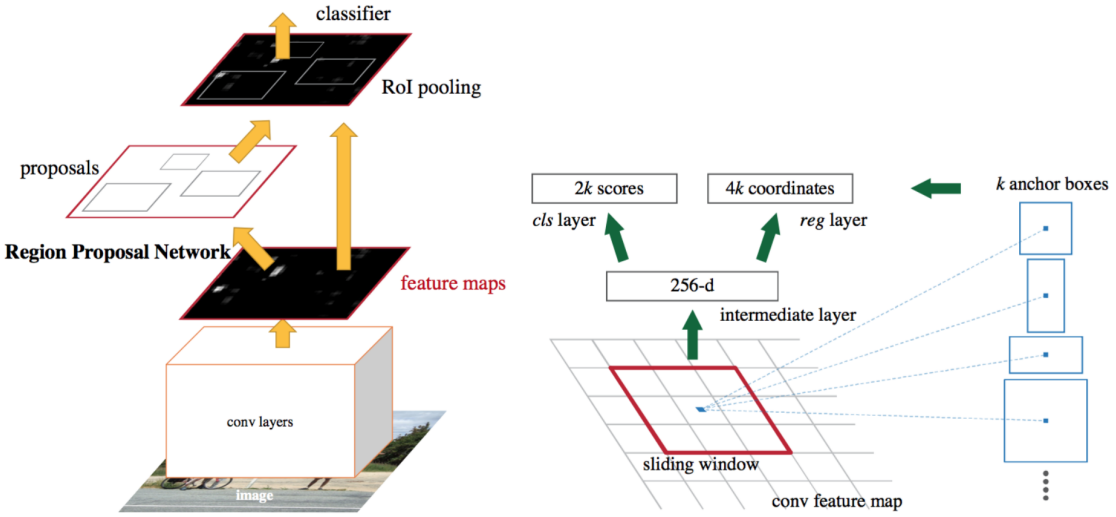


Figure 2.24 – The Faster R-CNN algorithm. Left: the shared extracted features of the ROI proposal network and classifier. Right: how the anchor box and the classifier output bounding boxes. Figure from [166]

The objective of the RPN is to tell whether a box contains an object or not, and (if it does) to adjust its coordinates to the found object. First, one defines a set of k anchor boxes with given sizes. Each is associated with a number between 0 and k. The bounding boxes are associated with an anchor with the best area matching and to a coordinate on the resulting feature map. The RPN model is trained to map an image to the anchors placed at the position closest to the center of the boxes. To that extent, a CNN (the model's backbone) extracts deep features from the image, then two parallel (1×1) convolution layers output a tensor of depth 2k and 4k, as illustrated in Figure 2.24 right.

The objective of these branches is to determine whether an anchor box contains an object or not, and (if it does) to precisely adjust the coordinates to fit the object The 2k output classifies each bounding box as either an object of interest or background. The original paper proposes to use 2k outputs, but only k are required if the information is encoded differently (0 for background and 1 for an object, for instance). This is the objectness score. The 4k branch regresses the exact size of the object compared to the anchor. If no object is to be found (the other branch outputs the class "background"), the expected result is 0. If not, it outputs the positional information (*i.e.* the shift and warp parameters to obtain the box from the anchor). To be more specific, the model outputs a function (defined in [80]) of this information, as they are normalized around 0 and are not made to reach a too extreme value (else, another anchor should be used). For the 2k classification branch, the dimensions $\{2 \times i, 2 \times i + 1\}$ are trained relatively the bounding box i. Likewise for the dimensions $\{4 \times i : 4 \times i + 3\}$ for the 4k regression branch.

To select the candidate ROIs, the authors propose to take the N (usually between 100 and 2000 [80, 59, 167]) highest objectness scores (independently of their value). One obtains N potential ROI, defined by the anchor box and the shift and warp provided by the regression branch, resulting in a unique rectangle. Other approaches using anchors [163, 164] propose instead to take every box with an objectness score above a certain threshold.

The last stage is the classifier. Each regressed box is analysed independently from the others. The model has a fix sized input of (w, h, d), d being the depth of the tensor outputted by the backbone (typically 512). On said tensor, a region corresponding to the box is extracted, resulting in a (W, H, d) tensor. A max-pooling operation is done so that the (W, H) region is transformed into a (w, h) region. This tensor is inputted to the classifier, which, after a few fully connected layers, outputs the object's class and a probability that an object of said class is in the box. This second probability is different from the objectness score: it ensures the detected object is well framed by the bounding box. Indeed, it is common for an anchor to have a high objectness score despite being too far from the targeted object's center. In this case, the bounding box regression will not likely be well-suited to the object. This second probability score ensures these cases are rejected by the model. Finally, a greedy NMS algorithm is used to avoid box redundancy. This is extremely frequent with Faster R-CNN, as the input and consequently the output have a high resolution, thus two neighbour regions of the output tensor describe a similar content.

Faster R-CNN is a very powerful object detection model. Because it uses several anchor scales, it is great at detecting small objects, which is a common problem in object detection. The entire architecture is trainable in an end-to-end fashion, which eases the training process. It is also possible to use transfer learning on other tasks to initialise the weights of its backbone (generally using Imagenet). Further, by sharing the convolution layers for the RPN and the classification, it reduces its inference and training time by a significant amount (10 $\times$ according to [127]) compared to Fast R-CNN (the previous version of the algorithm). However, it is still very slow, 3-4 Frames per Second (FPS) on average. The many ROI analysis takes some time, especially considering the strategy of taking the 300 to 2000 best examples, as suggested by the authors. The backbone inference is also heavy: it is better to use big images (600 $\times$ 1000 pixels in the paper) to have a higher resolution for the RPN and thus be more precise.

2.4.3 Single-Shot Object Detection

Single-shot detectors is the real-time pendant of object detection. Whereas multi-shot object detection usually are **[SOTA]** according to precision metrics, they are way slower. Although the paper of DINO states 10 to 24 FPS, it is only made possible with the GPU it uses, an A100 NVIDIA with 80 GB of RAM. It

also uses the ResNet backbone (and does not specify framerate with the Swin Transformer backbone).

In opposition to this, are thus the real-time detectors. They generally input the image once, with no sub-patches division or multiple refinements over the same area. As such, they are sing-shot detectors. The two first main algorithms actually explicit it in their name: YOLO (You Only Look Once) [163] and SSD (Sing-Shot Detector) [131]. They both rely on the same underneath technique: a backbone for feature extraction, directly connected to regression and classification layers. Their main difference is in the architecture, SSD stacking different scales of feature maps in order to get more precise results, but at the cost of time. As such, YOLO will be explained in this section (with its variations up to the third version YOLO v3 [165]). The fourth and fifth versions mostly differ in implementation additions and training optimization.

You Only Look Once (YOLO) is an extremely popular algorithm with multiple incremental improvements [163, 164, 165]. Its quality comes from the simplicity to understand it, and the fact that it was one of the first real-time detectors. It performs detection and classification at once using anchor box adjustment.

Its behaviour is straightforward. A backbone extracts features, resulting in a (W, H, d) tensor. One can see each spatial dimension of the tensor as a region's deep representation. YOLO assumes only a small number of elements can be present in a sub-region, as it is small compared to the input image. In each sub-region, YOLO regresses k bounding box, each associated with :

- positional information (4 values $\in \mathbb{R}$)
- an objectness score (1 value $\in [0, 1]$)
- a class vector probability (C values $\in [0, 1]$).

With C classes, the output of YOLO contains a number of elements equal to $E = (4 + 1 + C) \times k$. As such, the global output size of YOLO is (W, H, E). The anchors associated with a high enough objectness score are kept (typically 0.8). This results in boxes one can position on the image, as illustrated in Figure 2.25 (center). A greedy NMS is applied to filter the redundant boxes, resulting in the boxes of interest (Figure 2.25, right). Here, the objectness score predicts the IOU between the object box and the anchor box: it is more nuanced than a binary 1/0, thus allowing a better box choice during NMS. The shift to apply to the anchor is measured from its top-left corner to the top-left corner of the sub-region it belongs to. This allows a more squeezed shift values distribution than if the reference was the corner of the entire image, thus stabilizing the regression.

Linking the backbone's output and this "YOLO head", can either be a fully-connected layer or a (1×1) convolution. The first provides general information on the entire image to each sub-region (good for general context understanding) and the second is less costly in parameters (better when few data are available). One can only consider the bounding boxes with an objectness score above a threshold to filter boxes. The images are reshaped to a fixed size of (416×416 in [164]) to

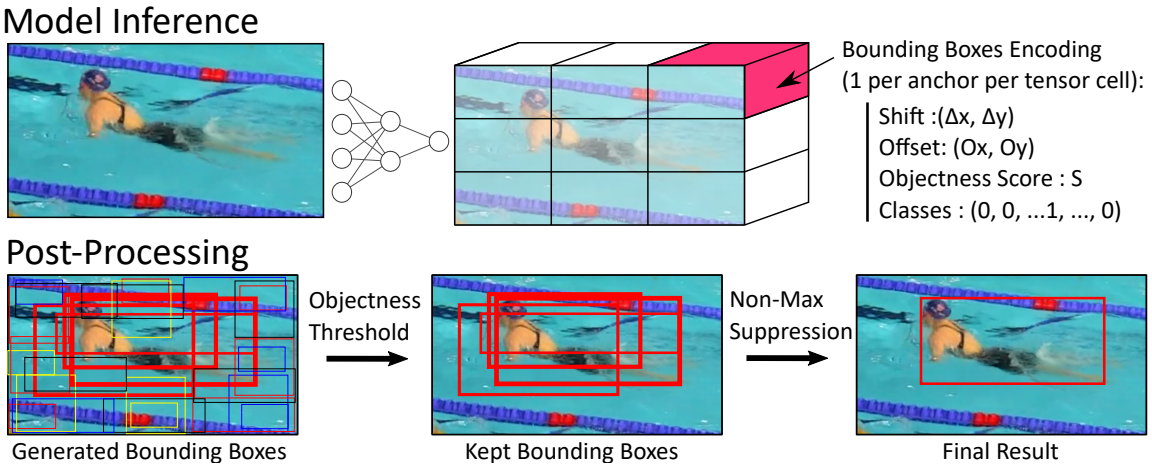


Figure 2.25 – An illustration of YOLO. The output tensor is only (3×3) for simplicity. Boxes colour represents their associated class. Objectness score is represented by boxes thickness. The model outputs an encoding for each anchor of each sub-region. Only the ones with an objectness score superior to a threshold are kept. NMS is applied to filter out the redundancies of boxes framing the same instance. As there are multiple anchor sizes, multiple objects can be detected in each sub-region.

have a constant reduction factor of 32. Significantly bigger images offer a better output resolution, but the inference time is then hugely increased. The parameters in the paper offer a good speed vs precision trade-off.

Despite a straightforward inference use, the training is unstable. There are 4 different terms in the loss, responsible for (i) the box size offset, (ii) the box shift, (iii) the objectness score, and (iv) the classification. Although they don't compete against each other, they all concern a different aspect of the expected output and must be weighted accordingly to one's priority. The sum squared error is used for the classification and objectness (which are probabilities), while the sum of the difference of squares is used for positional information (standard for regressed values which can be positive or negative). To be more precise, YOLO uses the square root of the anchor's width and height offset instead of their direct value. This addresses the fact that imprecision in the size of small boxes is more problematic than for big boxes. Further, the majority of bounding boxes in the output tensor do not contain an object: with a (13×13) output resolution and 5 anchor boxes, there are $13 \times 13 \times 5 = 845$ possible boxes, which is orders of magnitude more than the average number of objects in COCO, which is 7.2. Therefore, the objectness score can overfit the value 0 and be right more than 99% of the time. To circumvent this issue, the authors add a weight to the objectness score which depends on whether it is supposed to detect an object. If it does, the weight is 5, if not, it is 0.5. This heuristic reduces the problem of over-representation of negatives. Furthermore, due to the multiple pooling layers, small objects tend to be filtered out and as such less detected (this is one major

drawback of YOLO's first version). They address this using a skip connection from the (26×26) intermediate tensor to the output layer (the higher resolution tensor is downsampled to have the same dimension as the end tensor). concatenating features coming from two different resolutions improves the accuracy by a whole point on the COCO benchmark. Finally, training such detection and classification network presents a challenge addressed in an original way in [164]. The training data is mixed: it uses classification and detection datasets. When fed an image for classification, it back-propagates gradient using the classification loss only. For detection data, the entire loss is used. This helps the network learn more complex features for the two equally important tasks.

As a result, YOLO is extremely popular. Although the training requires tricks to improve the performances, at inference time, this model is extremely simple to use. Compared to slow models such as Faster R-CNN, YOLO provides a major advantage. However, its precision is significantly lower than other slow models. Although the added skip connection addresses issues the first version had with small objects detection, this issue is not resolved. The algorithm is also limited in the density of its detection as there is a fixed amount of anchors per region. For dense object detection applications, one must be careful about the number of anchors and the size of the output's sub-regions.

CHAPTER 3

DETECTION

Contents

Chapter abstract

This chapter tackles the problem of detecting swimmers in an image. General detection challenges will be considered, with different solutions each times. Our particular swimming context will be explained to motivate the final solution's design. The method will be discussed using quantitative and qualitative analyses. All of this will finally be adapted and contextualized for the definitive tool that is the aim of this thesis.

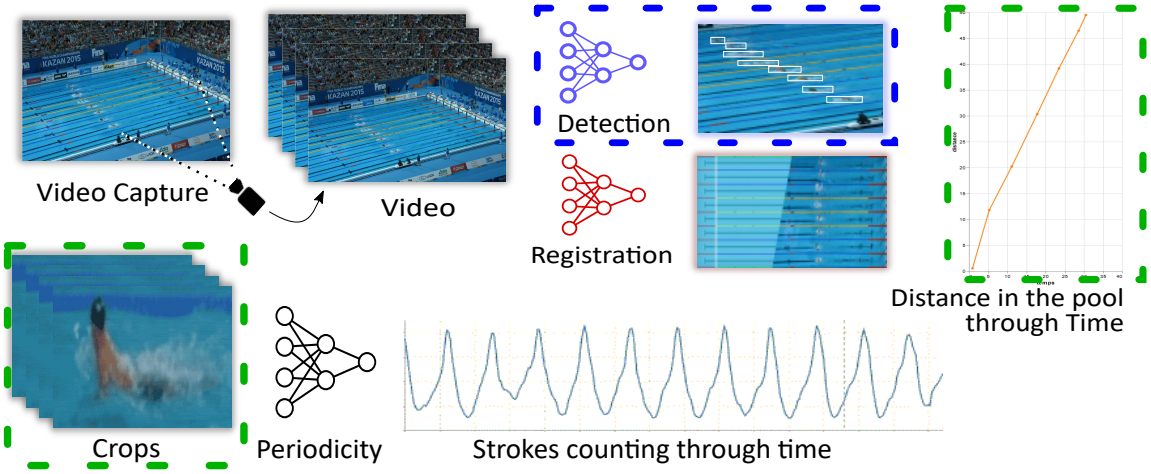


Figure 3.1 – The swimming race analysis pipeline. The detection part, framed in blue, is tackled in this chapter. The pipeline parts affected by detection are framed in green. It is arguably the most critical part of this pipeline, as both swimmers placement in the pool and periodicity analysis depend on it. Indeed, periodicity counting relies on crops around a swimmer provided by the detection model. A bad detection may result in meaningless periodicity analysis. If detection is not correctly handled, the two main outputs of this entire pipeline are deeply affected.

A dataset for swimmers detection as well as preliminary works on an annotation tool will also be presented. They enables the training and evaluation of our detection model. The finally acquired amount and nature of data will be discussed, as it shapes the overall detection model's properties. This chapter is mainly taken from the work:
Nicolas Jacquelin, Stefan Duffner, Romain Vuillemot. *Detecting Swimmers in Unconstrained Videos with Few Training Data*. Machine Learning and Data Mining for Sports Analytics, Sep 2021, Ghand, Belgium. ⟨hal-03358375⟩.

3.1 Introduction

Regarding swimmer analyses from competition videos, the main and most important aspect is to detect the swimmers in each frame. Although it is often not the terminal goal, a precise and reliable detection allows to perform further studies on swimming quality, helping the swimmers to improve their overall performance. It is especially important to be able to do it in a non-intrusive way. Indeed, placing sensors on a body can give good precision with no need of going further with Computer Vision (CV), but it is limited to specific environments. A championship forbids to use of electronic device or special colored marker which facilitates tracking. Moreover, during competitions, one could not equip

each swimmer with their own special sensor. Video analysis is more complex to initially setup as it requires specific methods, but it is the most efficient way to return the swimmers position in a set of already filmed passed races. A robust algorithm to effectively detect all swimmers of a race in a video is therefore of utmost importance for both the athletes and their coach. The position of such method in our analysis pipeline is shown in Figure 3.1.

3.1.1 Problem Formulation

Swimmers detection in videos is the process of identifying the visible parts of a swimmer's body in a picture. Detecting a swimmer in an image - and by extend in a video - may seem like a relatively easy task as state-of-the-art methods reach excellent results for human detection. However, the visible features in the environment of a pool are very different from those of daily-life walking and standing persons. A swimmer is mostly hidden under small and noisy waves, so a precise detection is much harder than for a normal human detection task. Therefore, an entirely different model has to be created to detect swimmers during competitions and training.

This task is far from trivial, due to the complexity of the environment: the pool. It is full of reflections and diffraction, affected by unpredictable waves creating many local deformations in the image. Moreover, the light on the water tends to saturate the camera sensor, or at least obfuscate the swimmers underneath. Apart from that, even recent deep learning methods [165, 80, 131] usually require a large amount of carefully labelled data which are not easily available. However, we argue that a dataset with tens of thousands of images is not accessible to everyone to train their data on due to their computational cost and time (not everyone has access to GPU servers).

This chapter shows a way to alleviate these problems without requiring thousands of labelled images, making the detection process work even with non-expert annotations. Our approach could be applicable to detect unusual objects which are not in common detection datasets or daily-life context (with unusual visual features). Our main contributions in this chapter are the following:

- a model for automatic and robust swimmers detection in competition videos outperforming state-of-the-art large-scale object detection models,
- a annotated swimmers dataset,
- a method to easily train the model which reaches high performances with few data.

3.1.2 Motivation

Detecting swimmers is an important part in automatic swimming video analysis, and it provides the basis for further, more detailed analyses. Indeed, to estimate the swimmers positions in a pool from a race video, the detection in the 2D images is obviously required. Then, after a geometric projection into the coordinate system of the pool, it is possible to know the position of the swimmers in the pool plane, *i.e.* their distance in meters to the starting blocks and their lane. This whole detection and transformation process informs the swimmers relative position and rank, as well as their speed, speed variation, lap time etc. Furthermore, the number of swimming strokes during a length is also a very important metric to measure the swimming quality. To extract this information automatically, one needs to crop a sub-region surrounding them first, which is doable with a detection model.

Apart from these objective and quantifiable metrics, subjective and qualitative observations made by coaches are extremely precious for a swimmer. To perform them efficiently, coaches often need zoomed videos around their swimmers: such videos could either be extracted manually by annotating the position on each frame, or be computed automatically. The extracted regions of interest and the automatic metrics mentioned before could greatly help the trainers to guide their swimmers. Automating this task would save a large amount of time allowing the coach to focus more on the technical aspect of swimming.

A good model is an important objective, but obtaining one with few data is rather challenging. Nowadays, there are many datasets with thousands or even millions of images of common objects [**coco**, 48], but some problems arise. First, the swimmer class is present in none of these public datasets. Although the class "person" is very well represented, it happens that from a CV perspective, they look rather different. One could simply not use it for swimmers detection. Second, not everyone can afford to train a model on such data collections since this is computationally expensive and time-consuming. This is one of the main emphases of this thesis: the feasibility, generalisation and accessibility of our method. This is especially important considering most sports coaches are not CV experts, and they could have a hard time adapting our ideas if they were too expensive. Third, although fine-tuning a robust generic model enables to get a new performing model with few data, it is not suited here. It works well if the image distribution is not too different from the daily-life context it was trained on, but is not the case for swimming. Due to the many local warping the pool environment creates, common existing models perform poorly. Regions Of Interest (ROI) detection and image embedding models are not fitted for this particular situation. Therefore, the use of small specialized well-crafted datasets gets more and more attention in many applications, especially if they allow the creation of good detection models. Finally, a detection model trained from few annotated data can then be

enhanced by training with more unlabelled data, using common semi-supervised learning techniques like self-training [31] or knowledge distillation [81]. This new, better model could itself be used to get a better precision in the aforementioned applications.

3.2 Related Work

This paper addresses the problem of swimmers detection from a CV perspective. Methods based on sensors placed on swimmers, or multiple points of view with underwater cameras will not be discussed here due to their huge difference in domain and application.

First, we need to mention the work of Benarab *et al.* [14, 13, 12], who recently proposed several techniques pursuing the same detection goal as ours. An interesting aspect of their work is the absence of deep-learning methods: their computer vision algorithms are mostly based on low-level techniques, and in particular Joint Transform Correlation (JTC). They transform the pool image and swimmer head reference images into a 2D complex spectrum and use a 2D convolution to find correlations between them. The reference images variety and choice is key: they must be representative of the swimmer head domain in order to work. They try different scales and rotation to be more exhaustive. Although it allows a faster inference and low computation needs, this choice leads to hand-crafted, pool-specific thresholds and a lack of generalisation (they always use the same 2 races as an example for their papers). In the end, they do not provide a model or metric to compare results. This is surely one of the best solutions without Deep Learning (DL), but it also illustrates the limitations of these classic techniques. It can, however, serve as inspiration for posterity.

Woinoski *et al.* [208] proposed a method based on a swimmer dataset annotated by themselves on a selection of 35 race videos, collecting about 25 000 images in the process. Sadly, it was not released before the end of our work, so we could not use it. However, creating such a data collection is a great milestone for the community once it is made public. They adapted a Yolo-V3 [165] model, where classes describe the swimmer state (normal swimming, diving, u-turning...). Their model reach an AP25 (Average Precision (AP) definition in Section 3.5) of 0.7.

Hall *et al.* [84] propose a similar work, with an even larger dataset containing 327 000 images from 249 different races. They used static videos and labelled each frame. This is tracking data which is comprehensive of time, therefore the task is different even if the objective is similar. Using a temporally dense 25 Frames per Second (FPS) swimmers head annotation, they can use information from previous frames to make a prediction on the next one. The model they used is a 2-step detection framework, similar to the R-CNN family. The first step is a rough head detection over the whole image. The second inputs a crop around this detection,

and refines it with a regression model. The tracking part of their method occurs only then, once detection has been performed. Again, they did not released their dataset or model, so the comparison is impossible.

3.3 Data

As mentioned in the related works, each method has its dataset but none was released. Therefore, to perform DL approach, we have to label images ourselves. Object detection annotation is a topic we addressed in Chapter 2 in a general sens. In this section, special properties and expectations of swimmers detection labelling will be discussed.

3.3.1 Labelling Fuzzy Objects

Contrarily to daily-life object, a swimmer does not have well defined edges. The extent of this problem depends on the swimming style, but as shown in Figure 3.2, it is rarely possible to perfectly know a swimmer's correct body boxing, either due to the waves they create or because of diffraction. Therefore, a choice must be made on what is to be annotated.

Moreover, it is difficult to be consistent: the edges are not well defined, but the annotation process must highlight the same body parts. If on some images, the bounding boxes stop where the skin is not visible, and for other images they stop where the legs actually are (even if one cannot see them), that will cause divergence during the model training.

With more time, one could label a swimmer and the waves separately, create classes depending on the degree of visibility of a swimmer, but it is not the point here. This chapter shows it is not necessary to spend too much time or money in the annotation process if the problem is though and defined enough prior to the annotation process.

3.3.2 Guided Annotation Tool

Creating an extensive dataset is time consuming, even with good assistance tool. To alleviate this problem, we worked on another tool to further help swimmers annotation. Although it did not resulted in a publication or a public release, the studies that started with this more modest project are interesting and contain valuable data. This tool relies on an already trained detection model, like the one presented in this chapter. It worked like a regular annotation tool, but bounding boxes were already placed by the model. The user could quickly adjust these boxes if they were almost correct, delete the false positive, and of course create

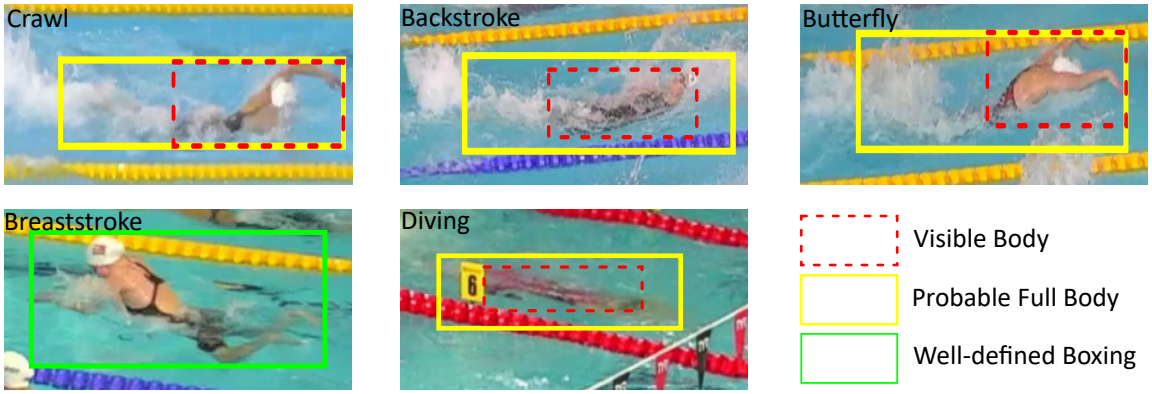


Figure 3.2 – Representative examples of the edge fuzziness problem in different styles. Although for breaststroke, the framing is generally well-defined, for other styles it is not. The swimmers create waves (especially with butterfly) keeping an observer from knowing the exact boxing of their body. Even with an unexcited water clean surface, diffraction problems warp our view and shift the visible position of a swimmer from their actual position.

new boxes. Conceiving this tool, our priority was labelling speed, as it is the most cumbersome problem of annotation.

Other that that, the key feature concerned the detection threshold of the model. Indeed, detection results changed in function of the level of zoom and other similar variation. The detection threshold (more on that in Section 3.5.2) was one key factor in this process: too low returns big bounding boxes and false positives, too high gives false negative and narrower boxes. But with distant swimmers, it is overall beneficial to have a low threshold. Therefore, while annotating a race distant from the camera, the user could set the threshold to low value in order to have better suggestions, and adapt it again for the next race. However, this tool was very incomplete: a lot of interesting features that could greatly improve the labelling speed were never implemented. For instance, the user had to fine tune the detection model with the new bounding boxes themselves, even if it would have been better to automatically do it during the labelling process: the model would get better and better with each new annotated image. Likewise, the images order is also an important factor if one wants to continuously train the model. Rules based on images variety (distance, lighting, angle...) would have been interesting to explore and create. Finally, a model could also predict the most interesting threshold depending on the input image, still enabling the user to adjust it as they wish.

This incomplete first version was nevertheless given to 3 different test subject to get a first evaluation of our tool. Quantitative results, available in Table 3.1, showed a speed increase using it, even if the small number of subject did not allow deeper quantitative analysis. Regarding the qualitative part of the study, test subjects were asked to describe their user feeling. They all though the presence

Expert	threshold 0.5	threshold 0.9	threshold 0.2	No Model
1	47s	1min 04	1min 08	1min 50
2	26s	46s	39s	1min 02
3	34s	1min 05	55s	1min 30
Average	35s	58s	54s	1min 37

Table 3.1 – Average annotation time per image depending on the candidate and the experience.

of boxes increased their performances and felt frustrated when a swimmer was not detected, proving the engaging interest of suggestions proposed by the tool. When asked, they said they preferred an imprecise box to no box at all. Although it was promising, this work was not pursued during the thesis because it was too far away from my personal skills, interests, and objectives. Other functioning labelling tools exist, and even if they were not adapted for swimming (because they too rely on detection model, but they never trained on swimmers), they were already more advanced and expertly made than mine.

3.3.3 Dataset Creation

Regarding the actual dataset creation, we selected 12 international level competitions with openly available race videos. Each had a different camera position relatively to the pool, which gave the dataset a great range of angle and size variation. One frame was saved every 3 seconds, resulting in 403 different images with a maximum of ten swimmers, where at least one of them is visible. In total, 3121 bounding boxes were created (7.7 per image in average). Some competition were inside, other outside, with variate lighting conditions. For each competition, races were selected to represent the 4 main swimming styles and the 2 genders (51% ♂, 49% ♀). The crawl style is a bit more represented (30%) and the butterfly is a bit less present (18%). Breaststroke and backstroke are equally represented (respectively 27 and 25%). The data from 3 pools out of the 12 was used as test data and the 9 remaining as training data. The resulting dataset, which we called $Swimm^{400}$, was composed of 80 test images and 323 train images.

The swimmers bounding box annotation was made with the open-access tool LabelImg [44] (our guided annotation tool was made after). Concerning the fuzzy edge problem, the decision was to frame the swimmers' visible parts only so that the boxes are as small as possible around them. Only one person annotated the whole dataset, as it resulted in more uniform results.

3.4 Proposed Approach

Designing our method, our main constraint was the small amount of data to train from scratch a detection model. This did not allow us to use classic methods such as YOLO [163] or Faster-RCNN [80]. Instead, we drew our inspiration from another domain with usually few data samples: medical imaging. In this domain, as each label has to be made by a specialist taking a lot of time, and as the images come from costly medical imaging processes, data is always missing. Therefore models have been developed to address the lack of data. They are designed to be deep enough to learn complex feature hierarchies and patterns of high variability, but not too much to overfit on the small dataset. Moreover, they have interesting spatial properties this section will explain. The inference time was also important according to the experts we worked with, because (i) a swimmer wants the analyse of their race just as it finished and (ii) a coach needs analyses of a swimmer during the training, not after. This discards Faster-RCNN [80] and similar methods with multiple steps and sub-image inference. This guided us to single-shot methods, which are usually faster. Our final solution gives excellent results and is usable in many different environments (inside/outside, pool/free water) and for a large variety of acquisition conditions (see Section 3.5). It also can easily be applied to other swimming-based sports like water polo (see Figure 3.8).

3.4.1 Detection Model

The segmentation and detection tasks have different objectives, as explained in Chapter 2, but it is not their only difference. Their training data is also distinct by nature. However, it is possible to use heuristics to convert one data type to the other.

3.4.1.1 Bounding box regression vs. segmentation

Bounding box regression-based approaches [165, 80, 131] all rely on the same principle. They transform a part of the image into a semantic vector, from which the model computes the probability of presence and position of the objects on the image. This is a very complex task. Learning this transformation and providing stable results requires large amounts of labelled training images.

On the other hand, fully-convolutional models like Unet [172] transform each pixel into a "1" or "0" response according to the objective. This relatively simpler task brings two main advantages. First, it requires fewer data because the overall task is not regression (of the bounding boxes), but a binary classification (*i.e.* thresholding) extended to the whole image. Second, the conversion of a pixel into a presence probability amounts to segmentation, which is a task of higher level than just a bounding box regression. Indeed, according to the object position

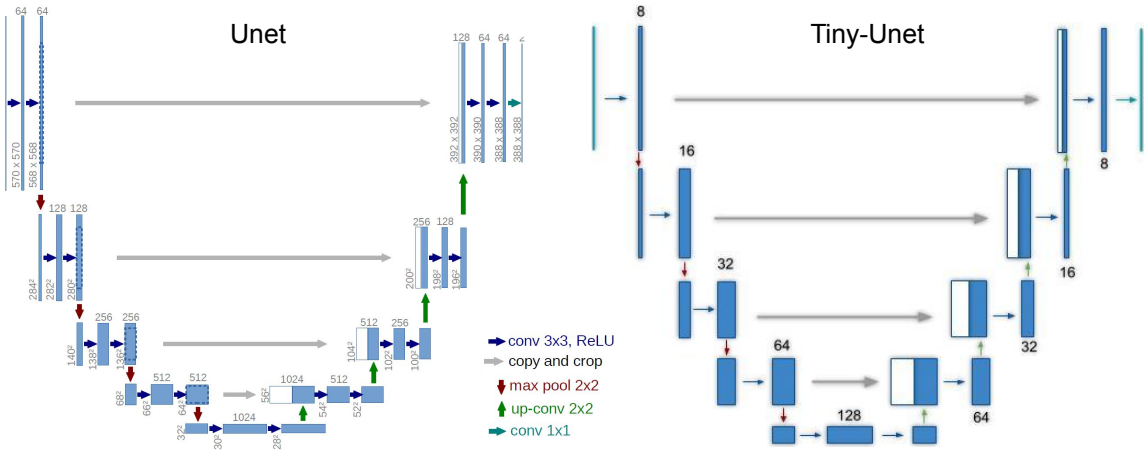


Figure 3.3 – Comparison of the classic UNET architecture and our tiny-UNET. The latter is much more compact, each level having both less convolution layers and less filters per convolution. It is thus significantly faster ($\tilde{5}\times$). Despite this complexity reduction, Tiny-UNET is still complex enough to isolate swimmers.

and orientation, much space contained inside a bounding box can be background, but most of a segmentation area designates the searched instance. Therefore, a segmentation model provides an alternative, more precise description of the regions of interest, as it has the possibility to exclude the parts of the surrounding background.

3.4.1.2 Tiny-Unet

We propose a variant of the well-known Unet architecture [172] for our swimmer detection model. The original model is a residual autoencoder with blocs of 3 convolutions layers with the same number of filters before a downsampling (in the encoder) or upsampling (in the decoder). The following modifications have been performed: instead of 3 convolution layers between each down/up-sampling, only one is performed. The filters are also smaller, increasing from 8 up to 128 instead of 64 to 1024 for the original Unet. A side-by-side scheme comparison of both is given in Figure 3.3 We will refer to our model as *tiny-Unet*. Due to its shallow architecture and low filters number, it is able to run at 260 FPS on a GTX 1080 NVIDIA GPU with (256×256) pixels images, where Unet runs at 50 FPS in the same conditions. Tiny-Unet is therefore faster than a real-time detector for 25, 50 or 60 FPS videos, which are standards in the camera industry.

To convert the model's output heatmap into bounding boxes, a threshold is applied to said heatmap, and the remaining areas are extracted. A bounding box is created by finding the circumscribed rectangle around each of them. This further allows for a fair comparison with the benchmark methods in Section 3.5, each of them creating bounding boxes.

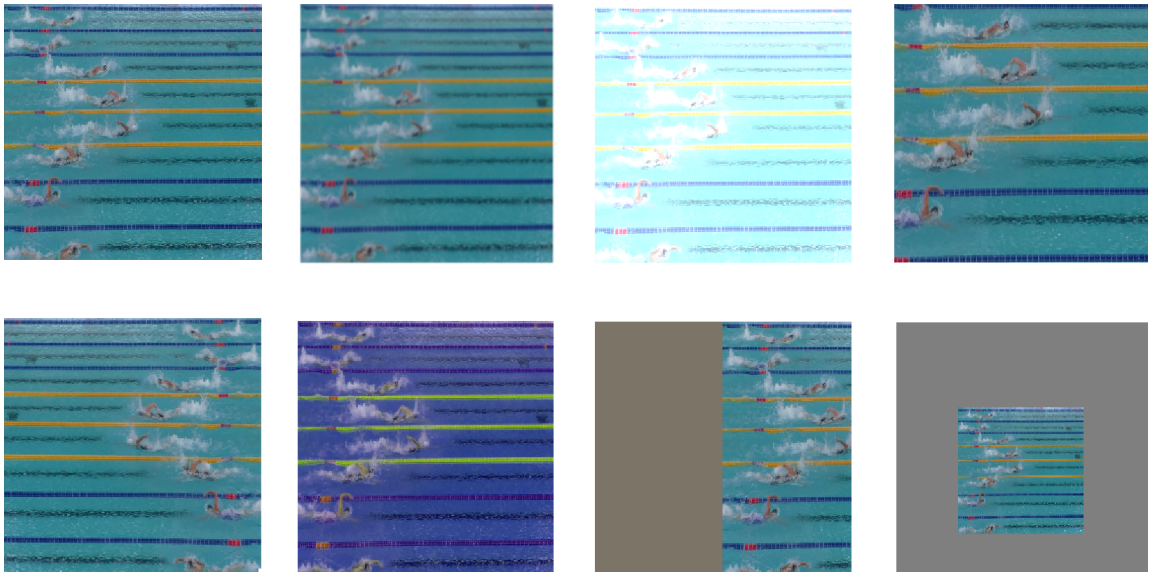


Figure 3.4 – The data augmentations used for the model training. From left to right, top to bottom: original image, blur, contrast and brightness change, crop, horizontal flip, hue change, side switch, zoom out.

Box-to-Segmentation-Map Transformation: the Unet model requires segmentation heatmaps for training. In order to convert the bounding boxes from $Swimm^{400}$into segmented data, an image with black background is created, and "filled" with white pixels inside the labelled boxes. Therefore, a pixel is 1 or 0 depending on whether there is or not a swimmer at the pixel. Multiple variants of this approach have been tested. Whiten inside the full box or only in the inscribed ellipsis; using smooth edges or hard edges. As shown in Table 3.2, the option giving the best result was to use the inscribed ellipsis with hard edges. As the boxes are reduced to approximate masks, we noticed that the model can be successfully trained even with mediocre and partly inconsistent annotation. This allows for a much quicker and less costly annotation process.

3.4.2 Data Augmentation

Training Tiny-Unet is direct as it is a simple forward model without hyperparameters. To allow batch training, the images were all resized to (256, 256) pixels. Data augmentation is used in order to increase the trained models performance. It will also compensate the low quantity of images in $Swimm^{400}$.

Zoom-in / zoom-out: the main augmentation was the zoom in and out (Fig 3.4, right column). Image crops are performed during training, such that the subjects occupy more space. Inversely, a neutral colour could be put around the reduced image, so that the swimmers look smaller. This helped the model to generalize its representation of swimmers independently from their size. $Swimm^{400}$originally

contains swimmers at different distances from the camera, but this data augmentation increased this benefit even more.

Side-switch: another important augmentation was the side-switch, seen in Figure 3.4, bottom row, third column. This transform is extremely useful to avoid central overfitting: most images present in $Swimm^{400}$tend to center the swimmer. The side-switch puts them on the side, preventing the model to only detect instances at the center. For fully-Convolutional Network this is less a problem, as they are mostly translation invariant.

Others: apart from these two transforms, other more common methods where used to train the model. The random left-right flip generalizes the swimmer direction to the model, by giving them the same chance to face each side. The colour change (in HSV format, the hue is rotated by max. 45° so that the water can have any blue shade plus some green ones) generalizes to many skin, pool and water colors. The contrast and brightness random variations adapts the model to the many lighting conditions that can happen during a different competitions. Finally, Gaussian blur increases the overall robustness.

Of course, all these augmentations do not require any further annotation, as they are automatically generated during the training. The probability to trigger them is 50% each, except for color variation (30%) and side-switch (10%), as they both are stronger changes and thus might make the model diverge if used too much. These trigger probabilities work well for our study case, but may need to be slightly varied to adapt them to other detection problems.

3.5 Experimental Results

This section shows how we found the best parameters to train our model. We first compare different variations of our method to find the optimal solution with our $Swimm^{400}$dataset, then we compare it to another existing method.

3.5.1 Metrics

The comparison will be made using the AP and Average Recall (AR) 25, defined as:

$$AP\ 25 = \frac{1}{N}\sum_i \frac{True\ Positives_i}{Positives_i}\ , \quad AR\ 25 = \frac{1}{N}\sum_i \frac{True\ Positives_i}{Relevant_i}\ , \tag{3.1}$$

on image i, $True\ Positives_i$ being the detected bounding boxes with an Intersection Over Union (IOU) of more than 25% with the true box, $Positives_i$ being the total number of boxes detected, and $Relevant_i$ being the true number of boxes from the annotation (*i.e.* the true positives and false negatives). N is the number of images in the set.

To get a better idea of what these metrics represent, consider the different detection scenarios presented in Figure 3.5. A perfect model would return the results from image A, with the correct number of boxes well enough placed, and no other boxes. Such a model would have an AP and AR equal to 1. Note that, as we use the AP/AR 25 metric, the boxes can be imperfectly fitted around the swimmer as long as their IOU with the annotated ground truth is superior to 0.25. On image B, all the objects are correctly detected, thus giving an AR of 1, but many background objects are found as well. As a result, the AP is greatly decreased. On image C, there is no false positive, resulting in an AP of 1. However, only 1 object among 3 is detected, which gives a low AR. Finally, image D shows a catastrophic model, detecting several background objects but none of the swimmers, therefore both metrics are null. Using these examples, one can conclude that both metrics have their own interest, depending on the conditions. A good AP means there are not too many false positives, and a good AR means most of the objects are detected. For our task, the AP is more suited for this task and will be prioritized, compared to the AR. Indeed, as in this work there is no discrimination process to be sure that a detection corresponds to a swimmer, we want to reduce as much as possible the false positives. Therefore, both metrics will be used, but the AP will be considered the most important.

Although both metrics used to be massively used, they are not detection standards anymore: it is the mean Average Precision (mAP) and mean Average Recall (mAR) [40], defined as follows:

$$mAP(dataset) = \frac{1}{10} \sum_{X=0.5:0.05}^{0.95} AP\ X(dataset)\ , \tag{3.2}$$

, which in summary corresponds to the mean of all AP/AR with IOUs between 0.5 and .95 (with a step of 0.05). There is a similar formula for the mAR. This metric aims pixel-perfect precision, which is not adapted for swimmers with blurry edges under the water. AP25 and AR25, on the other hand, are closer to what real-world applications seek. Indeed, the precise delimitation of the swimmers' contour is not the priority here. The main objective of our applications is to obtain the overall position of the swimmers, for example, to extract a sub-region around them. Therefore, estimating the boxes' barycenter and general size is enough, so the AP/AR 25 are more appropriate.

3.5.2 Ablation Study

First, we trained our tiny-Unet model on different variants of the box-to-segmentation map strategy. The results are shown in Table 3.2.

This table clearly shows the superiority of shapes with hard edges. Smoothed ones tend to reduce the model convergence during training. Finally, filling an

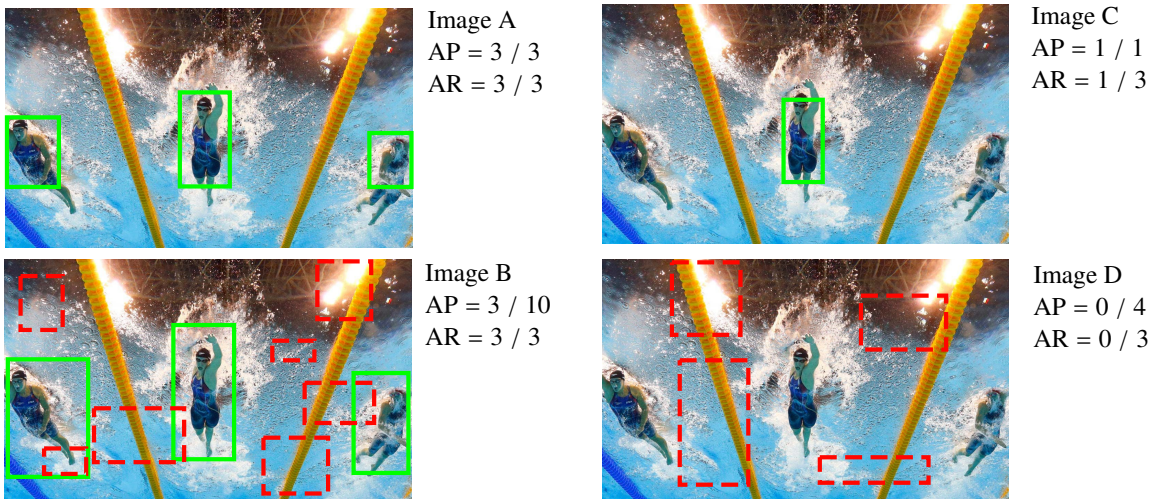


Figure 3.5 – Different detection scenarios on the same image. In green and continuous: boxes with more than 25% IOU with a true box (*i.e.* true positives). In red and dashed lines: incorrect detections (*i.e.*: false positives).

Table 3.2 – Detection performance of our model trained with different representations of the target heatmaps. The input images size is (256, 256) pixels. * 72 in the original paper, improved since.

Training Data	AP 25	AR 25
Ellipse Hard Edges	**76***	**60**
Rectangle Hard Edges	60	28
Ellipse Smooth Edges	21	5
Rectangle Smooth Edges	13	3

ellipse shape is better than filling the whole rectangle bounding box. An intuitive explanation could be that corner regions are less likely to contain pixels from the instance. In fact, the ellipse mask contains almost only the swimmer, and the remaining pixels can be understood by the model as regularisation. As the edges of a swimmer are fuzzy anyway, it probably does not differ much from a precisely-labelled pixel-perfect mask.

To compare the results of tiny-Unet with current state-of-the-art methods, we trained two variants of Yolo on $Swimm^{400}$. The first version is YoloV3 [165], a deep model with a 2048 fully-connected layer after the convolutions. The second is Yolo-tiny, which is shallower. Moreover, it replaces the fully connected layer with a 1×1 convolution layer with 56 filters, in order to drastically reduce the number of parameters to train. The 3 models are trained with $Swimm^{400}$dataset. The Adam optimizer is selected and starts with a learning rate of 10^{-3} with a decrease of 0.1 if the test loss plateaus more than 10 epochs. As the dataset is quite small, a batch size of 16 is chosen, and the loss is the Mean-Squared Error (MSE) in each case. For the Yolo-based models, the λ confidence training trick described in [163] Section 2.2 is followed. From Table 3.4, it appears that our tiny-Unet model outperforms by far the two others. Indeed, Yolo is a great model as long as enough data is available because of its conversion from feature vectors to output tensors. On the other hand, tiny-Unet does not require such a transformation. This result is confirmed by testing the 3 models on the same videos : the tiny-Unet gives the best results. Moreover, with a video analysis, the tiny-Unet is much more stable between one frame and the next one.

Using this model, we explored further different domains to get a better knowledge of its abilities and limits. A first experiment we did was to measure its speed. First, we compared it in function of the batch size, which enables to treat several images at the same time using our GPU. It appeared that for any batch superior to 4, the speed did not change significantly. The inference speed is significantly more impacted by the image size, also affecting the performance. Table 3.3 displays this trade-off. It shows a peak in the AP at (256, 256) pixels, and in the AR at (512, 512) pixels. Indeed, despite extensive size-focused data augmentation, smaller images do not have enough pixels per swimmers to get a good detection. On the other hand, to big images tend to contain many false positives, even if few swimmers are missed. The (256, 256) pixels inputs are chosen for further experiments as they offer the best AP vs AR trade-off, and excellent inference speed.

We also studied the impact of the threshold on the performances, which is often underestimated. Indeed, if one observes a very thin performance peak around one optimal threshold, it does not mean the model is optimized for the *task*, but mostly for the *test dataset*. Such a peak is a bad thing for generalization, especially with small datasets such as $Swimm^{400}$. On Table 3.3 (rightmost part), we observe that our optimum is fairly flat between 03 and 0.6, which is an good point. The optimum value is 0.45.

Table 3.3 – Speed and performances results in function of the input size (left), and AP25/AR25 in function of the heatmap threshold (right). We tested the speed with the same set of 1700 images. The different AP25/AR25 results were evaluated on the $Swimm^{400}$test set.

Side	FPS	AP25/AR25	0.30	0.45	0.60	0.75	0.90
128	**490**	46/13	69/59	**76/60**	69/54	65/50	64/50
256	260	**76**/60					
512	80	55/**67**					
1024	20	28/63					

Table 3.4 – Performance comparison for the different detection models. They are all trained with the same data, except for the first line. In bold, the best of a category. We observe that the original UNET architecture gives significantly worst results compared to tiny-Unet, as it overfits on the few data.

Model	AP 25	AP 50
Yolo (from [208])	70	-
Yolo	24	12
Yolo-tiny	31	20
Unet	39	25
Tiny-Unet	**76**	**60**

3.5.3 Comparative Results

Also shown in Table 3.4, the Yolo model trained on 25,000 images by Woinoski *et al.* [208] gives results comparable to the tiny-Unet trained on $Swimm^{400}$. It is not measured on the same benchmark though, thus we cannot assure which model exactly is superior. However, ours seems comparable to theirs with only a fraction of their amount of data and model size. Our model also performs some segmentation. Depending on the intended task, both the segmentation heatmap and the bounding boxes can be used, which is another advantage compared to the Yolo-based models.

Finally, Tiny-Unet is extremely efficient in term of scalability. Being designed to be trained with small datasets, it is quite shallow and can run extremely fast (see Figure 3.3) with not-so-recent GPUs (GTX 1080 NVIDIA GPU). The experts we interrogate can determine the position of a swimmer in real time, but only one swimmer at a time. With our detection model, we know the position of them all swimmers faster than real time, so faster than an expert by a huge margin.

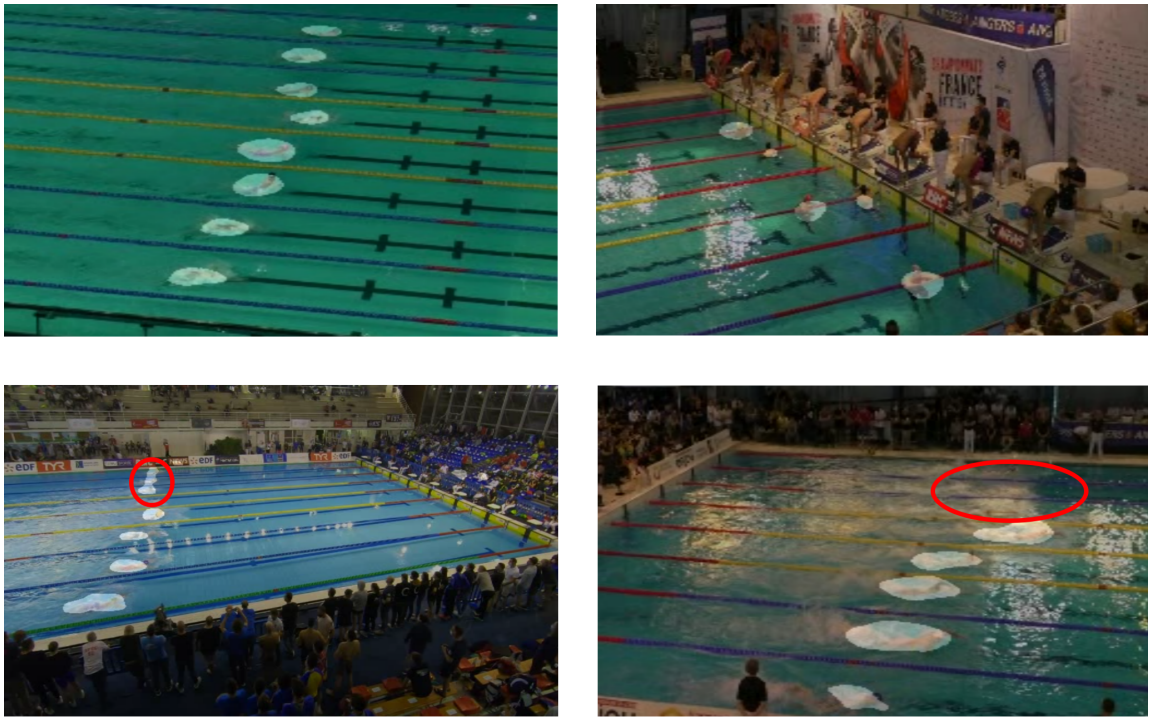


Figure 3.6 – The thresholded segmentation output overlaid on the input image with a size of (256, 256) pixels. Circled in red are mistakes to focus on.

3.6 Visual and Qualitative results

In order to get a more complete analysis, we also provide a few visual examples of results. These are not cherry-picked: they have been selected because they are representative of a global behavior of our model.

3.6.1 Swimming Races

First, it is important to study the model in its normal context. Figure 3.6 displays a few of them, illustrating different behaviors of the model. The top-left image displays the segmentation on a classic swimming segment. The overall detection quality is very good, each swimmer is well detected and separated from the others. Although one could not really say it is precise segmentation, the bounding boxes we can infer from the segmentation heatmap are of great quality. The top-right image show both one limit and one unexpected performance. The limit, obviously, is the lack of detection of swimmers about to dive, even if some are present in the training dataset. This is not really problematic, though, considering the important swimming phases to detect are inside of the pool, where detection works correctly. The unexpected result this image shows is the accurate detection of peoples in the water who are not swimming. In our dataset, there is no occurrence of such

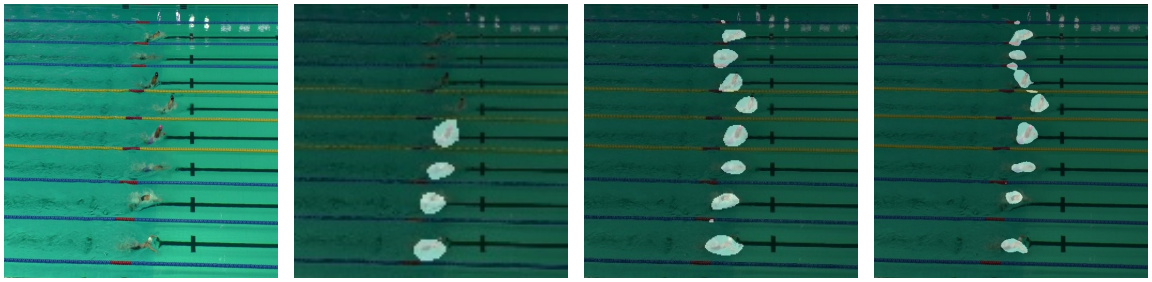


Figure 3.7 – Classic use-case image overlapping with the segmentation heatmap outputted by the model with different input sizes. From left to right, the input sides are 128 pixels, 256 pixels and 512 pixels.

"objects", so the network has apparently learnt to generalize enough to detect them. This is a great thing as this means the model is not strictly limited to professional swimmers in pools (see the water-polo and lake examples below). The bottom-left image highlights an important problem: segmented blobs which are too small and too close tend to merge. This is especially common when swimmers are far from a low camera. Individually, each "sub-blob" is correct, but they should be separated. This is addressed by the pipeline, where each swimming lane is isolated from the others, splitting the merged blobs. One could also segment the swimming lines and mask them out of the heatmap to divide the merged blobs. Finally, the bottom-left image shows the other side of the problem with swimmers too far from a too low camera: the lack of detection. Usually this is caused by a threshold too high, but lowering it would also create false positives. Again, this will directly addressed in Chapter 6: as the heatmap of each swimming lane is isolated, if no detection is found, one could reduce the detection threshold until something is finally detected. However, this is not optimal and it would be preferable to manage these results during training.

Table 3.3 displayed the importance of the input size for the performances. Further, one can see that the AP and AR optimum do not occur for the same size. Figures 3.7, 3.8 and 3.9 all highlight this phenomenon on their own domain. The general observation is that as the input grows, the swimmers segmentation improves. In Figure 3.7, for instance, the blobs shrink as the size increases, but they contain less water and a bigger part of the swimmer. From detection, the model almost achieves segmentation. This is especially interesting considering the original data: ellipses inscribed inside bounding boxes. This proves that this shape was a good heuristic, nicely highlighting the athlete. In this case, we can consider a case of weakly supervised learning, with an initial data less complete than the result one. One could arguably unravel this thread to generate great swimmers segmentation with only bounding box annotation. Increasing the size, though, does come with compromises. Indeed, as the AR increases, the AP reduces, due to many false positives appearing between the swimmers. Red markers on swimming buoys, especially, tend to be confused with humans and are sometimes

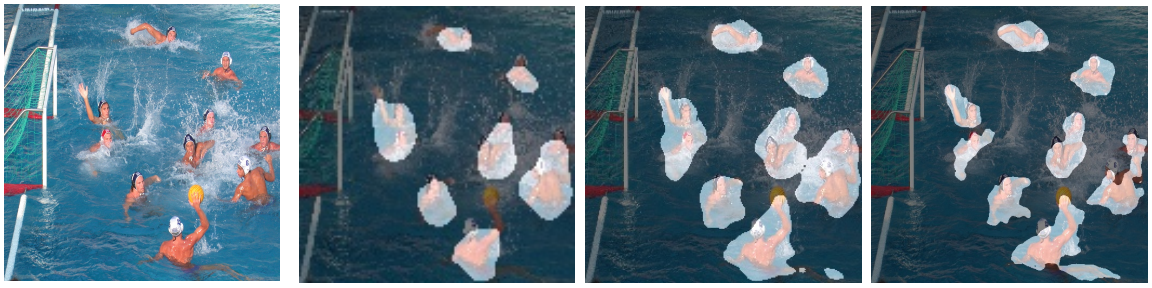


Figure 3.8 – Model Segmentation of a water-polo image, slightly out of the training domain. From left to right, the input sides are 128 pixels, 256 pixels and 512 pixels.

detected when they occupy a big enough part of the image. The opposite scenario arises with smaller images: few no false detection, but many swimmers missed (half, here). Depending on the exact use-case, one could prefer one extreme or the other. The intermediate size, (256, 256) pixels, seems to be a good compromise for general purpose swimmers detection.

3.6.2 Other Swimming-Based Activities

Our model performs great swimmers detection, but it can also be used in slightly out of domain contexts, such as water-polo players detection. Figure 3.8 shows the results on such an image, again with increasing input sizes. In all cases, the detection is generally good, at least of the same level as with swimmers. Note that this image is a bit zoomed-in and from a high enough point of view, which is one of the best video capture situations, for detection.

In this case, the model performs well despite the players generally not in a swimming position. They are more vertical and have a big part of their torso out of the water. This is very encouraging regarding the generalisation power of our model. Moreover, the segmentation is here even more precise than with splashing swimmers during a race. The (512, 512) pixels image, in particular, shows great segmentation of the player with the ball (at the bottom), with the whole arm segmented. Underwater limbs are detected for several players.

One step can be considered missing, however, which is the swimmers blobs disentanglement. Indeed, contrarily to classic swimming races with separated lanes, here the players can get very close to each other, which causes detection troubles. Someone focusing on water-polo could think of a solution or a heuristic to alleviate this problem, but as this thesis is mainly on swimming races, we did not elaborated further.

Finally, 3.9 shows a result example completely out of the training domain, as the persons are not swimming and not in a pool. This background and environment is completely new and different from what the model was trained on. Although the results are generally less precise here, both in low and high resolution, interesting

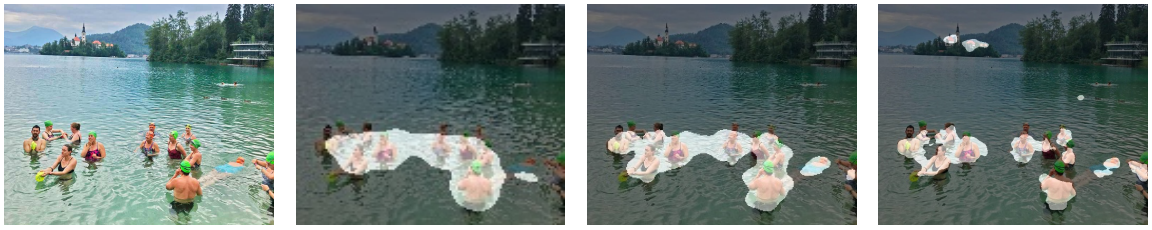


Figure 3.9 – Segmentation results of persons in a lake, which is completely out of the training domain. From left to right, the input sides are 128 pixels, 256 pixels and 512 pixels.

comments can be made on them. With low resolution, first, the group of close-enough persons is essentially well detected, with imprecise edges and a lot of background water (*i.e.* false positives) segmentation. With higher resolution, though, results are more refined, the different blobs follow decently the peoples shape. Further, even the farthest group gets detected. This means such a model could be used for swimmers monitoring on public beaches, if we authorize a high recall. Indeed, the background town is a bit detected too, there are some false positives. However, for such monitoring tools it is always better to have false detection leading to waste of time (or simply visual checking) than false negative, *i.e.* we do not detect a sinking person.

These qualitative results showed different aspects of the model which were not expected from the AP and AR studies. The first one is the better segmentation precision when size increases. This can be very interesting for close-up analyses and swimming posture extraction - which is not yet resolved. The second point is the great generalisation performed by the model, which is able to detect peoples in the water in general, as opposed to simply swimmers in a race. The applications of such a model are beyond race analyses, and could be used to save lives.

3.7 Discussions and Perspectives

The detection method described in this chapter is functional, but a few things can still be improved or modified to increase the overall performance. Also, despite having been proposed for swimming analyses, it can be generalized to other sports with low cost and small annotation time.

3.7.1 Improvements and Future Works

The swimmers coordinates output by the framework could either be read as a segmentation area (the raw heatmap) or a bounding box (after the heatmap processing). The model is currently able to detect the general position of a swimmer, but it does not detect any body part in particular. This results in an

accurate but not precise position. The barycenter of a blob is always somewhere on the athlete, but one could not predict exactly which. If the barycenter changes too much, from the shoulders to the hips, for instance, the local speed cannot be precisely measured. Depending on the intended application, it can cause issues, for instance to measure a swimmer's inter-cycle speed, which is measured during just a second. To alleviate this problem, one could create another small dataset with only the swimmers head annotated. By training a model to detect the head only, we will obtain a higher robustness. This might be incorporated into a 2-stage detector similar to Faster-RCNN [80]: the first stage would be the raw detector described in this paper, the second a head detection on a crop around the extracted swimmer position. Having this second stage will also potentially remove false positives. However, it would also reduce inference speed by an important amount due to image crops resising and a new model inference. To implement a faster version, one could have a Unet-like model with 2 decoder branches, following ProstAttention-Net ([55]). The first one would be the one described in this chapter, the resulting heatmap serving as an attention map for the other, detecting the head.

One recurrent problem, as seen in Figure 3.6, is the different blobs merge. Model agnostic ideas have been suggested to solve the problem, but it would be better to address it directly in the model. One simple idea, that is yet to be tested, would be the addition of blob barycenters to the heatmap. The general idea would be to have the model generate a second heatmap which only highlights a small region of constant size corresponding to the blobs barycenters. This region would follow a gaussian intensity distribution and one would only detect the local maxima of sufficient intensity. Therefore, even if two of these barycenter blobs are close enough to merge (unlikely as they would be very small), as only their local maximum would be detected, one would distinguish them.

As seen in the previous section, the model also has troubles detecting the smallest swimmers (*i.e.* the farther from the camera). Extensive scale-oriented data augmentation could be used to reduce a bit this problem, but it is also deeply part of Tiny-Unet's nature. The network is shallow and simple by design, which inevitably creates this limit. However, this complexity VS speed trade-off will be addressed in Chapter 6 with all the required CV elements established.

Overall, the performances could be improved with the creation of a bigger dataset (and then the use of a bigger model), but the main point of this chapter is to prove that powerful specific analysis tasks can be achieved at low cost without large computational and human resources.

3.7.2 Generalization to Other Sports

The detection process is quite general and easy to handle. For sports with atypical objects that are not present in usual datasets, our annotation and detection

process can be reused and adapted. Indeed, the low quantity of data is enough for most detection tasks if the background does not change too much (a pool, a sport field etc.) as the model will more easily understand what actually matters. Data augmentation has to be adapted for each case, though, but knowing which one is pertinent is trivial for an expert.

Further, as we explained, this model can directly be applied to water-polo, despite the absence of buoy lanes. Small fine-tuning could obviously improve the precision, as swimmers can be in very different settings than for typical races (grouped, under one-another, chest out of the water...).

3.8 Conclusion

This article proposes a framework for video analysis of swimming races for any number of athletes in the water. Its main advantage is its simplicity and feasibility: anyone can use it and recreate it for other sports, as it does not require a lot of data or big pre-trained neural networks. Peculiar objects (balls, weights, javelins ...) can be quickly detected with a small dataset, following our methods. Moreover, other swimming-based competitions (water polo for instance) can directly benefit from the present detection model.

In the end, the detection is done in a fully automated fashion, liberating to coaches the time they previously needed to do all this. It allows them to focus only on the athletic part of their role in a swimmer formation. We hope it helps them increase their swimmers performances.

CHAPTER 4

REGISTRATION

Contents

Chapter abstract

This chapter presents the problem of field registration, also named camera calibration. This task aims at projecting a given image taken with unknown camera position and orientation parameters to a known 3D coordinate system in order to obtain higher-level information like the position and speed of players. Indeed, the simple detection of swimmers in a pool is not sufficient to get these data, as their position on the image depends on camera movements and placement too. Existing methods usually first create a rough projection estimation and then use an iterative refinement algorithm to iteratively get closer to the desired calibration. These different methods will be discussed, highlighting their strengths and weaknesses. They are usually only compared in terms of precision on a standard benchmark without considering other

metrics. However, execution speed is also important, mainly in the context of live broadcast TV and sports analysis. A new automatic field registration method is introduced in this chapter, achieving excellent performance on the WorldCup Soccer benchmark, while neither depending on specific visible landmarks nor any refinement, resulting in a very high execution speed. Finally, to complement the usual soccer benchmark, we introduce a new Swimming Pool registration benchmark which is more challenging for the task at hand. This chapter is mainly based upon this contribution:
Nicolas Jacquelin, Stefan Duffner, Romain Vuillemot. *Efficient One-Shot Sports Field Image Registration with Arbitrary Keypoint Segmentation*. IEEE International Conference on Image Processing, Oct 2022, Bordeaux, France. ⟨hal-03738153⟩..

4.1 Introduction

Field registration designates the common method to align the visible field in a frame to a known coordinate system. As sports fields are planar, the registration is performed using a linear projection called homography. It can be computed by mapping 4 points from the original image to 4 positions on the template. The homography matrix is computed using the DLT [86], and enables to convert any position from the initial image into a position in the point of view reference.

The end goal of this task is to convert the position of elements in an image into their position in the field. In the context of sports analysis, registration is the process of transforming human data (videos, swimmers' positions in pixels...) into computer data (data tables, placement of a point in a rectangle...). A detection algorithm as the one described in the previous chapter is only halfway through placing swimmers in the field, as no camera information is known. The video capture system may be moving or not, and the part of the field being framed is not trivially obtained. Therefore, measuring a player's speed or position using solely a detection method is meaningless. The video must be calibrated to match a known point of view before inferring any positional information. The target viewpoint is usually a top-view, as it removes perspective warping problems, thus giving a better understanding of the scene. The use of registration as a part of the automatic analysis pipeline is shown in figure 4.1.

Manual calibration is long (at least a dozen of second per frame, with training), thus costly because most video streams come from moving cameras and would require a frame-by-frame annotation. Although it is theoretically possible to do so, in practice it takes significantly too long, thus an efficient solution for automatic field registration is crucial.

Automatic methods [184, 32, 183, 146, 106] tend to decompose the task into a two-stage process: first getting an initial rough projection, then several refinement

steps to get a more precise result. Both steps are necessary to perform well. The refinement process is similar in many aspects to gradient descent, but the search space is far from being concave. Without a good initial estimation, no refinement step is able to find a good solution. However, although this rough initialisation gives a general idea of the registration solution, it is always improvable, hence the refinement steps. This second stage takes much longer, 96% of the total processing time according to [146]. This chapter proposes an automatic field registration method which does not need this costly refinement step to give good results. Our results are already good (see Results in Section 4.5.2), thus the refinement step is not necessary if one wants exploitable results. A model segments the input image into a map that highlights a specific (grid-like) pattern corresponding to points on the 3D field plane (see Figure 5.2, template). Our approach can be applied to any type of 2D sports field with TV streams or side stadium view. While maintaining excellent precision on the WordCup Soccer benchmark [95], it achieves an inference speed of around 50 Frames per Second (FPS) on rather modest hardware (see Figure 4.3). This is important as it is critical to calibrate a field in real time, e.g. 1) for live-TV visualisation tools, or 2) for athletes to get quick feedbacks on their performance during training, and 3) cameras positions and fields characteristics change during shots and across competitions.

WordCup Soccer benchmark [95] is the only public dataset that has been widely used in the literature, although some private datasets have been introduced for registration [183, 146]. However, a soccer field is relatively simple in appearance as it contains a bi-axial symmetry with many unique visual local patterns. Thus we introduce a more challenging benchmark for Olympic swimming pool registration. Indeed, a swimming pool contains many repetitive patterns at different places in the pool (see Figure 4.2) leading to ambiguities in the image and making the registration difficult. We hope this will push forward the research on generic and robust sports field registration methods.

In summary, our contributions presented in this chapter are:

- a new benchmark for swimming pool registration with new spatial and textural challenges,
- a new efficient sports field registration method that can be applied to any type of sports and reaches high execution speed and state-of-the-art precision.

4.2 Related Work

Registration has been studied in domains as varied as medical imaging and sports video analysis. It is used to associate two different views of a similar context so that a point on one picture corresponds to the same point on the other. Creating panorama images is a common application of registration: one takes several images, each time rotating slightly their camera, and then a registration

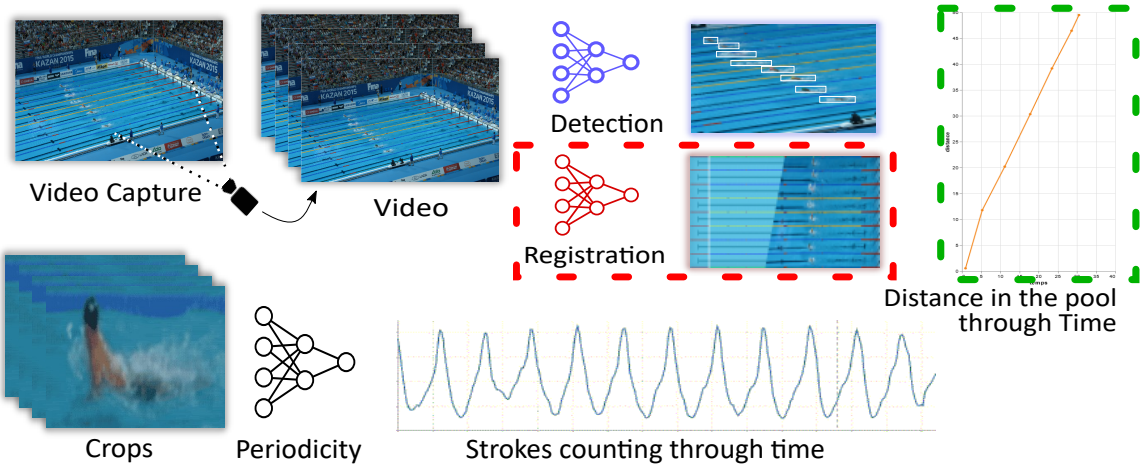


Figure 4.1 – The swimming race analysis pipeline. The registration part, framed in red, is tackled in this chapter. It does not depend on any other method as it directly inputs the raw video. Used with the swimmers detection (in the image) algorithm, it gives the swimmers position in the pool (framed in green) and all the data coming from it (speed, acceleration, etc.).

algorithm spatially associates each neighbouring picture to create a unique seemingly continuous wide image. In medical imaging, registration can be used to rotate and translate different images of one patient to compare their organ over time. This comparison is made more simple for the doctor if the images can be overlayed, which is done using registration.

4.2.1 Pair of Points and Consensus Algorithm

In the context of registration, a pair of point is defined by a point on the source image (X) and a point in the absolute 3D coordinate system (Y), linked by the equation: $X \times H = Y$ where H is a homography matrix. Combining 4 unaligned pairs (*i.e.*: forming a quadrilateral on both the source image and the other coordinate system) allows computing the homography matrix using the DLT [86]. However, automatic methods based on pairs identification can mismatch them (the elements of the pair do not correspond), which result in a completely false homography matrix. To alleviate that, most methods presented here identify more than 4 pairs, and use a consensus algorithm to determine the most likely output.

In particular, RANSAC [68] is used. In the case of homography matrix generation with more than 4 points, its general behaviour is as follows. It randomly selects 4 unaligned pairs in the whole pairs set and computes the corresponding homography matrix $\tilde{H}$. All the other pairs are used to compute a loss function measuring the distance between the source point projected ($\tilde{Y} = X \times \tilde{H}$) and the other point ($Y$). The average loss is associated to this matrix. If it is bigger

than a given threshold, the matrix is rejected and 4 other points are selected to do the same thing again. If not, the matrix serves as a basis for the rest. A new (randomly chosen) element is added to the initial set of pairs, and DLT is used again, this time with 5 elements, to compute the matrix. The result is again evaluated on the remaining other points. If the average distance is smaller than previously, the refined matrix is kept. If not, RANSAC keeps the previous matrix. A new randomly chosen point is again added to compute the matrix. This matrix refinement process is repeated either for a fixed number of iterations or until the resulting average distance is lower than a defined threshold. Despite the presence of thresholds and the omnipresence of randomness, RANSAC is not very sensitive to critical threshold values and gives robust and reliable results. Indeed, as it is very fast, its number of repetitions is often over-dimensioned, which increases the robustness of algorithms based on random.

4.2.2 A Semi-Manual Approach

Although associating pairs of points is a challenging task, associating points from a similar view is common using Scale Invariant Feature Transform (SIFT) [134] and other algorithms of the same nature. Applied to the successive frames of a race video, one can create temporal pairs of points, *i.e.* points at the same position in space but (probably) different coordinates in the image. Using RANSAC, it it possible to estimate the homography matrix between frames. Thus, one can register a frame with the previous one, projecting it to the coordinate system of the other. It is therefore possible to perform "relative registration" of race video. We call it "relative" because although the frames are all in the same reference frame, they are not associated to any absolute 3D coordinate system.

With this consideration, one can think of a semi-manual registration approach. This idea was developed in [53, 82]. They relied on sparse human video annotation (e.g. one frame per second of video) and used SIFT to determine the camera shift between calibrated frames and the others. Using only one annotation (e.g. on the first frame only) would not suffice, as SIFT does not create perfect spatial matches and the homography between frames is not perfect. This results in temporal drifting, with each new registration slightly wronger than the previous. Regular manual calibrations are thus necessary throughout the video the reduce this problem by resetting the error frequently.

The aforementioned methods do not use deep learning approaches, but it is likely they would benefit from the newest approaches. Instead of relying on SIFT and RANSAC to estimate the homography matrix, newer methods input two subsequent frames and directly regress the matrix [50, 120]. The framework SuperPoint [178] also proposes an improvement on SIFT using deep learning to improve general landmarks detection and matching. These more robust methods would reduce temporal drifting, but they would probably not remove it entirely,

due to sampling approximation. To our knowledge, no sports field registration method implements these techniques.

We also precise it is not possible to input a frame from a race and a generic top-view image of a pool to directly regress the homography matrix between them using [50, 120]: as they are too different, no matching feature appears resulting in unusable output.

4.2.3 Sport Field Registration

The first sport fields registration methods [202, 111] relied on lines and circle detection using Hough Transforms [54]. The detected patterns were used as keypoints and, combined with RANSAC [68], enabled to compute a homography giving the absolute position of the camera view on the field.

Using more recent deep learning approaches, fully automated robust methods appeared. Homayounfar *et al.* [95] created a segmentation map and used a Markov Random Field and an SVM to compute the parameters of the cameras, which determine the homography. Other works [184, 32, 183] used a similar deep segmentation model approach using synthetic datasets. They generated a set of synthetic field views with varying camera angles, extracted features from them, and associated them with their homography (easy to obtain in a synthetic environment). At inference time, they generated similar features from real images, which they compared to their database, giving a good initial homography. Then they adjusted this homography by comparing their input image to their dataset template. In fact, the idea of refining an initial result is present in all recent works of the domain, with different methods for the initialisation. For instance, Jiang *et al.* [106] used a Neural Networks (NN) to directly estimate the image homography. They used another model to refine the matrix by comparing the image and a template projected in the same point of view. Other approaches are based on field keypoint detection. Citraro *et al.* [38] used visual landmarks on the field (mostly line intersections). The main limitation of using visible elements is that the image may not show enough visual keypoints. Nie *et al.* [146] directly address this problem, creating a generic template made of equally distributed points across all the field, which is similar to our proposed approach. The key difference is that in [146] each point is disconnected from the others, despite spatial regularities.

4.3 A More Challenging Benchmark

As Computer Vision (CV) techniques develop, the field registration task reaches excellent performance on existing benchmarks, which does not allow to compare the newest methods with significant margins. This phenomenon is not specific for the task presented in this chapter, and has happened before. For instance,

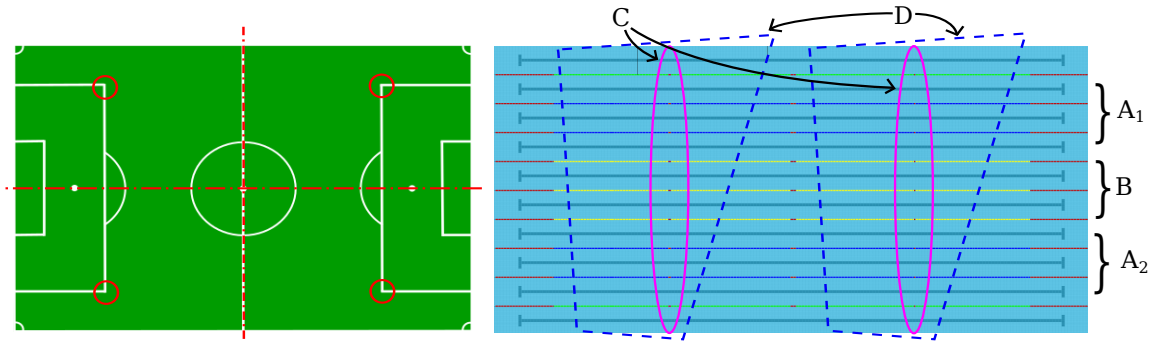


Figure 4.2 – Local appearance ambiguities of a swimming pool. $A_{1,2}$: 4 exact same lines at different places. B: 3 exact same lines in the middle. Both A and B create line mismatch problems. C: the 15m and 35m markers are identical. D: an example of 2 different camera view projections on the pool that display the exact same content, despite being at two completely separate places of the pool. Best viewed in color.

in handwritten numbers classification, the MNIST dataset [49] has reached an accuracy of 99.91% [6], and the most recent classification techniques focus on other more challenging benchmarks (Imagenet, for instance). As a response to adapt to this rapid evolution of registration techniques, we propose to study the unusual sport environment of a swimming pool.

Compared to a soccer field, a swimming pool contains harder patterns to correctly identify and associate with a position on a pool. In Figure 4.2, both fields are shown aside with their distinct features highlighted. Both contain bi-axial symmetry (not represented on the pool for clarity), but the main differences are the visual landmarks. For soccer, each landmark is unique because none is repeated throughout the field. Some of them represent the same things and are in 4 instances (like the ones circles in red), but even in this case they are distinct (with 4 different angles here). Further, the Soccer WorldCup dataset contains very little zoom variations and the camera is always placed closed to the edge center. As such, it is always possible to distinguish more than enough landmarks to know without ambiguity the correct projection of the image in the field.

A pool, however, contains many more challenges (see Figure 4.2, right), namely positioning along the Y axis (A, B), positioning along the X axis (C, D) - both due to landmarks repetitions - and unstable background (wavelets, reflections, light problems etc.). Finally, swimmers occlude part of the landmarks. In a soccer field, that would not be too important, as they are part of bigger patterns (a corner can be inferred by only seeing the two lines creating it, despite their intersection being hidden). In a pool, the majority of the markers are buoys coloured in red instead of yellow or blue. Although there is one of these markers on each line, their size, the waves, and the possibility for a swimmer to hide one, make their detection difficult.

Table 4.1 – Statistics of the RegiSwim^{500}dataset. The races contain important lighting, textural, and spatial variations.

	#images	#races	images / s
Train Standard	226	6	1/3
Train Sequential	150	4	5
Train Merge	329	6	5 & 1/3
Test	174	3	5

To articulate these challenges, we introduce the RegiSwim^{500}dataset, a swimming pool registration benchmark containing 503 manually annotated images of international events associated with a corresponding homography matrix. The source videos are included to enable the use of temporal information. Numeric details of the dataset are summarized in Table 4.1. In the dataset, the level of zoom and distance from the pool also change a lot depending on the competition. This introduces a new challenge in field registration benchmarks, as the notion of scale is not present in Soccer WorldCup due to its general lack of zoom variation. There are two train sets: standard and sequential. The first one has been created in a way similar to WorldCup Soccer and aims to be generic: it contains frames separated by 3 seconds from different matches. As such, a model tackling it only inputs one frame and outputs one homography matrix. The second set has temporally dense annotations (5 annotated frames per second), which can be used to train models with temporal aspects, inputting information on several successive frames (to temporally stabilize the homography output for instance). These two can be merged to create a bigger, temporally heterogeneous dataset. Finally, the test set is also densely annotated, as this makes no difference on a standard benchmark perspective, but it allows also sequential models evaluation. The github page gives an open link to the dataset.

4.4 Registration Method

To find the homography transform from a camera view to a standard top-view, our method uses pairs of points with RANSAC. The overall pipeline is explained in Figure 5.2. The main emphases of this work are computational efficiency and generalization. Other methods [183] claim a fast inference speed but require powerful hardware which may not be accessible in practice. Our method uses a much smaller one-shot model (*i.e.*: without iterative refinement) such that real-time registration is possible with modest hardware (1080 GTX with 8GB). Regarding generalization, it comes from the arbitrary points that are detected on the image: they do not necessarily need to correspond to visual elements on the field, although it helps. This is especially visible with soccer field images, where most landmarks detected by our model are untextured unremarkable grass areas.

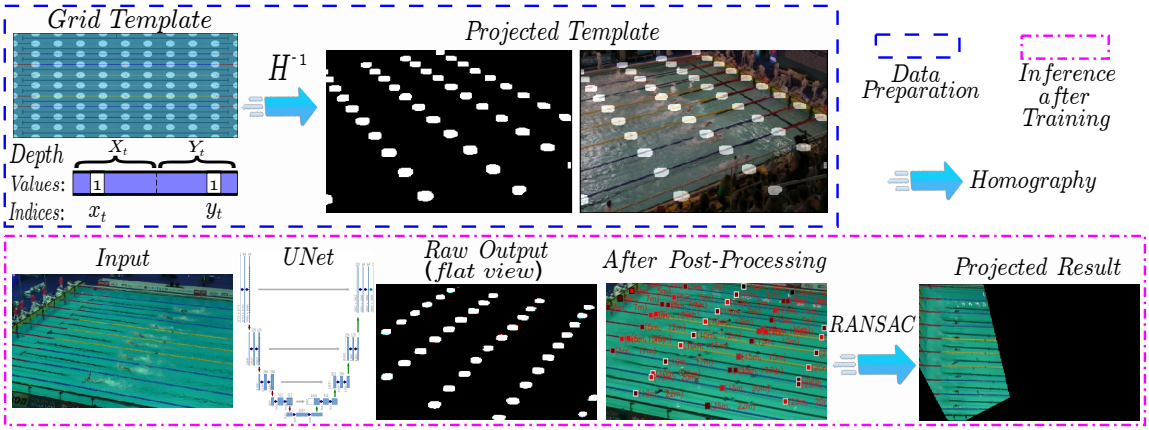


Figure 4.3 – Top: data preparation. A generic template with regularly spaced keypoints is created. The template's depth encodes the keypoints' position in the top-view frame. For each image in the dataset, a corresponding projection of the template is created. Bottom: inference. The model generates a heatmap of keypoints. These keypoints have a known position in the image as well as a known position in the absolute pool coordinate system, encoded in their depth. Using the RANSAC algorithm, they enable the homography matrix estimation, giving the final projection of the input image in the top-view frame. Best viewed in colour.

4.4.1 Template Heatmap

This work proposes a model that, given a $(W \times H)$ input image of a sports field, outputs a $(W \times H \times D)$ heatmap of keypoints, D being the keypoints encoding dimension. The keypoints do not necessarily represent a visual landmark on the field: they are spread regularly, creating a grid (Figure 5.2, "Grid Template"). One unique aspect of this method is the way it encodes the points. The depth vector is composed of two subsets: X_t and Y_t. They are one-hot vectors whose maxima index (x_t, y_t) encode one line/column along the grid axis: a combination of any value of x_t and y_t gives a node position in the top-view frame (Figure 5.2, "Depth").

Compared to having C channels for the C keypoints in the template, as in [146], this method has speed benefits: it avoids the depth to increase geometrically with the number of keypoints. A pair of one-hot vectors only linearly increases the output depth, for the same level of encoding. This improves the speed and scaling of the solution. For instance, a grid of (15×7) contains 105 channels in [146] but only 22 in ours. In addition, as each channel does not only represent one point, but one line/column in the field, their semantic meaning is more interesting and enables a better scene understanding.

4.4.2 Data Generation and Model Training

Once the top-view template is created, the data generation can start using a dataset that contains images with their corresponding homography matrix. The matrix is used to project the template into the point of view of its image (Fig. 5.2, "Projected Template"). With such projection only semantic information has to be inferred.

Our approach relies on a UNet architecture [172], which is widely used for image segmentation. The cross-entropy loss is used to train the pixel-wise keypoints one-hot classification. As there is no "background" class (which would be over-represented in the data), this loss is only applied at the ground truth keypoints location, using a mask. To ensures that the keypoints are at the correct place, the binary cross-entropy loss (BCE) is used. To do so, the ground truth (Truth) and output (Out) heatmaps are flattened with a depth-wise MAX operation. The 2D resulting heatmaps are compared, in order to align the estimated "blobs" with the expected ones. Formally:

$$
\begin{aligned}
L_{class}^{axis} &= CrossEntropy(Out, Truth) * Mask_{keypoints}^{truth} \ , \\
L_{pos} &= BCE(Max_{depth}(Out), Max_{depth}(Truth)) \ , \\
L_{total} &= L_{class}^{x} + L_{class}^{y} + \lambda \cdot L_{pos} \ ,
\end{aligned}
$$

with $\lambda \in \mathbb{R}$ being a weighting coefficient.

4.4.3 Matrix Estimation

To extract the keypoints' absolute position from the heatmap, one could study each pair of (X,Y) channels to verify if each (x, y) point is represented. This results in a $X_G \times Y_G \times K$ complexity (X_G and Y_G being the template grid resolution, and K the number of keypoints to be found). We propose a much faster algorithm whose complexity is in $(X_G + Y_G) \times K$ (the K operations are parallelizable). A depth-wise MAX operation is applied to Out, the whole output, resulting in Out_{flat}, a 2D heatmap (the Max operation is extremely well optimized in processors and insignificant compared to the rest). Its M local maxima are identified and if they exceed a certain threshold, their (x^m, y^m) positions are kept. On Out, the depth vectors at these (x^m, y^m) positions are isolated. Their one-hot vectors return the index of their most activated dimension, (x_t^m, y_t^m), the position on the top-view template. Based on these $((x^m, y^m), (x_t^m, y_t^m))$ pairs, RANSAC [68] can be used to compute the homography matrix. This is formally described in the Algorithm 4.1.

Algorithm 4.1 Fast identification of keypoints on a heatmap. *Det* returns the position of the local maxima in the heatmap. The correspondence table *Tab* associates to each channel an absolute position in the field template.

Require: Model Output Out, Threshold T, maxima detector Det, Correspondence Table Tab
  $Pairs \leftarrow \emptyset$
  $Out_{flat} \leftarrow Max_{depth}(Out)$
  $Max\_List \leftarrow Det(Out_{flat})$
  for (x^m, y^m) in Max_List **do**
    if $Out_{flat}[x^m, y^m] < T$: SKIP
    $depth\_vector \leftarrow Out[x^m, y^m]$
    $X_t, Y_t \leftarrow depth\_vector$
    $x_t^m \leftarrow Tab(argmax(X_t))$
    $y_t^m \leftarrow Tab(argmax(Y_t))$
    $Pairs \leftarrow Pairs \cup ((x^m, y^m), (x_t^m, y_t^m))$
  end for
  $Homography\ Matrix \leftarrow RANSAC(Pairs)$
  return $Homography\ Matrix$

4.4.4 Post-Processing

For individual images, the method can be applied directly, but to register an entire video, no temporal constraint is applied. The method not being perfect, each registration is inconsistent with the others. With our model, the projected top-view of a full race video appears shaking. Stabilization methods can be applied to improve the registration smoothness on videos.

A first straightforward method is temporal averaging of each individual coefficient of the matrix. Each can be taken and plotted through time, as shown in Figure 4.4. A simple approach is using a sliding window to smooth the matrices through time. Outliers (*i.e.* completely wrong matrix estimation) can be easily identified if a given value is too far from its neighbourhood. They are removed before the averaging and replaced by the median of a time window around them. Such smoothing is shown in Figure 4.4. It would also be possible to complexify the smoothing process using the 2/3 Power Law [118], which describes the human motor system's acceleration parameters. One could fit such curve to the matrix elements' temporal signal, and use the result instead of the original matrices.

Instead of smoothing the resulting elements, it is also possible to smooth the position of the points detected on the different frames through time. A point corresponding to a given coordinate in the pool should not move too much between frames. Further, if different distant points are (wrongly) classified as the same, the corresponding neighbour frames' point can vote for the one with

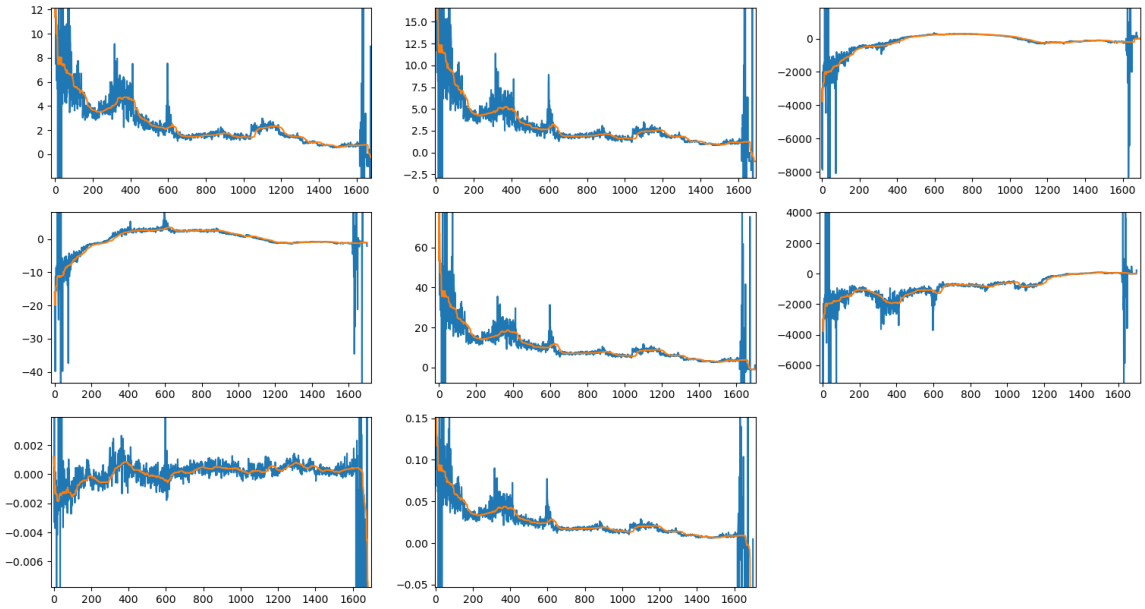


Figure 4.4 – Smoothing of the 8 parameters of the homography matrix estimated for a race. The parameter at position (3,3) (bottom right) of the matrix is always 1. The values are represented through time. The outliers being orders of magnitude different from the truth, they have been cut-out in the figures frame. Blue: the original signal. Orange: the smoothed signal.

the smaller distance to them. This method results in a smaller selection of points for RANSAC, but they are of better quality and give more temporally consistent results. However, such algorithm is much more complex to create and would require a chapter for itself.

As registration benchmarks do not handle temporal data, a quantitative evaluation of the methods is not possible. However, a qualitative appreciation of them on different registered videos is possible. Further, visualizations are presented in Figure 4.4 to showcase the interest of this post-processing. Imprecision in the homography estimation can be understood as noise on the parameters value through time. In consequence, the stabilization is the smoothed signal with much less visible noise.

4.5 Results

The model was trained for 150 epochs with Adam optimizer [112]. The learning rate started at $1e{-}3$ for 50 epochs and was then set to $1e{-}4$ for the remaining 100 epochs, with a batch size of 16. Our metric is the Intersection Over Union (IOU) between binary masks of the ground truth top view and the estimated homography. This is either done with only the visible field (IOU_{part}) or using the

Table 4.2 – Ablation study on the training parameter λ. best in bold.

λ value	0.5	1	2	5
IOU^{avg}_{part}	**83.6**	81.3	83.3	75.0
IOU^{med}_{part}	89.54	84.4	**94.7**	80.4
IOU^{avg}_{whole}	67.6	66.0	**72.6**	63.9
IOU^{med}_{whole}	82.8	85.0	**91.5**	81.1

whole field (IOU_{whole}). The average and median of these metrics are computed on the test dataset. Results are shown in Table 4.3.

4.5.1 Parameter Study

The parameter λ, weighting the importance of landmarks classification with respect to their position, is an important parameter of this method. We compared different orders of magnitude of the value to conclude on its importance. We did no extensive hyperparameter search, as this would only fit the solution to the studied datasets, with no proof of generalization to other contexts.

This parameter is not trivial to weight, as is represents the balance between the markers' classification and position loss. As the markers mask (serving to the position loss) is made of the maximum of each channel, it is better for it to put all the channels to 1 at the places of interest. On the contrary, only 1 channel must be maxed to optimize the classification loss. Antagonist behaviors naturally emerge from them. It is thus important to now how to balance one with respect to the other.

With a value of $\lambda = 2$, the results are the best by a significant amount. With lower values, the results are similar yet less precise, but it seems that with a λ too big, here 5, there is an important drop. The model gives too many importance to the mask precision and neglects the points classification, although it intuitively seems like the most important one, being responsible for the pairs of points mapping.

4.5.2 Comparing to State of the Art

Although our approach does not quite reach the top results from the literature, it is still among the best ones, as shown in Table 4.3). This is remarkable, considering it contains no refinement process while all the other methods do. However, this impacts the IOU_{whole} metric, where the slightest shift on the visible side of the field has big repercussions on the other side. Nonetheless, this second metric can be considered less interesting for real-world applications, such as placing the players on a field, as they must be visible on image to be detected in the first

Table 4.3 – Quantitative results on Soccer World Cup and RegiSwim^{500}datasets. Best in bold. Real-time methods underlined.

Method	Benchmark	IOU^{avg}_{part}	IOU^{med}_{part}	IOU^{avg}_{whole}	IOU^{med}_{whole}	FPS	Memory - GPU
Citraro *et al.* [38]	WorldCup	93.9	95.5	-	-	9	NA - Titan RTX
Sha *et al.* [183]	WorldCup	94.2	95.4	83.2	84.6	<u>250</u>	**48GB - Titan RTX**
Chen *et al.* [32]	WorldCup	94.5	96.1	89.4	**93.8**	2	16GB - NA
Jiang *et al.* [106]	WorldCup	95.1	96.7	89.8	92.9	0.74	8GB - 1080 GTX
Nie *et al.* [146]	WorldCup	**95.9**	**97.1**	**91.6**	93.4	2	8GB - 1080 GTX
Ours, soccer field	WorldCup	94.6	95.9	81.2	86.0	<u>50</u>	8GB - 1080 GTX
Ours, swimming pool	RegiSwim^{500}	83.3	94.7	72.6	91.5	<u>50</u>	8GB - 1080 GTX

place. These results might be improved using methods such as self-training on unlabelled data.

Regarding speed, our model is one of the only two exceeding real-time (> 25 FPS), although it has been tested on the least powerful hardware according to benchmarks [2, 47]. Looking in the details, one can even argue that our model is faster than Sha *et al.* [183] on the same hardware. Indeed, our architecture is a subset of theirs, to which they add 2 more deep models, a Spatial Transformer Network, and an exhaustive search among field templates. All these additional steps have a significant time cost and our method might be faster by up to this amount. The model's speed could be increased even more using distillation [91] to train a more condensed, shallower and faster version of UNet. However, registration is far from being the current speed bottleneck of the pipeline, so such optimization is not necessary.

Naturally, for our more challenging RegiSwim^{500}dataset, the performance is lower. Our model handles correctly Y-axis challenges (A and B in Figure 4.2) and lighting problems, mostly because of the grid density and distribution, which prevents focusing on a single part of the image. The big difference between the mean and median result is due to multiple left-right inversions. In this failure case, the IOU score can drop down to 0, reducing the mean but not the median as they are in minority. These are quite difficult to prevent in a pool (challenges C and D in Figure 4.2). This first baseline clearly shows the challenges and limitations raised by this new benchmark. Calibrating a pool, especially with different levels of zoom and multiple camera placement, is more challenging than what Woccer WorldCup proposes.

4.5.3 Failure Cases

In various situation, our trained model did not perform well. This can be observed by projecting a video in top view (without temporal smoothing), where misprojected frames appear obvious. One can also observe this phenomenon with the benchmarks, by displaying the images with the lowest scores.

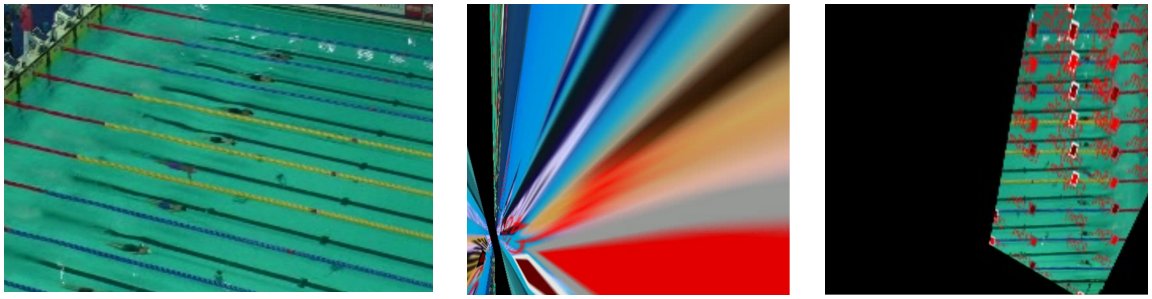


Figure 4.5 – Examples of failure cases. The results were obtained from two models trained on different condition. The Red squares represent the detected landmarks. Despite the mirror failure looking consistent, its IOU with the ground truth is 0.

4.5.3.1 Wrong Classification

First, there are frames where a majority of the detected landmarks are not correctly classified. It is rare that they are placed at a random position, so it is usually a classification error more than a segmentation (*i.e.* landmark placement) problem. There is a proportion of wrong points from which RANSAC cannot correctly compute a coherent homography matrix anymore. The result of an image projected in such condition is not exploitable at all. An example is shown in Figure 4.5, center.

4.5.3.2 Mirror Field

A similar failure case is when part of the landmarks are wrong in a coherent manner. For instance, all the 15m markers are falsely classified as 35m markers, as in Figure 4.5, right. Here, the output is a plausible result, but with a left-right inversion error. This failure is harder to automatically detect than the previous one, either with a human eye or with a model trained to classify images as possible results or not, as it appears correct without the original image. It can be the cause of important errors.

4.5.3.3 General Geometric Misunderstanding

Both of these failure case show an important limitation of our method: the model does not have an understanding of the pool's spatial disposition. If 5m and 25m markers are detected with a high confidence (and they tend to be, being easy visual markers), the model should not give a high probability that a 35m marker is between them. This is not spatially coherent. However, the model fails to do this logical operation. An idea to solve this problem might involve a Generative Adversarial Networks (GAN). Indeed, one can easily differentiate a labelled heatmap and a generated heatmap solely using this geometry criterion.

Adding a discriminator's loss could force a spatially logical output. However, it is never simple to use GAN and we leave that to posterity.

4.5.3.4 Important Zoom

The last type of common failure is when not enough landmarks are detected. In the majority of cases, this happens when the level of zoom is too high and it is impossible to see multiple markers, or to find a scale reference. In these cases, even a human could not register the image. Using the space between swimming lanes gives the camera angle from the water surface, the scale of the different elements gives the level of zoom, but it is not enough. One can only say what the camera does *not* frame. To circumvent such problem, one must not zoom too much on the pool, in order to keep enough spatial context. Temporal information can also help, as in [146], but a tracking-based registration method is outside of the scope of this chapter.

4.6 Discussion on the One-Shot Approach

The advantages of a fast method may seem obvious, especially with good results as in our case, but one must consider the application first, before judging it. In CV, the speed vs precision trade-off is ubiquitous and sports field registration is no exception. Before developing a method, one should think about where they want it on this trade-off spectrum in the context of their application. Further, we announce a given speed using our method. The speed is in fact entirely dependant on the UNet model we chose (here, the original one in [172]). To our knowledge, no extensive studies have been done on precision losses in function of the model size, in the case of registration. If speed were a higher constraint, one could reduce its complexity by removing a layer or reducing the number of filters. On the other hand, it is also possible to increase the UNet model's size to improve the results, if that were more critical than speed. One could even use a few refinement steps to that purpose, as long as an inference time is not crossed. For all the presented results in Table 4.3, an arbitrary refinement limit is chosen. Practically, they correspond to a point where improvement is too little to be considered worth the time spent.

This thesis is very application oriented, with many contributions directly used in a tool destined to the Fédération Française de Natation (French Swimming Federation) (FFN). To create and optimize said tool, one must consider the actual needs in precision and in speed, before choosing a registration method. The one presented in this chapter is fully compatible with any other refinement method, as it can serve as initialisation model. Depending on the final use of this work, one could use any of the many refinements proposed in the related works.

4.7 Conclusion

This work introduces an efficient and precise method for automatic sports fields registration, which reaches very good performance and real-time inference speed. It is very well fitted to online practices, such as live-stream broadcast analysis, or post-race performance review.

The RegiSwim^{500} dataset has been introduced and made publicly available in order to improve the registration challenge. Future works will include ways to optimize even more the model's inference speed, and new methods to increase its precision.

The model, however, is not perfect and cannot handle all the possible video cases. Several limitations have been listed, with propositions to address them. Further, it was shown that a simple temporal sliding mean can smooth results and get rid of many anomalies if the majority of the video is correctly enough calibrated. Finally, the importance of a fast method is discussed in relation to the end-application of such method. In CV, there is always two metrics: one relative to performance and another to speed. A user must always consider which number they want better than the related work in priority.

CHAPTER 5

PERIODICITY

Contents

Chapter abstract

We introduce a context-agnostic unsupervised method to count periodicity in videos. Current methods estimate periodicity for a specific type of application (e.g. *some repetitive human motion). We propose a novel method that provides a powerful generalisation ability since it is not biased towards specific visual features. It is thus applicable to a range of diverse domains that require no adaptation, by relying on a deep Neural Networks (NN) that is trained completely unsupervised. More specifically, it is trained to transform the periodic temporal data into some lower-dimensional latent encoding in such a way that it forms a cyclic path in this latent space. We also introduce a novel algorithm that is able to reliably detect and count periods in complex time series. Despite being unsupervised and facing supervised methods with complex architectures, our experimental results demonstrate that our approach is able to reach state-of-the-art performance for periodicity counting on the challenging QUVA video benchmark. Its remaining limits will be addressed by*

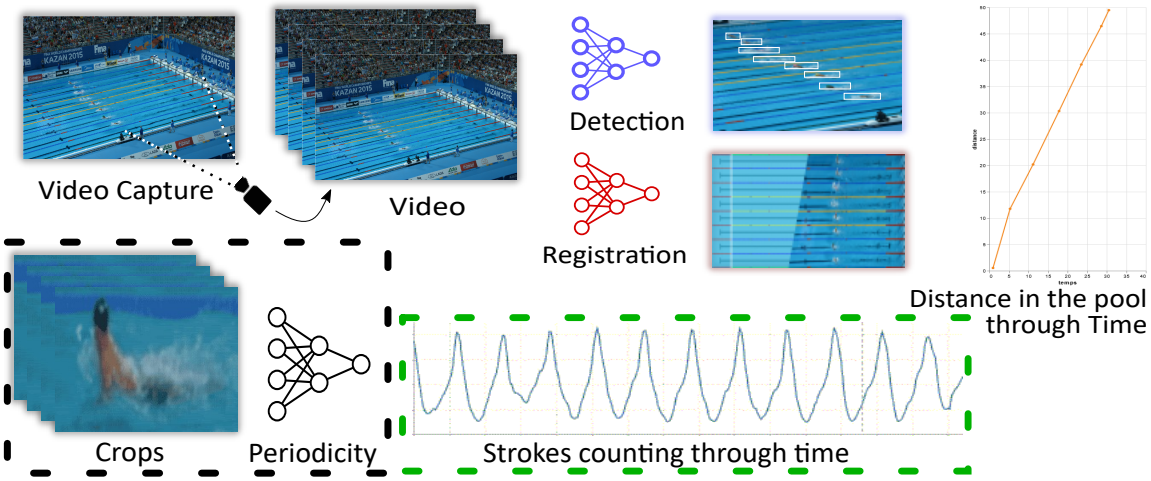


Figure 5.1 – The swimming race analysis pipeline. The periodicity counting part, framed in black, is tackled in this chapter. This part is built on top of swimmers detection because it relies on sub-videos crops around swimmers. It can output the number of cycles per pool length or the duration of cycles (framed in green). Both metrics are used by coaches.

> *an additional method based on supervised training and an annotated dataset. This chapter is based on the work from:*
> **Nicolas Jacquelin, Stefan Duffner, Romain Vuillemot**. *Periodicity Counting in Videos with Unsupervised Learning of Cyclic Embeddings*. Pattern Recognition Letters, Elsevier, 2022, ⟨10.1016/j.patrec.2022.07.013⟩, ⟨hal-03738161⟩.

5.1 Introduction

We define periodicity as any phenomenon that happens multiple times in a similar way over time. Periodicity is ubiquitous in real-world scenes and occurs at multiple scales. In elite sports, the tracking of the athletes' motion is a key issue and is highly repetitive. In swimming, in particular, the stroke pace is one of the most important metrics to determine a race quality and infer other statistics (*e.g.* stroke amplitude, rate etc.). Combined with swimmers' position in the pool, it gives almost every information a coach needs to grasp a complete and quantitative understanding of performance. In the automatic swimming races analysis framework, it intervenes at the end, as shown in Figure 5.1.

This task is challenging for many reasons. First, two successive repetitions may significantly differ (*e.g.* swimming strokes rate and amplitude change during the race, as well as the swimmer's position with respect to the camera). Second, the precise beginning and end of a cycle are ambiguous. Finally, there exist other artefacts, such as the different sub-cycles that may be mistakenly detected as cycles. Furthermore, the notion of periodicity is context-dependent: the same

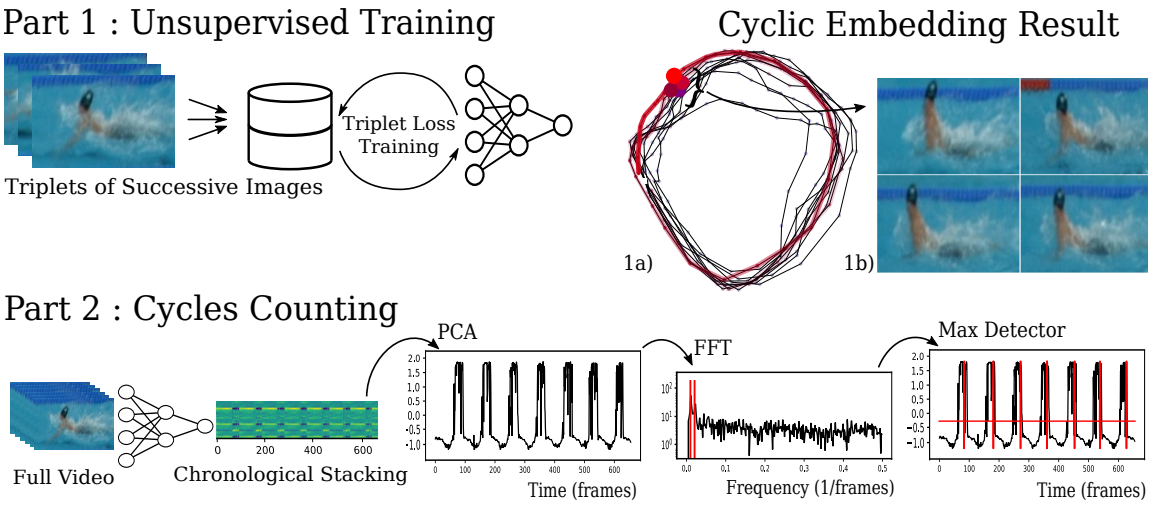


Figure 5.2 – The periodicity counting framework. In Part 1, a Convolutional Neural Network (CNN) is first trained in an unsupervised way on the data to analyze, as described in Section 5.3.1. Then, it is used to extract an embedding for each image of a video. (1a) shows an example of the 2D PCA projection of these embeddings. The last 50 embeddings are linked chronologically (in red), revealing the cyclic path. (1b) shows the input images whose embeddings correspond to the highlighted points. They belong to different swimming cycles but correspond to the same phase, therefore the points are close in the latent space. In Part 2, we chronologically stack the embeddings obtained from the trained network to form a multi-variate time signal. The PCA's first component of the signal reduces it into a uni-variate signal. Finally, our *Max Detector* algorithm is used to count the cycles on the signal, which corresponds to the number of cycles on the video. Best seen in color.

event in two different sequences might be periodic or not depending on whether it is repeated or not. Therefore, the signal must be studied globally and not frame-wise.

Estimating periodicity is particularly challenging with videos recorded under unconstrained conditions. Any spacial shift, background noise or viewpoint change results in important variations in the captured signal, which often makes it hard to automatically detect the dominant cycle. Although these problems can be alleviated with recent Machine Learning (ML) methods based on CNN that are capable of extracting noise-robust abstract representations of an image [64], those Deep Neural Networks (DNN) models often require large amounts of training data [**coco**, 48, 89]. In the case of swimming periodicity counting, such dataset does not exist. This issue is often circumvented by pre-training such networks on large annotated datasets [88, 58], but then the model may be biased towards specific visual features which may not be relevant for the task at hand and thus lead to a lower performance [195, 152]. In particular, NN trained on daily-life context do not transfer well in the new domain of a pool. Its specific visual features and colour distributions are too strong biases. To tackle the periodicity counting problem in

videos, state-of-the-art methods [56, 221, 222, 215] are trained on Kinetics [109], a videos dataset of persons doing repetitive actions. These models are thus domain specific to humans doing a finite number of activities. Their performance is likely to drop when they are used on less frequent domains, such as astrophysics or medical videos, but also on human activities not present in kinetics (swimming). In contexts which are too different from the initial training domain, a new dataset is required to adapt the model, which is extremely time consuming and costly to create. Moreover, not all periodicity problems concern videos of regular human activities: there are other types of complex time series, like multi-source sensors monitoring air quality or biophysical activities [7, 198, 116], 4D MRI videos (*i.e.* 3D images through time) of breathing lungs [100], active brains [8] or beating hearts [194]. For these reasons, it is important to have a domain-agnostic method.

This chapter presents such a technique introducing a new training method suited for temporal periodic data in general. With an adapted NN architecture, it could even be used outside of the video domain to study other types of multivariate time-series.

Our approach is summarized in Figure 5.2. The training is made on the test video using unsupervised learning. Once trained, the model reduces the video into a periodic 1D signal and counts its repetitive patterns using a novel peak detection algorithm based on various signal processing techniques. This counting process is performed in a single step. It does not require to test different time-scales, or to use a sliding window through the whole signal to process it completely. The computational cost is therefore greatly reduced compared to other methods based on transformer architectures [56] or multimodal fusion models [222].

Further, during this thesis, we launched the creation of a labelled dataset for supervised learning of swimming periodicity. This chapter will explain what composes it and how to train a model from it. Qualitative results will be shown and analyzed. This new model addresses the remaining challenges not completely addressed by the unsupervised method. Our main contributions are the following:

- An unsupervised method to train a NN with the triplet loss to encode any kind of video (Section 5.3.1).
- An algorithm to count the periodic patterns in time-series (Section 5.3.2).
- A framework combining these algorithms for automatic periodicity counting in videos, based on the analysis of a learnt embedding.
- a swimming periodicity dataset.
- a method to train a model on such dataset (Section 5.5).

5.2 Related Work

To analyse videos recorded under unconstrained conditions, recent approaches use CNN, as they are the current state of the art for image classification [192],

action recognition [159], objects tracking in videos [37] and saliency detection [11]. They are also used in periodicity *detection* [153, 56], which is very similar to periodicity *counting*: the first classifies each frame of a video as periodic or not (the PERTUBE dataset [153] typically is used as a benchmark), whereas the latter operates on a periodic video and counts the repetitions.

To specifically address periodicity counting in daily life videos, Levy and Wolf [126] proposed a 3D CNN architecture: the input is composed of 20 chronologically ordered images, each separated by N frames in the timeline. In this way, the temporal information is integrated into the input. They trained the model in a supervised way on synthetic data to separate the sequences on their temporal dimension. This feature-oriented method is robust to colour and lighting variations, but one needs to test several timescales (*i.e.* many different values of N) in order to obtain good results. Also, as for supervised trained models, the performance directly depends on the dataset size and quality.

Similar to our method, other works aim to reduce a video to a one-dimensional signal. Polana and Nelson [158] detected the pixels responsible for motion, and considered them as temporal signals varying throughout the video. They extracted a signal period by detecting the peaks on its Fourier Transform. Yang, Zhang, and Peng [213] used a method based on pixel-wise joint entropy to estimate the similarity between a reference image and the other ones, resulting in a 1D temporal function.

Runia *et al.* [175], introduced another method to convert a video into a 1D signal. They studied the main direction of the foreground's optical flow in order to create multiple 1D signals from its directional gradient components through a wavelet transform. Their paper also introduced the QUVA benchmark dataset for periodicity counting in everyday videos.

More recently, Dwibedi *et al.* [56] proposed a complex architecture mixing CNN and transformers [197], trained in a fully-supervised fashion on the Countix dataset which they introduced themselves. In their experiments, they also trained their model on a considerable amount of synthetic data obtaining impressive results, but unfortunately they did not publish this dataset. This method achieves good results on public benchmarks, but it is by far the most computationally expensive and data dependant. Using the Countix dataset, Zhang *et al.* [222] proposed a multi-modal approach relying on sound and sight to improve the state-of-the-art on the Countix benchmark. They did no evaluation it on QUVA, however.

The work of Yin *et al.* [215] shares some similarities with our work, as it also extracts periodic features from a video with a learning-based method, reduces it to a 1D signal, and counts the repetitions with an algorithm relying on the Fourier transform. However, their approach is not generalizable to other types of data since it uses a NN that is pre-trained on a large annotated video dataset (Kinetics [109]) in a supervised way. As such, they can only analyze conventional

videos of 2D images and the learnt visual features are domain dependant, which may not give satisfactory results on other types of videos. In addition, the signal processing part of their method is quite different from ours. To detect the dominant frequency, it uses a specific multi-threshold filter in the frequency domain with several empirically determined thresholds, and then detects the peaks in the reconstructed signal with the inverse Fourier transform. Our model is trained unsupervised and end-to-end, and our robust peak detection algorithm operates on the original 1D signal obtained from PCA.

Zhang *et al.* [221] proposed an approach based on a context-aware model. However, it is not designed to generalize to unseen domains: the method uses the Kinetics dataset [109], where a separate model is trained for each sports type resulting in excellent overall scores on public benchmarks. Finally, the work of Feirrera *et al.* [65] is also context-specific: it uses human pose classification to count repetitions of workout routines. This approach is suited but limited to the context of human motion repetition counting.

As most of these methods ([56, 222, 221, 215, 65]) are trained on a human motion video dataset (Countix being built on top of Kinetics), they are well adapted to human gestures and actions. However, this makes them (i) specific to videos and not any other type of input data and (ii) biased towards human motion. On the contrary, we designed our method to be applicable to any type of periodic data.

5.3 Unsupervised Periodicity Counting

We introduce a novel unsupervised learning process, illustrated in Figure 5.2 Part 1, to encode a video in a way that highlights its periodic features. To that purpose, a CNN is trained directly on the video to be analyzed. The resulting video embedding is a periodic signal that is processed by a novel algorithm to count its cycles.This new method does not follow the classical training/validation/test protocol. The different steps of the pipeline are describes in detail in this section.

5.3.1 Latent Representation Learning

Before the model can be trained, one needs to group successive frames from the video. The frame at time index t is grouped with the frames $t+1$ and $t-1$ forming a triplet. Each frame belongs to 3 different groups (triplets) where it plays the 3 roles $t-1$, t and $t+1$, except for the first and last frames (because there is respectively no frame before it to be $t-1$ and no frame after it to be $t+1$). With T frames in the video, there are $T-2$ triplets in the end.

The output vector of the image at time index t is called $\phi(t)$. The images need to be embedded by the CNN in such a way that, in chronological order, they form a repetitive pattern in the latent space, *i.e.* a loop. This is achieved by using

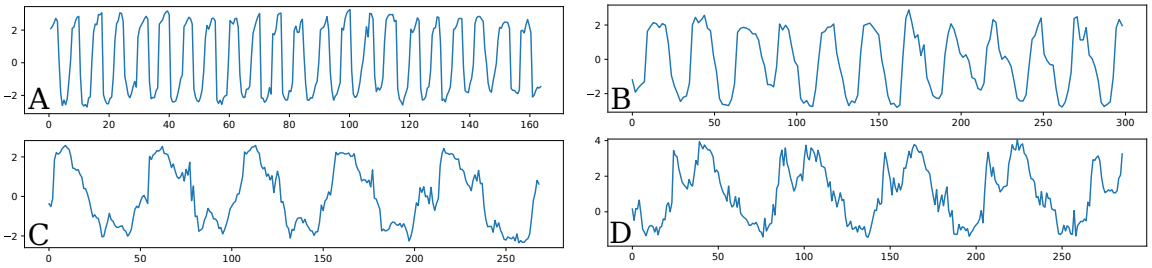


Figure 5.3 – Examples of 1D PCA projections of embeddings. The result's amplitude is displayed through time. A, B and C curves show the embedding results for different cycles durations, from 8 to 50 frames per cycle (on average). D shows a more complex pattern containing 2 distinct local maxima. In such cases, our *Max Detector* might count 2 cycles per pattern, resulting in a false result, like mentioned in Section 5.4.3.

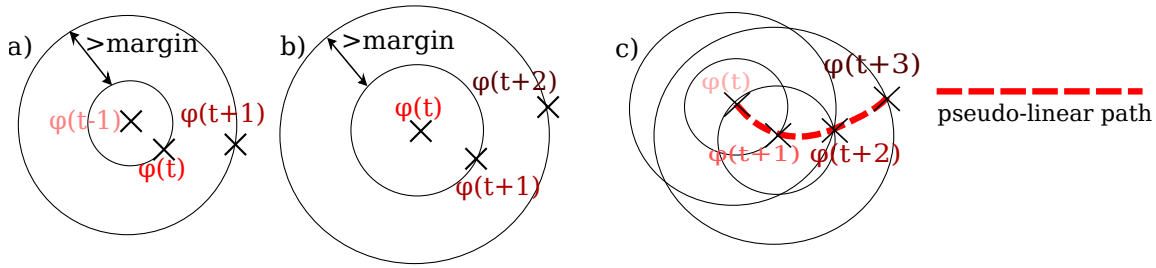


Figure 5.4 – Unsupervised learning of the pseudo-linear path using the Triplet Loss. The anchor is at the center, the positive is on the smaller circle (not necessarily the same size each time), and the negative is outside of the bigger circle. a) The anchor is $\phi(t-1)$, so $\phi(t)$ and $\phi(t+1)$ are separated. b) The anchor is $\phi(t)$ $\phi(t)$ and $\phi(t+1)$ are drawn together. When the training starts, the negative can be at the other side of the big circle compared to the positive. But this situation is no longer possible when the constraint is applied to all the successive frames after convergence, as shown in c): a pseudo-linear path is naturally formed, as it is the only way to respect the constraints imposed by the loss. Best seen in color.

a continuity criterion and a periodicity criterion. The first forces the images' successive embeddings to be temporally ordered along a pseudo-linear path. The latter forces this path to contain repetitive patterns.

To guarantee the continuity criterion, the triplet loss is used as objective function:

$$L(A,P,N) = \max(0, |\phi(A) - \phi(P)| - |\phi(A) - \phi(N)| + \alpha) \,, \tag{5.1}$$

where $\alpha \in \mathbb{R}$ is the margin, A is the anchor, P is the positive and N is the negative image. The purpose of the triplet loss is to make the distance between the embeddings of A and N larger than the distance between the ones of A and P up to a minimum distance defined by α. Our approach defines the image at time index $t-1$ as the anchor, t as the positive and $t+1$ as the negative. The overall consequence of applying this training method to each value of t in the video is that each $\phi(t)$ is "pulled towards" its direct neighbors ($\phi(t-1)$ and $\phi(t+1)$), and

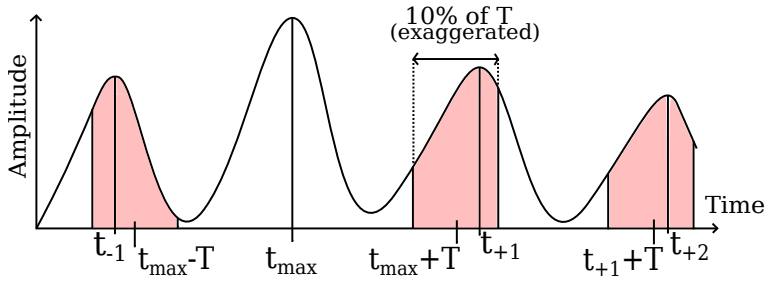


Figure 5.5 – Illustration of *Max Detector*. Starting from the global maximum's index t_{max}, the algorithm shifts by one period T and finds the maximum's index t_{+1} in a window of 10% of T (in red, exaggerated for a better understanding). This window makes *Max Detector* robust to period variations. Starting from t_{+1}, this is repeated to find t_{+2}, t_{+3} and so on until reaching the signal's end. A first iteration goes from t_{max} to the end of the signal and a second from t_{max} to $t = 0$. Best seen in color.

"pushed away" from its 2^{nd} degree neighbors ($\phi(t-2)$ and $\phi(t+2)$). Therefore, the positive embedding is "placed" between the anchor and the negative one, with a tolerance of α, as explained in Figure 5.4. This forces the creation of a pseudo-linear path chronologically aligning the embeddings in the latent space.

To guarantee the periodicity criterion we rely on the property of CNN that two similar inputs will have similar outputs unless explicitly trained otherwise [76]. With periodic videos, if one cycle has a period T, then the images at time indexes t and $t+T$ will have the same phase in the cycle and look alike. Therefore, the images have an embedding close to the other images corresponding to the same phase in the cycle. This cyclic behavior is illustrated in Figure 5.2, images 1a) and 1b).

The resulting model closely fits the data it was trained on. Therefore, to get the most adapted latent space representation for a video, a model needs to be specifically trained on it (and no other videos). This requires some training time, but, as explained in Section 5.4.1, it is not too expensive.

The training process has been presented using frames as a temporal unit, but it can be enriched by other information. In Section 5.4.2, we show that adding the optical flow to a frame gives better results (*i.e.* frame t is enriched with the optical flow between frames t and $t+1$). In this case, we concatenate the 3 image channels (RGB) to the 2 optical flow channels (direction & magnitude) resulting in $5 \times W \times H$ temporal unit tensors (W and H being the width and height of the video). This section presented a way to fit a latent space to a video, but it also works for other complex time series. Similarly to adding the optical flow, which is the variation of a frame with respect to the next one, one could add the gradient between successive temporal vectors to augment the information encoded by the model.

5.3.2 Cycle Counting

After training, the images in the video are embedded in the latent space in such a way that they form a cyclic pattern. The next step, illustrated in Figure 5.2, Part 2, is to count these cycles.

In order to effectively work in the frequency domain and apply common signal processing techniques, the model's output vectors have to be transformed into a one-dimension signal. To do so, the embedding vectors of the M images are chronologically "stacked" to form a matrix like in Figure 5.2, 2a). This is, if the latent space has D dimensions, the resulting matrix is of size $D \times M$. A PCA projection is applied to the matrix in order to keep the features combination with the most importance. By only keeping the 1^{st} element of the PCA, it results in a $1 \times M$ temporal signal S with periodic information, *i.e.* a recurring pattern like in Figure 5.3, corresponding to a repetition in the video.

The subsequent algorithm uses the Fourier Transform to detect the signal's F main frequencies. These candidate frequencies will all be tested by our proposed algorithm named *Max Detector* explained in the following.

The main goal of *Max Detector* is to detect the maximum of each cycle in S, and to save their time indices in a list named *MaxList*. These maxima will be used to distinguish and count the cycles. We name f_i the current analyzed frequency (one of the F detected by the Fourier transform), *Max List*$_i$ its corresponding maxima list, and T_i its corresponding period. *Max Detector* starts by finding the signal S global maximum's time index, which is added to *Max List*$_i$. We suppose the neighbour cycle maxima are approximately one period away from each other. Therefore, to find the next maximum, one creates a time window by shifting of $T_i \pm 10\%$ from the current maximum. In this window, the local maximum is located and its time index is added to the list *Max List*$_i$. This operation is performed again from this new local maximum, until reaching the signal's edge. This procedure is repeated twice, each time starting from the global maximum: once forward towards the end, and once backward to the beginning of the signal. This is graphically explained in Figure 5.5 and formally explained in Algorithm 5.1.

Once the F different frequencies have been processed, there are F different candidate lists *Max List*$_i$. Each list is evaluated individually and the best solution is retained. To evaluate a *Max List*$_i$, each of its local maxima will be compared to their local region accordingly to equation 5.2. This score computes the proportion of elements in *Max List*$_i$ that correspond to the local maximum in half a period centered on them.

$$Score\_i = \frac{1}{L_i} \sum_{k}^{Max\_List\_i} \left[S[k] = \max\left(S[k - \frac{T}{4} : k + \frac{T}{4}] \right) \right] , \tag{5.2}$$

L_i being the number of elements in *Max List*$_i$ (*i.e.* its length), k representing the different local maxima indices. As a result, a list that contains each and every

Algorithm 5.1 Max Detector: creation of candidate lists $MaxList_m$

Require: signal S, $f_m, m \in (1, ..., F)$
  $MaxList = \emptyset$
  for m in (1, ..., F) **do**
    $MaxList_m = \emptyset$
    $T_m = 1/f_m$
    $t_0^{max} = \arg\max S(t)$
    $MaxList_m \leftarrow MaxList_m \cup t_i^{max}$
    $t_i^{max} = t_0^{max}$
    while $t_i^{max} - T_m \geq 0$ **do**
      $t_i = t_i^{max} - T_m$
      $W_i = (t_i - 0.1 \cdot T_m \ , \ t_i + 0.1 \cdot T_m)$
      $t_i^{max} = \arg\max_{t \in W_i} S(t)$
      $MaxList_m \leftarrow MaxList_m \cup t_i^{max}$
    end while
    $t_i^{max} = t_0^{max}$
    while $t_i^{max} + T_m < length(S)$ **do**
      $t_i = t_i^{max} + T_m$
      $W_i = (t_i - 0.1 \cdot T_m \ , \ t_i + 0.1 \cdot T_m)$
      $t_i^{max} = \arg\max_{t \in W_i} S(t)$
      $MaxList_m \leftarrow MaxList_m \cup t_i^{max}$
    end while
    $MaxList \leftarrow MaxList \cup MaxList_m$
  end for
  return $MaxList$

local maxima of the signal separated by approximately T has a score of 1. On the contrary, the more incorrect maxima a list contains, the lower its score is.

The list with the highest score is kept , whose number of elements represent the number of cycles in the signal and therefore the number of repetitions on the video.

5.4 Experiments and Results

To compare our method with the current state of the art, we used the QUVA [175] and Countix [56] benchmarks. QUVA is composed of 100 videos showing between 4 and 63 repetitions. The videos are very diverse and recorded in real-life situations, often with camera motion and background variation. Countix contains a similar visual variety. It is the first large video repetition dataset, containing more than 8000 clips showing 2 to 73 repetitions. The metrics used for performance comparison are the Mean Absolute Error (MAE) and the Off-By-One Accuracy (OBOA), defined as:

$$MAE = \frac{1}{N}\sum_{i}^{N}\frac{|c_i - \hat{c}_i|}{c_i} \qquad OBOA = \frac{1}{N}\sum_{i}^{N}[|c_i - \hat{c}_i| \leq 1] \ ,$$

where c_i is the true count and $\hat{c}_i$ is our model estimation on the same video i and N is the number of videos in the dataset. The OBOA, introduced in [175], counts the proportion of correct predictions with a tolerance of 1. This margin serves to reduce the importance of rounding mistakes, as ambiguous cycle cut-offs can happen at both ends of the video.

Each model was trained independently on one video at a time. This means that for a dataset of 100 videos like QUVA, 100 different models have been trained and evaluated for each experiment (except said otherwise). The following sections describe the experiments performed on the two benchmarks and the results obtained with the two metrics.

5.4.1 CNN Architecture

During our test phase, we did not notice a significant difference of performances using different CNN architectures (we tried VGG19 and VGG11 [185], results shown in Table 5.1). We also designed a straightforward CNN model with fewer layers than VGG11 as it would train better on the few images of the video clips. Our custom model is composed of 6 layers of 3×3 convolutions with ReLU activation [145], each layer doubling the number of filters (starting at 4, finishing at 128) and 2×2 max pooling [179] after each layer, and a final global average pooling giving a 32 dimensions output vector.

For each study, we trained a model for 30 epochs with a batch size of 16, a learning rate of 10^{-3} and the Adam optimizer [112]. Under these conditions, the

Table 5.1 – Results of different variations of our approach on the QUVA dataset. Pretrained models did not perform well at embedding the images in a cyclic manner. The same architectures, trained using our method, give much better results. Different architectures do not change the results.

Variations	MAE$\pm\sigma\downarrow$	OBOA $\uparrow$
1 img + F=4	0.388$\pm$ 0.512	0.43
VGG19 (pretrained) + F=4	0.758$\pm$ 0.812	0.21
VGG11 (pretrained) + F=4	0.783$\pm$ 0.761	0.17
VGG19 + flow + F=4	0.252$\pm$ 0.400	0.60
VGG11 + flow + F=4	0.241$\pm$ 0.367	0.62
flow + F=2	0.291$\pm$ 0.445	0.59
flow + F=5	0.239$\pm$ 0.335	0.62
flow + F=7	0.244$\pm$ 0.328	0.61
flow + F=10	0.378$\pm$ 0.710	0.57
flow + Scholkmann *et al.* [181]	0.307$\pm$ 0.408	0.51
flow + F=4	**0.231$\pm$ 0.326**	**0.64**

training took about 1.1 times the total duration of a video using a NVIDIA GTX 1080 GPU.

5.4.2 Ablation Study

Our initial baseline CNN model just takes one image as input (Variation “1 img" of Table 5.1). To improve performances, we enriched the input with the optical flow between two consecutive frames, similar to Zhou *et al.* [225], as mentioned in Section 5.3.1. The new input is therefore made of an image concatenated with the optical flow from this image to the next one. This variation is named “flow" in Table 5.1.

To show the importance of our training policy, we used common CNN models trained on Imagenet [48] to do the embedding, with only one image as an input, as required by these architectures (they were not retrained on the cyclic videos images). The obtained embeddings did not give easily exploitable cyclic curves, resulting in bad performance. With our training policy, however, the different CNN architectures all reached comparable results, our shallow model being better than the deeper ones. For all lines in Table 5.1 not stating a specific architecture, we used our custom shallow CNN.

In the *Max Detector* algorithm, we compare F different frequencies. As shown in Table 5.1, we studied the performance obtained for different values of F. The QUVA benchmark does not provide a specific evaluation protocol, so we used cross validation on QUVA with 50/30/20 splits (*i.e.* random splits with said sizes were created to evaluate different values of the parameter F without changing

Table 5.2 – Results for different methods of periodicity counting methods. Bold: the best result of a category. Underlined: the second best. Our unsupervised method reaches comparable performances to the best fully-supervised models. This proves the overall interest of our method. Q for QUVA benchmark, C for Countix.

Method	Unsupervised	Q: MAE$\pm\sigma\downarrow$	Q: OBOA $\uparrow$	C: MAE$\pm\sigma\downarrow$	C: OBOA $\uparrow$
Levy and Wolf [126]		0.482$\pm$ 0.615	0.45	-	-
Yin *et al.* [215]		**0.199** $\pm$ 0.335	-	-	-
Dwibedi *et al.* [56]		0.322	**0.66**	0.364	**0.697**
Zhang *et al.* [222]		-	-	**0.307**	0.511
Pogalin *et al* [157]	✓	0.389$\pm$ 0.376	0.49	-	-
Runia *et al* [175]	✓	0.232$\pm$ 0.344	0.62	-	-
Our method, F=4	✓	0.231$\pm$ **0.326**	0.64	0.495 $\pm$ 0.769	0.517
Our method, F=2	✓	0.291$\pm$ 0.445	0.59	0.419 $\pm$ 0.496	0.545

anything else, in particular the temporal input signal). The results were the same for the different splits: between 4 an 7, F seems to have little impact on the result, $F = 4$ being the optimum. On the other hand, Countix has a training dataset, which we used to compute the best value for F. The results were similar between 2 and 7 again, obtaining an optimum for F=2.

Finally, we measured the importance of *Max Detector*, so we used another automatic peak detection algorithm, described in [181] by Scholkmann *et al.*. It counts the cycles of the same signal as our *Max Detector*, but performs significantly worse. This shows the effectiveness of our algorithm and the importance of a more specialized algorithm for periodicity counting.

5.4.3 Quantitative Results

Table 5.2 shows the results compared to other supervised and unsupervised methods. On QUVA, our model has the best MAE and OBOA of all the unsupervised methods. This is achieved with no prior bias or complex model, which demonstrates the efficiency of our framework. Moreover, even compared to supervised models, it is outperformed by only one model with a small margin.

Regarding Countix, we would like to highlight a few major weaknesses of the dataset. First, many clips with only 2 repetitions are cutting out parts of the periodic actions (at the start or the end of the video), resulting in no fully repeated movement. Moreover, the shortest video is 0.2s, which corresponds to 6 frames at 30 fps. In our opinion, such video clips are too short to contain distinct repetitions. In addition, the choice to keep the same train/validation/test splits as originally in Kinetics seems questionable, each action category being represented in both the train/validation set and test sets. To create a more context agnostic dataset, it would be preferable to have specific test categories missing from the train/validation split to challenge the generalisation of the method. On Countix,

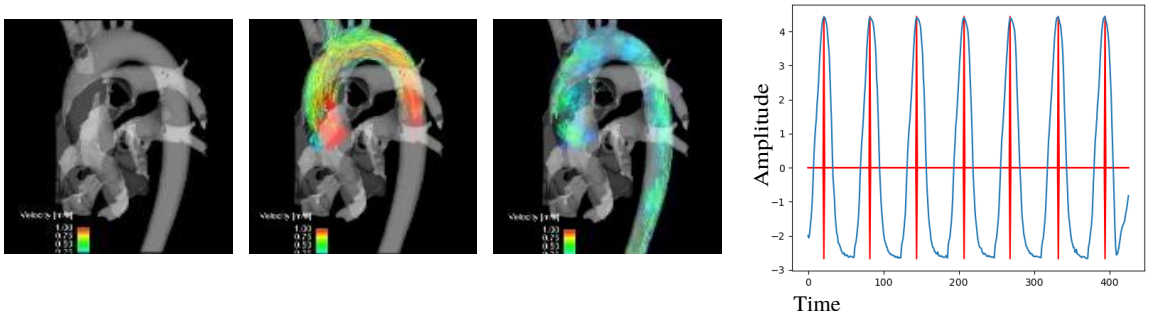


Figure 5.6 – 4D MRI video analyzed by our method. This is a proof of concept of the method's generalisation to different input types. Left: 2D slices of 3D input images (for display purpose) at different moments. The blood pulses through the artery. Right: the 1D PCA (blue) and peak detection of our model (red). As MRI contain very little noise, the periodic pattern is perfectly smooth. Better seen in color.

our unsupervised method gives an OBOA better than Zhang *et al.* [222] and is only outperformed by Dwibedi *et al.* [56]. The MAE is slightly worse than the supervised methods, but not by a big margin. In fact, the difference between our score and Dwibedi *et al.*'s equals the difference between them and Zhang *et al.*.

In addition, we observed a behavior in most of the "OBOA failure" cases (*i.e.* where $|c_i - \hat{c}_i| \geq 2$). Our *Max Detector* sometimes counts 2 repetitions instead of 1 for each cyclic pattern, therefore doubling the prediction compared to the ground truth. Indeed, a lot of ambiguity in the cycles count exist, the most usual being the "double action" that can be counted as either one or two periods. For instance, on a freestyle swimming clip, the annotated ground truth cycle can either be one "left and right arm movement" or only one "arm movement" depending on the labeller. Such ambiguity can often not be managed by context-agnostic methods, which will "guess" the answer between N and $2 \times N$ cycles when it occurs. This partly explains the difference between our score and supervised method's score, which are specifically trained to correctly choose in these ambiguous contexts. This problem artificially increases the MAE in an "unsymmetrical" way. If the truth is 10 repetitions, but our model gives 5, $MAE = 0.5$. If it is the opposite, $MAE = 2$. We could use the Normed MAE (NMAE) as a new metric, as it does not cause this "unsymmetrical" issue:

$$NMAE = \frac{1}{N} \sum_{i}^{N} \frac{|c_i - \hat{c}_i|}{max(c_i, \hat{c}_i)}$$

On QUVA and Countix, the NMAE of our method is respectively 0.158 and 0.345.

5.4.4 Application to 4D videos

Many applications in medical imaging deeply rely on 4D videos (*i.e.* 3D images through time), acquired with Magnetic Resonance Imaging (MRI) for instance. However, state-of-the-art periodicity counting methods cannot analyze them as their model can only input regular videos with 2D images. They could circumvent the problem by individually processing each 2D slice of the 3D images, but in doing so contextual data is lost and many model inferences would be required. In the end, one count per slice would be obtained and further post-processing methods would be needed to determine the final result.

On the other hand, our method can perform 4D video analysis with no loss of context, as the model is created with the data itself. Adapting the CNN architecture is straightforward in this case: the 2D convolutions are replaced by 3D convolutions. The remaining training method is unchanged and the results obtained by our approach are as good as for conventional videos. Figure 5.6 gives an example of a 4D MRI video, from the results of [180], showing a beating heart. The 1D signal obtained by our method is extremely smooth and easy to interpret. Although further quantitative evaluation would need to be done, these promising results represent a proof of concept that the method is able to generalize well to other types of data.

5.5 Going Further with Supervision

In our first discussions with them, race summary sheets were used as examples of what they expected, where the number of cycles per 50m was a metric. This metric enabled the coaches to grasp a general quality of swimming. It allowed studying the periodicity evolution during a long race (e.g.: 1500m) or to compare swimmers. However, coaches also want the exact beginning and end of each cycle. This allows them to measure how the distance per cycle evolves through a pool length, to identify possible local problems. Our periodicity counting method was not tailor-made for this finer-grained metric.

Coaches define the extremities of a cycle by a position of the swimmer, which depends on the style. For breaststroke, it is when the head is at its highest, for the others it is when the right arm enters the water (or both arms for butterfly). These are conventions, which cannot be found in the raw data. As such, they require some sort of supervision.

5.5.1 Supervised Training

The supervised method proposed relies on the same basics as the previous one: the input is a cropped video of one swimmer, and the output is a 1D temporal

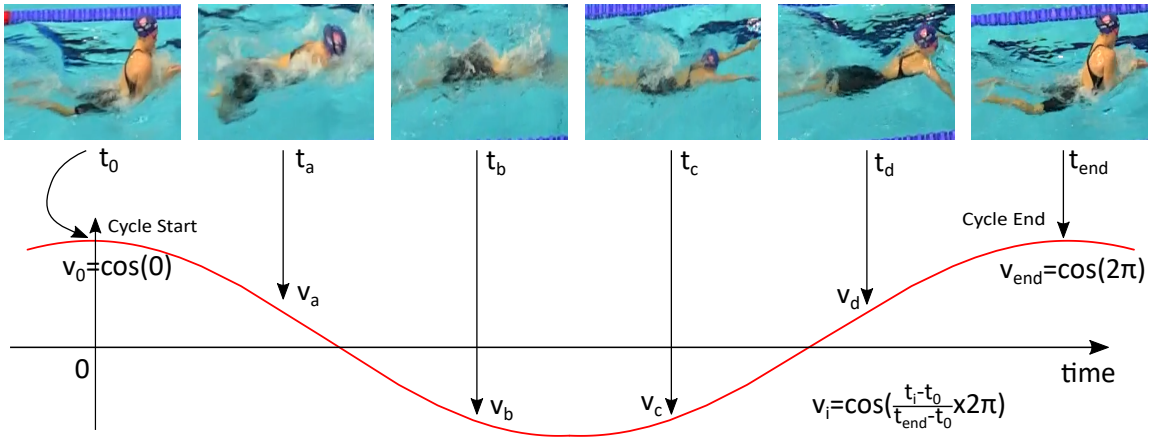


Figure 5.7 – Value associated to each frame according to their phase in the cycle. The arrow point at the value $v \in [-1, 1[$ taken by the frame on the sinusoid. Both extremity frames are associated to 1.

signal. It relies on a dataset composed of swimmers cropped, associated with their "instantaneous phase". To compute this phase, we label each image where a cycle ends and associate them with the value 1. The intermediate frames are associated with the values of a sinusoid between successive cycle ends, as explained in Figure 5.7.

The trained model inputs a frame and learns to output the associated value. The exact input can either be a single frame or two successive frames, or one frame and the optic flow, as for the unsupervised method. The resulting sinusoid is then analysed using the *Max Detector* algorithm. However, as the new model focuses on swimmers, heuristics can be used. Cycles are bounded by biophysical limitations and have a duration of around one second. In our dataset, the extreme values are 0.92 Hz and 1.09 Hz. Therefore, we limit the search for a frequency maximum to a minimum of 0.85 Hz and a maximum of 1.15 Hz to give a margin. This prevents the algorithm from finding out-of-bounds false positives.

The sources from this dataset are competitions which were filmed by our team. As coaches analyzed them and created the labels they want to automate from their videos, it was possible to use these data to our benefit. Therefore, this required no extra annotation and as long as the Fédération Française de Natation (French Swimming Federation) (FFN) keeps analyzing videos filmed by our team, the dataset keeps growing.

However, the number of championships filmed is fairly small (6 during the writing of this manuscript). Although dozens of races of each swimming style can feed the dataset, they are not very varied, which is a critical flaw, limiting the generalization ability of the model. The camera point of view and the lighting conditions are also very similar in the videos, reducing the domain representation.

The videos have varied framerate between 25 and 60. In both case, the proportion of frames associated with the value 1 is under-represented, despite arguably

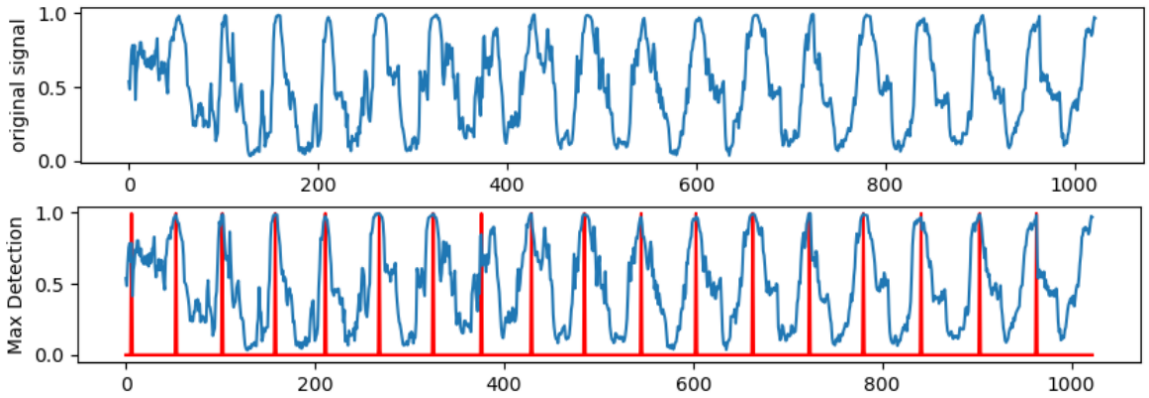


Figure 5.8 – Crops of a race around one swimmer, analyzed by our periodicity regression model. Top: the raw output signal. Bottom, maxima detection by the *Max Detector* algorithm. As maxima correspond to cycles extremities, the algorithm outputs the cycles extremities' timecodes.

being the most important. Indeed, the final goal is to detect the extremities of the cycles, where the value is equal to 1. As such, confusing intermediate values is less problematic than confusing an extremity with an intermediate. This problem was addressed by over-representing extremities in the dataset. A third of the training images are extremities, and the other two-thirds are regularly spread intermediate images.

5.5.2 Qualitative results

As this work started at the end of the thesis, only a proof of concept has been realized. The periodicity regression model is made of 8 sequential convolution layers with 8 up to 64 filters. This simple model was chosen to increase the training speed on the one hand, and also because the used training data was limited due to time constraints during the implementation of the idea. This model is also close to the one used with the unsupervised method (few straightforward convolution layers). We trained with videos from only 1 competition and tested on a race from another one in another pool. Result is shown in Figure 5.8.

Note that we moved values range from $v \in [-1, 1]$ to $v \in [0, 1]$, which has no impact on the end result. The curves are not perfect sigmoids, but a clear periodic pattern emerges with distinct peaks at 1. The first detected peak corresponds to a diving phase, where no cycle has started yet; this is a false positive one can threshold out by only starting the analysis after the 15m, where the swimmer must have started swimming. Visual examination showcase that the frames identified by *Max Detector* correspond to cycles edges. Quantitative metrics have not been used as the number of examples is too limited and this is just a proof of concept.

The results of this preliminary work are encouraging and suggest we are heading towards a good direction.

5.6 Conclusion

We introduced a framework to count repetitions in periodic videos. This method is outside of the usual training set - validation set - testing set paradigm, as the training is unsupervised and directly done on the test data. We believe that such an unsupervised approach may be of increasing importance in the future for different applications, in order to reduce the need for big datasets and complex architectures.

5.6.0.1 Good Unsupervised Results

Despite being unsupervised and based on a shallow model, our method gives results comparable to state-of-the-art supervised techniques with complex architectures. Due to its nature, it can work on any kind of video, even the ones that differ considerably from daily life (aeronautics, medical, astrophysics etc.). Moreover, with appropriate NN architecture, it can also perform well on other temporal data, such as 4D videos, biological sensors, and audio.

5.6.0.2 Promising Supervised Results

The supervised method shows promising results too to separate cycles when the swimmers are in precise given positions. From a more applied point of view, this is hopeful regarding the automatic swimming race analysis tool. The in progress creation of the supervised periodicity dataset is also encouraging as it will lead to better and better results.

5.6.0.3 Important Limitation to Address

Periodicity counting in the case of swimming, however has an important limit that cannot be addressed by any of the two described methods. It is the dependency on swimmers detection. If the detection is imprecise, there is another level of difficulty added to the task. A shaky detection might introduce periodicity in the placement of the swimmers inside the cropped videos, which would not depend on their position. The supervised method could address it by applying extensive shift-based data augmentation, but by artificially increasing the problem's difficulty, a bigger model may be needed. Also, a detection system centered on the backside of swimmers might give crops with undefinable style and periodicity. Likewise if the detection is too wide to the point of framing

multiple swimmers at once, in which case our periodicity counting method would be lost.

The detection system responsible for the crops extraction must be particularly smooth. This might mean not always centering the same point (the head), which can be subject to rapid local translations. As such, the detection method for crops extraction must be a bit different from the one responsible of measuring the instant position of swimmers in the image. The first might be drawn from the other using heuristics such as temporal smoothing to improve the method's results on the task.

Another limitation of this solution is the need to extract crops from the video. With 8 swimmers in the race, there must be 8 sub-videos generated. Cropping and resizing are two long task, computation-wise. In fact it is one order of magnitude slower than a UNet model inference. As such, periodicity counting based on crops extraction is the temporal bottleneck of the pipeline.

5.6.0.4 Possible Solution

All the methods referenced in this chapter, both ours and the ones from the related works, do the same strong hypothesis: there is only one periodic element in the video. This causes the two limitations previously mentioned (important dependency on detection and slow cropping process). A method inputting directly the uncropped swimming race video and outputting periodicity information for each swimmers would therefore address the two limitations at once. Although we did not work on this idea, one could imagine a solution based on (yet another) Unet-based model, where each area containing a swimmer would output the swimming phase. The output could either be a value between 0 and one (phase regression) or there could also be multiple channels, each associated to a range of phases (phase classification). We do not have the data to train a model based on this method, but if the supervised periodicity dataset develops enough, that might become the case.

CHAPTER 6

SWIMMING RACE AUTOMATIC ANALYSIS

Contents

Chapter abstract

This chapter unifies the different methods developed in this thesis. Methods to improve the performance of individual tasks using models for other tasks are explained. As such, the resulting framework performs better detection, registration and periodicity counting than the models that compose it. The result will be discussed, addressing its limits and the challenges it did not tackle, with examples and illustrations. This will be studied in the context of practical application of race analysis after placing a camera on the poolside. The objective of this chapter is also to interpret the metrics described before with respect to the task at hand and to propose more adapted ones. This enables a better interpretation of the results, avoiding over-reliance on the raw metrics number.

6.1 Introduction

The objective of this work is to create an automated tool able to analyze swimming races with little human intervention. The tasks required to perform such analysis have guided the thesis, resulting in (i) swimmers detection in chapter 3, (ii) pool registration/camera calibration in chapter 4, and (iii) periodicity counting and strokes detection in chapter 5. These methods were explained independently as their development and following contributions were separate. In this chapter, they will be unified under one framework performing the wanted automatic swimming race analysis.

This framework is more optimized than simply applying the 3 models sequentially. A result from one can improve or deteriorate the result of another. As such, it is important to tackle this merging task with a global view to maximize positive interactions and avoid negative ones. Further optimizations will also be discussed to improve the framework in future works. There also exist other (simpler) challenges required to perform race analysis that have not been addressed in this thesis. An existing solution will be explained for the ones that were studied and outlines of ideas will be given for the others.

The final method will be analyzed through different lenses of the swimming domain and the live race analysis. Video acquisition conditions will be discussed, as it is a crucial point for the framework's performance. Examples and illustrations will explain what results to expect in function of camera angle and placement and general filming conditions.

Also, a model giving the best result for a given metric might not be optimal for a specific application. The final part of this chapter thus puts in perspective the different metrics used in the previous chapters. Their meaning and impact will be explained with respect to swimming competitions. We also discuss new metrics better describing reality instead of other abstract data.

The contributions of this chapter are

- the framework: its design, its performance, and its limitations.
- an explanation of other side-algorithms (start detection, video synchronization, etc.)
- a discussion on the relevance of classical benchmark techniques of Computer Vision (CV) in the context of an application.
- suggestions of new metrics specific to swimming race analysis.

6.2 Framework

This section explains the design choices leading to the framework's architecture. Some limitations present in individual models seen previously are compensated by the tool's design. This is explained in the section. We also use elements not

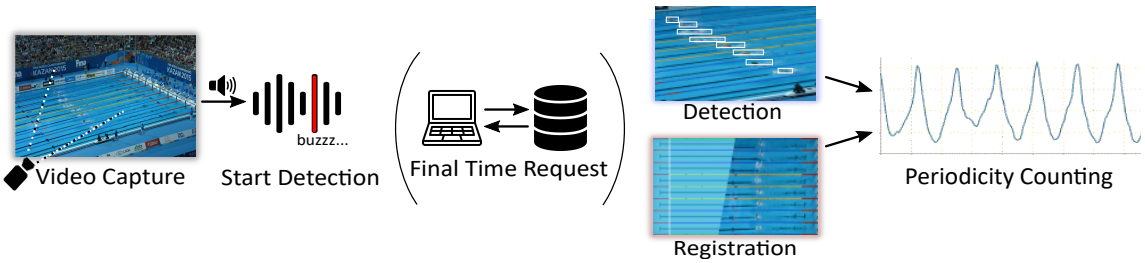


Figure 6.1 – A more complete illustration of our swimming race automatic analysis pipeline. The Final Time Request is between parenthesis as it can only be asked sometimes after the race. This, along with start detection, was not mentioned in previous chapters.

tackled in previous chapters. We first describe the framework's design assuming they exist and we later describe them.

6.2.1 Framework's Description

Figure 6.1 illustrates the framework. This illustration is different from previous sections as it details less the main contributions of the thesis to give a more global view of the operations. A lot of ideas described in this section have not been explored, despite being considered superior to what actually exists. The majority of them emerged around the end of the thesis, when a global view was possible and when we knew the possibilities and limits of our models. They are left to future works.

6.2.1.1 Pre-Processing

First, the race is filmed by cameras we placed next to the pool. Although the models work with other video flows (TV flows mainly), we chose to use our own to have more control over it. The capture system is composed of two static cameras, each filming half of the pool. This choice has been made at a time no work on registration had started. Now, it could be possible to use one tracking camera instead, but we prefer to keep the system unchanged for now, for robustness reasons (see section 6.2.1.2).

Using the audio of the two videos, we temporally synchronize them (see section 6.2.2.1). We then use our start detection system (see section 6.2.2.2) to know when the race starts. If the analysis is made some time after the race, we can also get the swimmers' finish time (see section 6.2.2.3). The slowest swimmer's time gives the end of the race on the video. This allows us to precisely segment the video and discard what is not part of the race (before and after). If on the other hand the analysis is performed live, we do not cut out the end of the video, thus the subsequent models study it until it ends. In both cases, the start is detected,

which results in a shorter video. We give a one-second margin to avoid missing anything.

The video is then resized to (256×256) pixels, which takes some time depending on the original size. This new size corresponds to optimal results for both registration and detection. Generating the heatmaps for these tasks can be done in either order. However, the detection pipeline relies on registration, so we start by explaining registration, for clarity.

6.2.1.2 Registration

As we use static videos, registration can be made on only a few frames to accelerate the process. As a result, there is one group of slightly different homography matrices (good regression) and several significantly smaller groups of very different matrices (failure cases). As there are many ways to fail but only one way to succeed (see section 6.2.3.1 and Figure 6.3), the most numerous group corresponds to the correct matrix. Within this group, we average each parameter to get the optimal homography matrix.

We also precise that, as we use static videos, an extra module could be added to judge the quality of the result. If it is not satisfying enough according to its criterion, the module could warn a human user and ask them to perform manual registration. It can be a simple model fed with top-views videos, trained to output 1 or 0 whether the registration is precise or not. The criterion is simply a threshold on the output. Such model was not developed but could be useful to set up in future works.

The matrices clustering is made using their 8 values (we exclude the value at position (3, 3) as it is always equal to 1). This is not optimal as this results in 8 different sets of clusters. If seven elements of two matrices are similar, but the last one is not, the resulting top-views will be significantly different, so the matrices should not be considered close. One could instead imagine a "whole matrix" clustering method to avoid splitting them into 8 independent clusters. For this, a unique matrix descriptor should be used, with closer values if matrices are *globally* closer, and more distant values if not. For that, it is possible to create an unsupervised embedding vector of matrices. One could for instance use a Variational Autoencoder (VAE) with a bottleneck of fewer than 8 dimensions to force the model to learn a good representation. The encoder could then create vectors for each homography matrix of the static video, and perform clustering on them instead of the individual parameters. Such model does not need to be very deep as there are only 8 inputs and outputs, so this would not slow down the framework significantly.

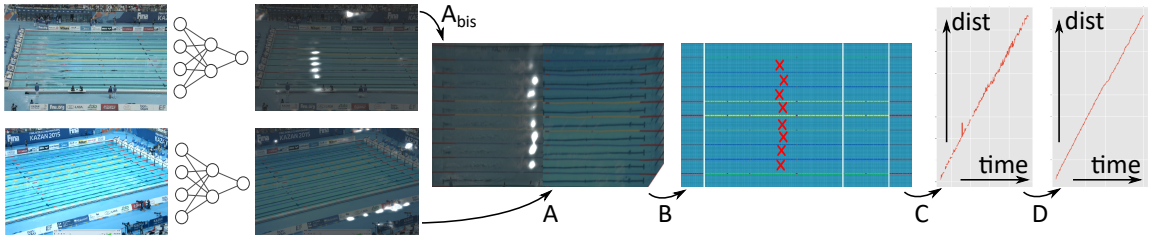


Figure 6.2 – The detection pipeline, optimized for race analysis. After the model inference on the right and left videos, there are 4 steps. A: top-view projection and fusion of the raw heatmaps. B: by-lane threshold, resulting in a position for each swimmer for a given frame. C: temporal aggregation of the positions through the entire race. D: position smoothing and gap-filling giving the position of each swimmer through time. Note that the topmost swimmer has a less intense blob. Its probability maximum is under 0.45, but our by-lane threshold definition allows us to detect it. The false positives, mostly detected in the right image outside of the pool, are neglected by the post-registration blob detection of step B.

6.2.1.3 Detection

Regarding detection, we use here a detection pipeline, illustrated in Figure 6.2, that is different from what is in chapter 3. The model generates heatmaps of swimmers probability from the images, but no threshold is applied yet. This is done for the entire video. Once complete, we use the registration result, as it significantly improves the detection performance. Indeed, using the homography matrix, the detection heatmaps are projected in top-view. The two views from the two cameras are merged, resulting in only one detection heatmap covering the entire pool. This allows us to segment each swimming lane, where we know a swimmer belongs. We find the highest value V on each heatmap's lane and determine a threshold from it, equal to $min(0.80 \times V, 0.45)$. Indeed, with a higher threshold than 0.45, we shown in chapter 3 that Average Precision (AP) and Average Recall (AR) decrease. However, we also showed that the farthest lanes from the camera give smaller swimmers probabilities, thus a lower threshold is necessary.

Once this thresholding operation is complete, it is possible that several blobs are identified on one lane, meaning there are false positives. We rely on blob size to discriminate them, the biggest blob being kept. This gives the position of the swimmers in a lane.

We previously mentioned the possibility to use a second-stage detector to confirm that a given blob is a swimmer. It could be used here to select the best blob of a lane. The blob with the highest probability according to this discriminator would be kept. However, this would increase computation time significantly. Indeed, the area should be extracted and resized, the model should

be fed the corresponding result, and all of this should be repeated for each candidate blob. In practice, the size heuristics works fine and is very fast.

With a swimmer detected on each frame in each lane, one can study their position through time. If a lane is actually not used in a race (which is frequent), then the biggest blobs' evolution of position through time will not correspond to a swimmer. They are easily discarded. If the analysis is performed some time after the race, it is also possible to know which lane is empty and not study it (see section 6.2.2.3), avoiding useless operations. If a swimmer is present in the lane, smoothing operations are used to remove outliers (*i.e.* incorrect or imprecise detections) and remove noisy accelerations. This results in a monotonously growing curve, corresponding to the position of the swimmer through time. Outlier-caused gaps are filled in by linear interpolation: on short distances, the speed of a swimmer does not change too significantly.

6.2.1.4 Periodicity

Finally, to perform periodicity detection, we must first crop a sub-video around each swimmer. To avoid warping artefacts coming from top-view projection, the sub-video comes from the original video. We project back the position of the swimmers in it (using the inverted homography matrix). If possible (*i.e.* depending on whether it was kept or not), the extraction is made from the video of the original size, before it was resized to (256×256) pixels. The swimmer's phase estimation model is used on the resulting cropped video. As the detection was smoothed during the detection phase, there should be no video shaking problems deteriorating the quality of the phase estimation. It results in the cyclic curve that we analyse with the *Max Detector* algorithm to separate and count the swimming phases.

6.2.2 New Elements

In the framework's pre-processing, we mentioned several elements that were not explained in previous chapters. As they are less related to CV and generally less complicated, we did not estimate they require an entire chapter. They are detailed here instead.

6.2.2.1 Videos Synchronization

With the current framework, the races are filmed with two cameras, each framing half of the pool. We need to synchronize them in order to merge their view. If not, it is not possible to do a temporal analysis of the race.

This task is straightforward and relies on the videos' audio. As the two cameras are placed next to each other, they record a very similar soundtrack. We do a cor-

relation between the two audio signals. There is one distinct peak corresponding to the time shift of the videos. By removing the corresponding number of frames at the start of the "earliest" video, we synchronize them. The audio being at 44100 Hz and the videos at orders of magnitude smaller framerates (<100 Hz), there is no sampling problem.

6.2.2.2 Start Detection

Start detection is a major feature for multiple purposes. First, knowing when the race starts is absolutely required for race analysis. Without it, it is impossible to estimate the time a swimmer needs to finish the race. Start detection is also useful to cut out the first part of the video before the race starts. This reduces the detection and registration's total processing time by not feeding them irrelevant parts of the video.

To perform this task, our objective is to detect the start buzzer. We labelled 35 beeps from different races, in order to construct a small positive samples dataset. Our method is to use classification to compare each sound window to the beeps.

Although it is possible to use audio correlation, this method is not very robust. Correlation is intensity sensitive, and the volume increases greatly during the race, as coaches whistle to communicate with their swimmers, spectators cheer, and music is played. To alleviate the limitations of raw sound, we use an audio embedding model [78]. It was trained on sound classification and can be used as a general audio embedding model. It inputs small windows of the audio signal and outputs a unique vector which is robust to sound level. We precise it runs extremely fast even on CPU and is not a time-bottleneck of the framework.

We used it to perform a semi-supervised classification task. We create an embedding vector for each beep in our dataset. We do the same for the race audio recording, which is split into small time windows. The embedding vector of each window is chronologically stacked to create a new temporal signal. The vectors corresponding to each of the 35 labelled beeps are correlated to it. The maximum is kept and corresponds to the start buzzer. In practice, some pre-processing must be made to optimize the performance, the most important of them being a band-pass filter between 870 and 1100Hz, corresponding to the range of the beep with some margin.

This buzzer detection algorithm also has another application. If instead of creating one individual video per race (per camera), one wanted to film an entire day of competition, they will have to segment the races. As the resulting video would last hours, it can be better to do it automatically. The algorithm we presented here can be thus used. However, during an entire day of competition, the buzzer can also be used for something else than starting a race (test, crowd cheering, etc.). These would be false positives hurting the segmentation algorithm. As such, the buzzer detection can be further improved with a "diving splash

sound" detector relying on the same idea. One can then create a heuristic such as "a race starts if a beep and a splash happen successively in less than a second". This should work as an efficient race segmentation algorithm in long videos.

6.2.2.3 Finish Detection

During the explanation of the framework, several references have been made to whether the analysis is performed either live or "some time after the race". This is because we also developed a system to automatically obtain the online metadata of a race using the Fédération Française de Natation (French Swimming Federation) (FFN) website (a similar one can also be made to grasp results in the FINA website for international competition data). The information we thus obtain concerns the participants of the race, the nature of the race (style, step of the competition, gender, etc.), and the overall championship. As such, we have access to the swimming lane and the name of each swimmer in any given competition as soon as it is published online. We can also know which lane is empty and avoid analysing it.

This is not a CV based method. We currently have not worked on "finish detection" using CV to know the exact frame a swimmer touches the edge. However, as we can detect the start with a high level of precision, and also know the duration of a race for any given swimmer, we can create a small dataset of swimmers touching the edge, similar to our strokes dataset from Chapter 5. Without this data, we select the moment a swimmer stops to move forward as a heuristic for their finish. It is not frame-perfect and is not robust enough to precisely determine the winner in close races. It is however a good indicator to know the general speed of swimmers.

6.2.3 Design Discussions

The individual models have limits that have been addressed in their respective chapters. No model is 100% precise, but it is possible to reach better performance by combining the models. In this section, we discuss how this was achieved and the consequence this has on the framework.

6.2.3.1 Improvement from the Uses

The detection part is the most obvious example of how performance is improved using the models altogether. Its pipeline has been significantly changed compared to the original much simpler one. In chapter 3, we mentioned several limitations of the model:

- small swimmers bad detection
- blobs merging

- false positives

Using the model *as is* can only recreate these problems. In the original case, the model would not detect swimmers on some lanes, it would detect several on others, and we would lose time studying elements outside of the pool. The new pipeline reduces these problems. The small blobs were not detected when using a threshold of 0.45, therefore we determine the threshold using the detection probabilities of the individual lanes. The blobs of different swimmers tended to merge, so we now segment the image into different lanes to prevent this from happening. There were false positives of multiple sorts, mostly outside of the pool, so the registration only minds what is in the pool by focusing on the lines.

This is still not perfect though. It is possible that a buoy or a reflection is recognized as a swimmer with a higher probability than the swimmer of the same line, hiding them under the 80% of the maximum value threshold. In this case, there will be a false detection. Further, our heuristics on only keeping the bigger blob is not perfect. However, we consider it sufficient in the majority of cases we met, as the other high probability blobs tend to represent buoys and thus be small. The smoothing process also removes them efficiently.

Considering the registration, the metrics described in chapter 4 do not allow a good understanding of our model's quality. Although our results are close to State of the Art (SOTA), they do not describe how they can be exploited in real conditions. As long as the Intersection Over Union (IOU) metrics are not at 100%, this means the image is either shifted, rotated, or at a wrong scale. The lanes are thus not correctly placed and the position in the image has no exact meaning in the pool. Registration is a very difficult task and its results are not yet comparable to the precision of others such as detection. The benchmark datasets contain difficult images dragging down the test results. In spite of this, our model can be efficiently used if the filming happens in good conditions.

Using static videos changes the problem as it is now a voting challenge. Although the camera does not move, the registration algorithm will give different results in function of the input frame, as shown in Figure 6.3. This is likely due to small variations of the waves and reflections changing the local distribution of pixels. As our method is based on landmarks detection and identification, if too many landmarks change, the output matrix will change too. The assumption is that although many images can have a wrong registration, they tend to fail in different ways (see the bottom line of Figure 6.3). However, as there is only one way to be correct, we can find the optimal solution by simply selecting the most numerous group after clustering the homography matrices. This alleviates the weakness of the registration model.

Compared to our raw registration algorithm, this unique matrix regression per video shows no shaking, no unusable projection, no bad warp, and no left-right inversion. It is also easier to manually correct it than if there was one registration per frame. Good registration is even more critical considering detection relies on

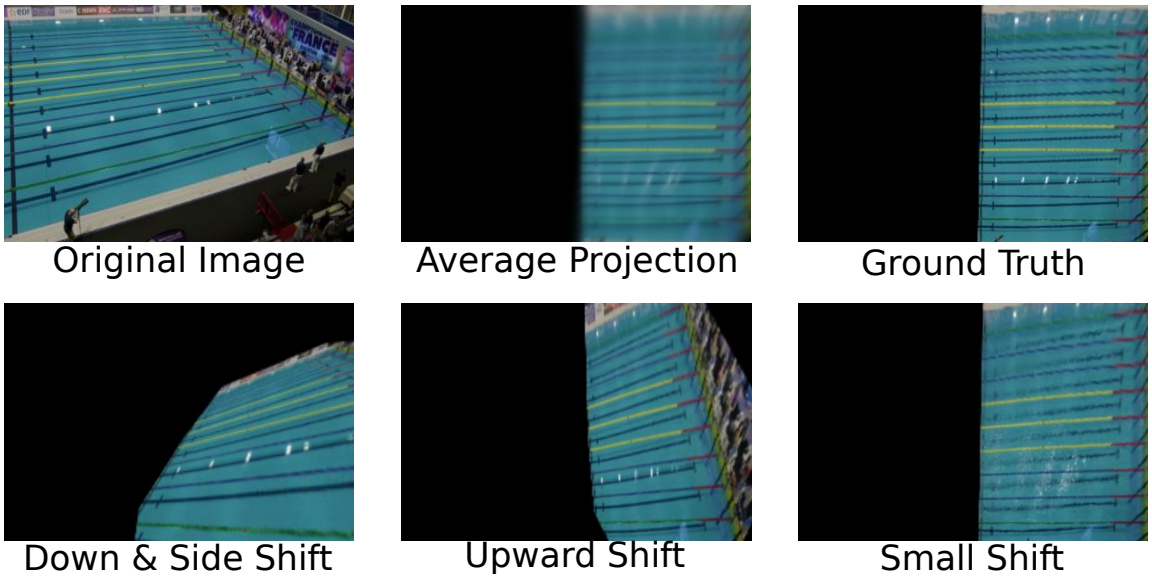


Figure 6.3 – Top-views obtained using the same model from a static camera filming a race. Different types of visible errors exist. There are also many small shifts, almost indistinguishable from the ground truth, causing flickering to the video. The average image of the projected video is sharper at the center than at the top and down: most small shifts are unstable at these parts. Both the raw projection and the median-based smoothed projection of the video are available at: https://drive.google.com/drive/folders/1oXKgDIzy3vTd0UHE9TaFjyozOeiR9gTt?usp=sharing.

it. Verifying its quality is important and manual intervention can be required if necessary.

Registration can also be improved by choosing an optimal camera placement. A wide view and a centered camera give better results. The orientation is important too, as Convolutional Neural Network (CNN) are very sensitive to it and rotation-based data augmentation during training does not always give better results. A more comprehensive study could be made to find the optimal point in a pool to place the camera, however competitions filming does not work like this: there is usually one unique area for filming (if there is one at all) and we must do with it. Therefore, the optimal solution is to frame half the pool with each camera without too much rotation so that the lines are horizontal on average.

Finally, one limitation of periodicity models is video shakiness, which can induce new undesired cycles. As detection and registration work together to smooth the position of the swimmers through time, the resulting videos are well-centered around them. The periodicity model thus has almost no external perturbation and can function optimally.

6.2.3.2 Speed Challenges

The time/performance trade-off is one of the most important in CV. We must position the tool we develop along this axis in function of our application and its constraints. With the swimming race analysis framework, we are asked to give a quick result after a race so that the athlete can inspect the video soon after they go out of the water. This application is not today feasible as the videos generally arrive to the framework and the processing computer long after the races. However, supposing it is possible in the close future, we can define our constraints according to it. The framework needs to process the video fast, but not at the point of real-time.

The current video acquisition process is not optimal. Cameras film different races and their memory cards are regularly switched with empty ones. The full cards are moved to the performance division cell and their content is uploaded to a computer. Then, either the computer does the analysis locally, or the videos are sent to an online server doing it. In both cases, there is a delay between a race and the moment the framework outputs results. This process was initially copied from the performance division which also films and analyzes videos. However, it is not adapted to our needs for quick feedback. A better way could be the setting up of live streams from the cameras to the computer. This way, the analysis could start during the race and be over soon after it. The delay would be removed, the only time constraint would be linked to the framework itself and no other external element.

A more technical aspect of speed limitation is related to the new pipeline for detection. Both detection heatmaps need to be projected in top-view to be further

Table 6.1 – The homography projection time and the detection model inference time in function of the input size, for the new detection pipeline. This was tested on 1700 images. "XX%" describes what percent of the pipeline the time corresponds to.

input size	projection time	model time	pre-processing
(128×128)	0.11s (8%)	0.58s (41%)	0.72s (51%)
(256×256)	0.42s (13%)	0.59s (18%)	2.3s (69%)
(512×512)	1.1s (11%)	0.59s (6%)	8.2s (83%)
(1024×1024)	2.0s (5.5%)	0.60s (1.7%)	40s (93%)

processed. Using the original pipeline, one would find the position on the image and then only project these positions in the top-view. Projecting an entire 2D array is slower than only projecting coordinates. Table 6.1 shows this for different sizes. Without heatmaps projection, the total time is reduced by the amount of the "projection time" column. For images of (256×256) pixels, the time increase is significant (13%, that is 0.42s for 1700 images) but not too important. This slowing-down is acceptable regarding the overall framework's speed and the performance gain this brings. For bigger images though, the pre-processing time being way more important, the projection time is less and less significant despite it being longer.

Races are shot in 4K, which is not an exact standard, but videos have a resolution close to (4000×2000) pixels. Figure 6.1, shows how pre-processing time increases with input image size. This takes into consideration the image loading in memory and resizing time, both increasing quadratically with image size. Inputting smaller images such as 720p would decrease substantially this time, thus improving the performance. Further, as the image is resized to (256×256) pixels anyway, it would not change the models' performance. For periodicity counting, videos cropped around a swimmer do not require 4K to perform well. If in the future other models for swimmer pose estimation are used, higher quality videos may be required. For now, it is not the case.

6.3 Limitations

We already mentioned some performance limitations of our framework. This section gives more details and context as to why they exist and what their consequences are. They will be illustrated for a better understanding. Solutions will be proposed to either circumvent them or remove them. Troubles related to the use of the framework and of the models will be mentioned. Unaddressed challenges also represent an important aspect of the framework's limitation, so they will be addressed too.

6.3.1 External Problems

Better use of the framework gives better results, but sometimes it is not possible to be optimal. Depending on the acquisition conditions, the videos can change drastically. This creates external problems, that are hardly solvable once the video has been made. The best solution is to avoid them.

6.3.1.1 Camera Position

The optimal placement of the camera is high above the pool, centered. However, it appears that in many events, placing the camera is not easily accessible. For French competitions, the performance division has no major problem, but acquiring international championships changes everything. It is not rare that video acquisition from several countries must share a small space following the rule of first-come, first-served. Finding a good position is then more difficult - or impossible. This results in sub-optimal analyses by the framework. Figure 6.4 shows some bad examples. For position detection, the best is to be centered and zoomed, contrarily to image C. This way, when the swimmers are on the opposite side of the pool, they are still visible. At least, a zoom is required to avoid simply missing them because they are too small. Concerning registration, it is generally better to be high above the water (contrarily to C) to have enough global context (contrarily to A, too much zoomed-in). Being too low tends to increase landmarks occlusion both by the swimmers and the buoy lanes.

A conundrum can be inferred from this: it is better to be zoomed in to perform better detection, but registration requires less zoom for better performance. However, this problem is not intractable: there is a range of levels of zoom giving a good performance for both, as illustrated in Figure 6.4, image D. However, such positioning is difficult to find during international events. In the worst conditions such as Figure 6.4 C, the best solution is to zoom more, in order to have good detection and then rely on manual registration.

Finally, occlusion can always be a problem and there is no good solution for it. If part of the image is hidden, part of the result will be meaningless. Models can learn some robustness to this but they have their limitations. To avoid it, placing the camera either high or in the first row in the stands is the best solution.

6.3.1.2 Lighting Conditions

Another external limitation concerning the environment of the race is the lighting conditions. If the camera is facing a bay window, there will probably be reflections on the pool. They can have different negative impacts on the framework. If too much light reflects on the water, this might saturate the camera sensors and only show white in some regions of the videos. In this case, light acts similarly to obfuscation as it hides the swimmers passing through the saturated area. It can

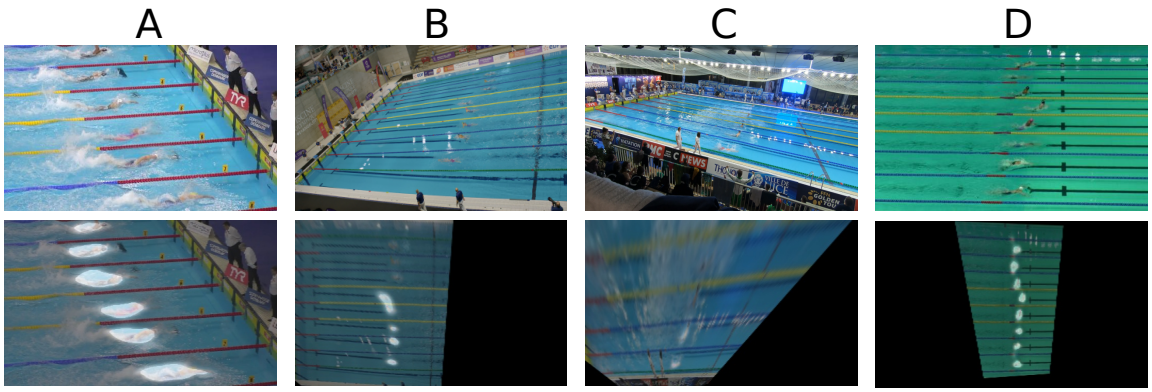


Figure 6.4 – Different camera positions and their result. The top row shows the original image, the bottom row shows the detection and registration. A: good detection but unusable registration (thus not shown) due to too much zoom. B: good registration but the detection performs poorly due to not enough zoom. C: both detection and registration perform badly due to not enough zoom and height. D: a good balance for both can be found with a high camera and an adequate zoom.

also hide the buoy's landmarks and give problems to the registration, although this can be compensated by the visible other landmarks.

Another problem of over-saturation comes with cameras in automatic mode. If too much light is present on the image, the ISO can decrease [139] or the aperture can close [140], changing the captured video. This can be a good response to ambient changes, but if the light source is outside of the pool, thus not an interesting part of the image, then it will deteriorate the video. The results can be a darker image, lower contrasts, and a smaller depth of field resulting in a blurry image. Each of these problems reduces the performance of the different models. To avoid this, the camera parameters can be selected prior to a race. However, as the light changes, the parameters might need some adjustment throughout the day. Automatic parameters can be used too, but they must be carefully chosen to only focus on the pool.

6.3.2 Internal Problems

The models do not have perfect accuracy, resulting in imprecisions in the framework. These limitations come from different elements, mainly their training dataset and task definition. These problems will be explained and illustrated. Elements of solution will be given, although a perfectly functioning solution would require further subsequent work.

6.3.2.1 Dataset Domain

Comparing the images from both RegiSwim[500]and *Swimm*[400], one can observe the images are taken from panning, zooming, and generally moving videos. On the other hand, the videos taken during competitions using our framework are static. These two types of videos have different properties, resulting in different domains. First, the pool is significantly more framed in moving videos: the camera-persons are able to frame exactly what they needed: the swimmers and the pool. As a result, the swimmers occupy a more centered place on the moving videos than on the static ones. The later videos have also the complex task of filming an entire half-pool at once. With enough height, this means shooting down to the pool and zooming in a bit to only frame the half pool, which is perfect. However, in most cases, the space is not sufficient and this results in complex angles with a significant part of the image not corresponding to the pool, not by choice but by constraints. These different viewpoints are illustrated in Figure 6.4: A and D are similar to the training datasets, while B and C have been filmed with a fix camera. The first couple is horizontally aligned to the lanes, the second is rotated to frame half the pool.

As periodicity detection functions with crops, the difference is lesser. However, this domain difference creates a significant drop in the performance for both detection and registration. In Figure 6.4 C, the camera is too low. The registration dataset does not feature such angle, and thus the model identifies poorly the different landmarks. Due to the same cause, the farthest swimmers are not big enough on the image and they are thus poorly detected.

To reduce the domain gap, several data augmentation techniques have been used, including but not reduced to image rotation, zooms in and out, and small warping perturbations (warping the image in too important manners reduces the performance). Data augmentation is a good tool for domain adaptation, but it is far from enough [187]. A better solution, that can be rapidly done, is to add images from the use case domain to the datasets.

6.3.2.2 Task Definition

Another problem regards the precision of detection. Using a method based on swimmers entire body detection will not exactly answer the problem asked by the FFN. Indeed, we currently define a swimmer position as the position of its corresponding blob's barycenter. However, depending on diverse parameters (threshold, training method, labelling choices, etc...), this barycenter can define different parts of the body. Thus, the detected points in the video can oscillate and flicker between frames. Further discussions with experts made us understand that a better detection system would rely on head detection, but the dataset was not built like this. An obvious solution would be to create a new detection dataset, where only the heads are annotated and thus detected by a trained model.

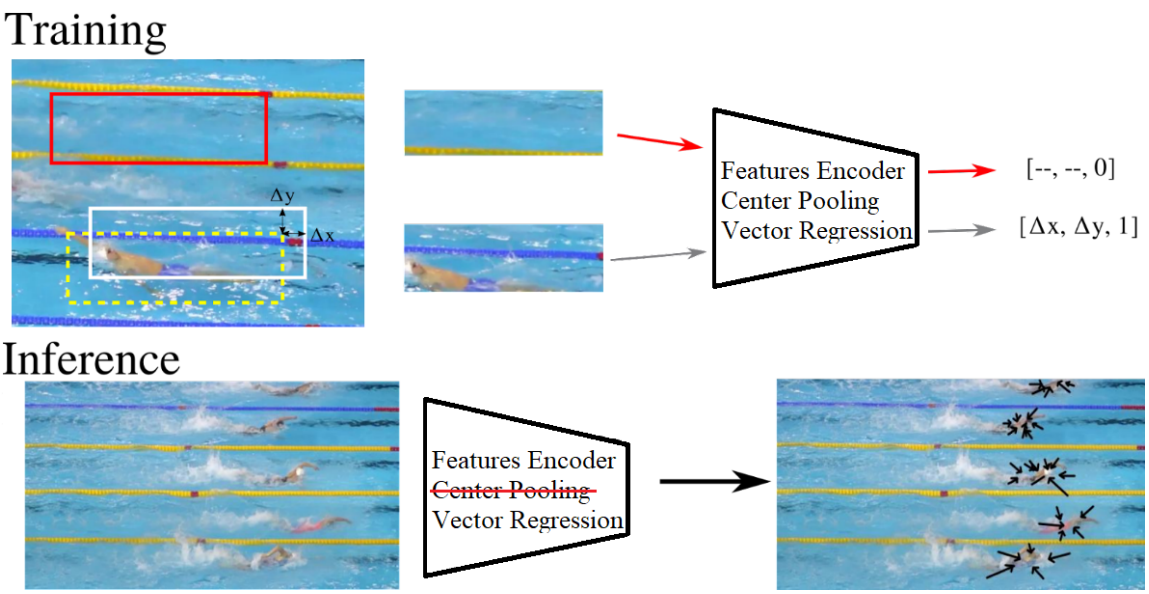


Figure 6.5 – Our model for swimmers temporally stable detection. On the training image, the ground truth box is the best-suited one. During training, negatives (red arrows and box) and shifted positives (grey arrows and white box) are alternatively fed to a model. For negatives, only the objectness score is learnt. "−−" means no back-propagation. For positives, the shift is learnt. During inference, the pooling is removed and a threshold on objectness is applied. This results in a field partial field of vectors pointing at a general bounding box center.

We also explored another solution to exploit our current full-body dataset. We extract a crop around a swimmer, centered on the center of the bounding box. The crop is then shifted by some pixels in the two main directions, and the shift model is saved as the ground truth label. We train a model, which input is the shifted crop around a swimmer, to regress the shift vector, not unlike [25] which detects precise body parts. The model contains an encoder, followed by a pooling of the output tensor's center element (*i.e.* center pooling). This element passes through a (1x1) convolution layer and outputs the shift vector, encoded following the box offset's encoding from [164]. The model also outputs a probability score defining whether there is a swimmer or not in the image. An explanation is given in Figure 6.5.

The objective is similar to training a model to estimate the labelled bounding boxes' barycenter using heatmaps, like what we did in chapter 3 to detect their entire body. However, this new task relies on regression instead of segmentation. Further, each labelled box is a piece of data that can be varied in an infinite amount of ways by changing the shift. On the other hand for swimmers segmentation, a piece of data is an entire image. Thus this new task has a lot more data to train a model on. As it also contains an objectness score and a voting system to aggregate the vectors, one can find more confidence in it than in the first method. The work

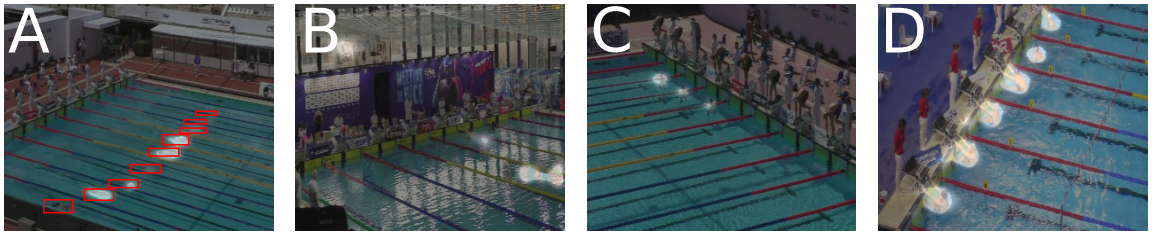


Figure 6.6 – The model's limitations regarding diving and underwater detection. A: some underwater swimmers are detected and others are not although both groups are visually similar. The bounding boxes are their ground truth position: they are narrower than the ones of not underwater swimmers. B: the dive is not detected at all. C: the swimmers on the starting blocks are not detected. D: the swimmers in backstroke start position are detected.

has not been finished, though, thus no comparison can be inferred. However, its final task is closer to what the FFN asked (estimation of one stable point) than the current model (blobs segmentation).

This method is similar to weakly-supervised learning [224, 128], as one task is used to actually tackle something else. In this case, the true objective is to learn the bounding boxes' center, on average. As the model is fully-convolutional, the center pooling can be removed so that instead of regressing a unique shift vector per crop, one can instead generate a vector field for an entire image. This results in a field of vectors, that are threshold according to the objectness score, only leaving the arrows close to swimmers. This work is not finished and needs more fundamental explanations. Intuitively, the difference of centers of the training boxes serves as regularization, allowing the model to get a representation of the average center.

6.3.3 Unaddressed Challenges

Apart from the limits of the models, there are also features that were never directly tackled during the thesis, despite being important for race analysis. Although most could be addressed without too much additional work, they are not currently part of the framework.

6.3.3.1 Diving

There are diving swimmers images present in the detection dataset. However, they are not very numerous and weighted in a particular way. As a result, swimmers tend to not be detected during their jump towards the water. This is problematic for early race automatic analysis. One could label the images which contain swimmers on the starting blocks and during their jump and give them

a special training weight to better allow their detection. An example of failed detection during the dive-in is given in Figure 6.6.

6.3.3.2 U-Turn

A similar problem arises for u-turns and underwater segments of the race (mostly before 15m). Underwater swimmers ground truth boxes are very narrow. This is illustrated in Figure 6.6 A, where the farthest swimmers are almost entirely hidden by the buoys. The heatmap-based model might not be the best suited for such narrow boxes, especially without particular case treatments. To improve their detection, one could artificially increase the height of bounding boxes that have a very low height/width ratio. Other solutions exist, based on specific data augmentation [115], zooms on image slices [4], edge-enhancement using Generative Adversarial Networks (GAN) **??**, or specific losses [203]. As this concerns only a specific use case of our application, we did not further work on this domain.

6.3.3.3 Swimming Segmentation

When a swimmer is detected, we define their barycenter, but we do not go further. Analysing its periodicity, we use a model that inputs a crop of a specific swimmer, but it only outputs one piece of information (the swimming phase). The same model does not study swimmers breathing, although it is very important data to measure. Likewise, we do not know the exact frame a swimmer touches the edge of the pool. A new feature, possibly using transfer learning based on the swimming periodicity model, could perform classification of "swimmer state". The states *i.e.* classes) can comprise breathing, touching the edge, the swimming style, and other aspects of a swimmer that can be extracted from a crop.

6.4 Computer Vision Metrics for Swimming Analysis

When studying the performance of a CV model, it is relevant to use certain metrics such as the AP, IOU, or the Off-By-One Accuracy (OBOA). They allow effective comparison of different models, as well as ablation studies for optimal hyperparameters estimation. However, the biggest number for a metric is not always relevant for a specific application. In this section, we put the CV metrics in perspective with swimming race analysis. This is first made by giving a definition of the metrics relative to this application. We further investigate new ways to compare and judge models in our specific context.

6.4.1 General Computer Vision Metrics

The 3 first chapters of this manuscript each rely mainly on one metric to evaluate the model they describe. We chose the best hyperparameters so that these metrics were as good as possible, but with the whole framework in mind, they can be different.

6.4.1.1 Average Precision(s)

Precision can be derived in several different metrics to measure almost anything. As it relies on the dichotomy between good and bad predictions, which must be adapted to bounding boxes. It was proposed in [39, 186, 60] to use IOU threshold between a found box and a true box to separate truth and error. A box is considered true for Precision-X if its IOU with a correct box is superior to X%. Averaging that on all the bounding boxes of a dataset gives the AP-X. Going further, the mean Average Precision (mAP) was later introduced [**coco**, 40] to refine AP for precise detections. It now averages the AP for different IOU thresholds between 50 and 95.

Our objective in chapter 3, was to find a good position of the swimmer, that can later be refined by detecting either a stable point, the head, or the foremost arm, depending on the needs. Further, we already explained how fuzzy the edges of a swimmer were because of the interaction of light at the surface of the water. Therefore, the exact location of the swimmer was not really tackled in this chapter. The extreme precision measured by mAP is absolutely not relevant here. This is why we choose AP-25, which is a low threshold compared to the usual ones in CV works (50 or 75). Using this threshold simply says whether a swimmer is here or not, no matter the exact bounds of its box. Feeding the found area to a refinement model can be made if the IOU of a given box overlaps by 25% or 50% with the truth, with no significant difference.

6.4.1.2 Intersection Over union

Intuitively, the IOU is easy to grasp. It represents how two areas overlap. In the context of pool registration, it measures how a registration overlaps with the ground truth pool. One can thus be satisfied by it as a simple yet efficient metric in this context: high IOU means good registration, and low means bad. However, in practice, humans are not very good at estimating the IOU between overlapping areas [176], which creates misplaced trust in the metric.

IOU does not explain how the result is different from the ground truth. In the context of pool registration, some errors are more impacting and consequential than others. In Figure 6.7, each registration has different degrees of precision according to the IOU metrics, but they can all be used to give the position of the swimmers in the pool. On the rightmost image (the worst one, according

Figure 6.7 – Illustration of visually assessing the IOU's difficulty. "w" stands for IOU_{whole}, "p" for IOU_{part}. Important variations of the first metric do not mean the second is impacted too much. The rightmost image is a case study of this phenomenon with a low IOU_{whole} but a decent IOU_{part}.

to both metrics), the more something is at the right, the bigger it is warped, due to horizontal warping. The swimmers are thus a bit deformed, but their positioning is almost perfect. On the other hand, vertical warping (see Figure 6.3, bottom-left image) artificially changes significantly the position of the top and bottom swimmers. Such vertical warping creates an outlier in the swimmers' position through time, while horizontal warping gives minor imprecision that can easily be flattened out. They have thus a very different impact on the framework, but an equivalent impact on the IOU metrics.

Regarding the two main metrics for registration, the IOU_{whole} is significantly less pertinent than IOU_{part}. It describes the entire pool, including parts outside of the frame, which is not interesting at all in our context. We want to analyse what is on the video, not what is outside of it. For non-specific domains, this metric gives a better appreciation of the registration, but it is pointless for positioning the (visible) swimmers in the pool. In Figure 6.7, the IOU_{part} of each image is beyond 0.90 which is acceptable for the framework, while the IOU_{whole} changes from 0.98 to 0.56 !

Finally, the IOU metrics for registration are not efficiently normalized. We mentioned that humans do not differentiate easily close IOU. During annotation, when performing ground truth visual verification, it was difficult to be sure the registration was perfect or simply "good enough" for our eyes. Therefore, the ground truth itself has major limits. Further, a model always returning the identity matrix for registration (*i.e.* a model that has no understanding of pool geometry), reaches an IOU_{part} of 0.29 on the Soccer WorldCup benchmark. As the maximum score is difficult to estimate and the minimum score is not 0, although the metric seems intuitive, it is actually not. Other methods must be used to grasp a better understanding of a registration model's quality.

6.4.1.3 Off-By-One Accuracy

The OBOA is not adapted to the specific needs of the FFN. Defining the task with them at the beginning of the thesis, we were given races summary sheets where the number of strokes repetitions during one pool length was important.

Exploring this direction, we researched periodicity counting on videos, where one of the most important metrics is the OBOA. However, what actually interested the FFN was the exact instant a stroke ends and another starts, according to the strokes definitions we gave in chapter 5.

Although it was misaligned with the actual goal, this topic made us focus on cropped videos around swimmers and allowed us to study how to embed them with little data. The OBOA metric and the work we did on the topic can both be directly used to fill in swimming race summary sheets. The further work described in Section 5.5 was enabled thanks to this research.

6.4.2 Swimming Analysis Metrics

The classic CV metrics not being perfectly suited for our specific task, one can propose new ones that could better represent the quality of different models for swimming race analysis.

6.4.2.1 Pool Registration

For registration, an option is to measure the distance between a set of points on the ground truth projected image and their position in the estimated projection. A similar metric, the projection error, was introduced in [38], where they propose to create a set of random points for this measure. One could simplify this concept by only using the corners of the field to measure the projection error.

Another approach would be to describe the deformation caused by the registration imperfection. Indeed, a simple number gives a general idea of the homography estimation quality but does not provide further details. Each matrix can be computed from 3 rotation parameters and 3 translation parameters. Comparing the difference between the estimated and true ones (in radiant or degree for the angles, in meters for the translations), would change the understanding of a registration. One would better grasp the nature of the error, what can be improved for a model, and it would also be possible to discard errors that do not impact the overall analysis, as mentioned in Section 6.4.1.2.

6.4.2.2 Swimmer Position

Regarding detection metrics, another approach would be to define a small set of points (the head and the hips for instance) and define as a truth a box containing exactly one instance of them. The AP can be used with this good prediction criterion instead of arbitrary IOU thresholds. This would allow boxes significantly bigger than the swimmer to be considered correct, but it would be compensated by a refinement step in a second time to exactly pace either the head, the hips, or the landmark of interest. This would require further work, as these points are

not labelled yet. Two points being faster to annotate than a box (which contains 4 points), this could have been another type of data used for training, following weakly supervised learning ideas [161, 10].

Finally, to really measure the quality of position estimation, one could simply compare the position of a swimmer obtained with their framework to the position manually annotated by experts. This is the only metric that really matters. Its problem is that it can only be computed once all the elements of a framework are functioning. It is not adapted to separated development of models, at least not before every brick outputs a satisfying result.

6.4.2.3 Periodicity

Regarding periodicity, to compensate for the limitations of the OBOA, another metric can be used. One could measure the number of frames between the detection of a cycle's end and the ground truth. Such metric could either be framerate-dependent or normalized by dividing by the number of Frames per Second (FPS). The Mean Squared Error (MSE) is perfectly suited for this evaluation task.

6.5 Conclusion

This chapter merged the previous ones into one functioning framework that specifically tackles swimming race analysis. For that, it changed the uses of the different models it is composed of. The detection pipeline is refashioned, the registration is only applied to a finite set of frames, and the periodicity is measured on smoothed cropped videos. All of that is to improve the framework's performance, although it partially comes at a time cost. However, this cost is acceptable compared to the advantages it provides.

The aspects that challenge the framework have also been explained. The models have important limitations, mainly regarding their precision. Combining the models can circumvent some, allowing for instance position smoothing through time to remove the outliers, or localized threshold avoiding false negatives, but some drawbacks remain. Different solutions have been proposed to improve them in future works. Increasing the size, diversity, and representativeness of the true use context of the different training datasets is also a promising, easy-to-execute solution.

Finally, the different metrics used to individually evaluate the models have been contextualized for swimming. Their meaning and limits have been explained so that their use can be clearer and one would not blindly rely on them. Other metrics were proposed too, to suit the specific swimming context.

CHAPTER 7

CONCLUSION

Contents

7.1 Contributions

This thesis introduced multiple contributions for swimming analysis. They bring different domains of Computer Vision (CV) together, like object detection / segmentation, registration, or unsupervised learning. Two datasets were also introduced to help the community moving forward and creating new methods.

7.1.0.1 Detection

The first contribution we made tackles swimmers detection in [103]. The *Swimm*^{400}dataset, also presented in the paper, contains 400 labelled images of swimmers with around 3100 bounding boxes. No publicly available swimmer detection dataset existed when was released. Different works [84, 208] already trained detection models on their own data, but they never published it. We hope that releasing ours publicly will encourage the others to do the same and push forward the sports analysis community. Further, the dataset is varied, containing several environment, pools, viewpoints, angles, etc. It can be used for swimmer detection without external constraints, such as camera positioning or type of pool.

Swimm^{400}is well-crafted, but it is orders of magnitude smaller than the usual detection datasets [**coco**, 48]. Consequentially, classic object detection techniques [163, 164, 165, 167] do not perform well on it. Usually, transfer learning is used to address a new type of object to detect. However, due to the unusual features present in pools, such method did not show good enough results. Swimmer detection is thus addressed using segmentation with bounding box input data. The model we used is a variation of the classic segmentation architecture UNet. Our tiny-UNet model is smaller and shallower, in order to manage the training set size.

To train it, we converted the bounding boxes into heatmaps by creating ellipses inscribed in the boxes. This heuristics worked efficiently to detect swimmers with a good precision. It is weakly-supervised learning (bounding boxes are at a lower level than segmentation) and it was shown that a rough segmentation of swimmers is possible with high enough input image resolution.

7.1.0.2 Registration

Our second contribution addresses automatic sports field registration techniques, in [104]. This task is also fundamental for swimming analysis, as once it is combined to swimmers detection, it allows the positioning of swimmers in the pool. Having an existing model for this task gives more adaptability to our analyses, as we do not have to rely exclusively on static videos and manually calibrate their view. As such, past videos can be analyzed, whereas they could not before due to their panning, zooming, or moving camera.

As the main registration benchmark, Soccer WorldCup, is not about swimming pools, we again introduced a dataset named RegiSwim^{500}. It contains 500 images of pools and their corresponding homography matrices to perform a top-view projection of the pool. Contrarily to the other benchmark, it contains very different levels of zooms and camera angles, making it more challenging. It further enables to train more adaptable models, which do not have to rely on specific filming properties.

The model relies on the UNet architecture to position landmarks on the image. Each is associated to a position in the pool, forming a pair of points. Detecting enough of them (dozens per image in practice) allows us to use a consensus algorithm to estimate the homography matrix. This consensus-based algorithm gives some robustness to the model which, contrarily to any other sports field registration method, do not rely on refinement. As such, it is both precise and fast, even without a state of the art Graphics Processing Unit (GPU).

7.1.0.3 Periodicity

In this thesis, we also studied periodicity counting and strokes end detection, in chapter 5 and in [105]. This method relies on unsupervised training to create a model that is able to embed a video to form a cyclic path in a latent space. Using other signal processing techniques, we can count the number of repetitions in a videos. The idea is to train a model on the test data, without label. Such approach is original and enables the creation of models specifically fitting a given type of input (videos, sequences...) and domain (swimmers, daily-life activities...).

Specifically for swimmers, we also trained a model for swimming cycles end's detection. It inputs a crop around a swimmer and outputs a value between 0 and 1 describing the swimmer phase. For our specific application, such model is very

important as it enables the coaches to evaluate their swimmers based on their stroke rate.

7.1.0.4 Framework

Finally, chapter 6 presented a complete tool for automatic swimming race analysis. It unifies the different models presented before and creates a synergy to improve them using either the others or the general context of analysis. The tool can be used by the Fédération Française de Natation (French Swimming Federation) (FFN) to automatically study a race that was filmed in the correct context. We hope it will further help the swimming community.

7.2 Limitations

The limitations of this thesis can directly be observed using the swimming race analysis framework, as it synthesizes the other works. Some design choices were made to address the problems of different models.

As such, the detection pipeline is now longer than the original one and depends on registration. If the detection model was better, or if it contained a second-step to refine its results, this new pipeline would not be required. Indeed, it tends to miss swimmers occupying a small area in the image. On a fix image, most humans would also miss them, but not on a video. As such, one limitation of the detection model is that it does not rely on temporal information (either past or future frames) to detect swimmers. Such approach would probably give significantly better results, as it is not rare that waves and diffraction occlude swimmers, making them almost impossible to detect using only one frame.

This temporal approach could also be used for registration. Some points of view lack information to allow robust and confident registration. However, using the landmarks visible in the previous frames would circumvent this problem. Registration needs more precision in order to be usable. We used tricks to make it work in our framework (static videos and voting for the best homography) but this adds up new constraints to the filming process. Freeing our tool from them would significantly increase its usability.

The final problem of the framework is that it would be more helpful for live competition analysis if the cameras filming races were streaming their video to the processing computer. It would allow direct feed-backs from the tool to the coaches and swimmers, but also from them to us, to improve the tool. Indeed, such faster speed to obtain results would encourage both wolds to interact more, which is beneficial for both.

7.3 Future Works

This work proposed different new methods, concepts models and datasets. As such, it unlocked some domains of swimming analysis, that can be explored in future works.

7.3.0.1 Automatic Swimming Race Analysis

First, this thesis can be the base for exciting new works to improve the automatic analysis of swimming races.

The models can generally be improved by labelling more images, specifically from the domain of fix videos as they are captured for the tool. New data are always appreciated by the community and can increase substantially the performance on different tasks. In particular, u-turns and underwater swimmers could be better detected with more context, as the waves they create hide them for a few frames. As we previously mentioned, using temporal data can improve the results in similar ways, making the models more robust. One should also change the architecture and algorithm, as the ones currently used do not handle such type of data. Similarly, a refinement step for swimmers detection can be added to improve the exact estimation of their position in the image and the pool. We proposed different ideas, such as shift vector regression or head annotation. Both seem promising.

Different challenges were never addressed in this thesis, such as breathing detection, swimming style classification, or finish detection (when the swimmer touches the pool edge at the end of a race). These tasks can be addressed by using a model that is fed a crop around the swimmer, and outputs probabilities for these diverse elements. Transfer learning can be considered for this task: the swimmer phase estimation model already inputs this data and has learnt a representation of the swimmer's general posture. Fine-tuning it can prove very efficient and give good results at small data costs.

Weakly supervised learning can also be applied to our framework. The model fed with crops of swimmers giving information on their phase could be extended with such method. Full images could be the new input, and the model would give phase information about only the areas with swimmers. If a model like this was possible, it would unify swimmer detection with periodicity analysis. This would increase the framework speed by merging different models and removing the costly crop operation.

Swimmers pose extraction can be studied. This was not tackled at all during this thesis, as the labelling task seemed overwhelming. However, based on our existing swimmers detection methods and on the progress in the domain, it might be feasible. However one must be realistic about the limitation of underwater joints' detection in uncontrolled environments. There likely are optimal camera

points of view for this task, but as we barely control the camera placement during swimming events, this will be a strong constraint.

7.3.0.2 Computer Vision with Few Data

The future of CV will be about small datasets. CV showed significant improvements during the past decade, mostly due to the better understanding of deep learning and the size expansion of datasets. However, facing new, out of domain challenges, CV must adapt to less data requirements or less labelling time. We think that the most promising methods concern none-fully-supervised learning, especially few-shot learning and weakly-supervised learning.

Few shot learning will generalize detection tasks. Our human brain is a few-shot learner [205], as we only need a couple of examples to already grasp a good understanding of a visual concept. It thus proves that few-shot learning is possible in any case with a significantly higher level of precision than what it currently reaches. Tackling this work by training a model to detect a wide range of classes to create a "general detector model", like in [62, 160] seems promising. Expanding the domain of the different classes can improve the generalization ability of the model. Currently, only daily-life objects are parts of these "wide" datasets (wide because they contain many classes but few examples per class). Including classes further away from this domain (swimmers, medical imaging objects, space objects, etc.), can improve domain generalization. New domains can also be artificially created from existing classes.

Weakly-supervised learning is very promising. Contrarily to few-shot, it can rely on massive amount of data, as they are extremely fast to label. One image of a given scene can be cut in smaller parts with the same label. Groups of different images that where gathered using search engine can all be labelled the same. A wide variety of techniques can be used to give classification labels to enormous amounts of data at a small cost. Weakly-supervised learning develops with the years and with enough innovation, it might reach the performance of fully-supervised techniques.

Vision Transformer architectures [52] might give better results for registration tasks. One could try using them instead of Convolutional Neural Network (CNN). Indeed, this task relies on landmarks that must be contextualized along with all the others, thus with the entire image. As transformer models are better than CNN to study the image globally, this could lead to better results. One could also use them to study the video as a sequence and introduce continuity in the analysis. Transformers, especially for vision, were developed a lot during the span of the thesis. If I were to restart it now, it is likely I would consider them. This architecture currently needs more data than CNN, though. This aspect would have to be handled in priority.

7.4 Conclusion

This thesis tackled the challenge of automatically analyzing swimming races using videos. It has to be generic enough to be used in competition situation, where one cannot place sensors on the swimmers and where even a camera has a constrained position.

To do so, we chose to focus on CV techniques, which seemed the most adapted. The general task was divided in 3: detection, registration, and periodicity counting. These task resulted in different models, each with its strengths and weaknesses. A framework unifying them all was finally presented to perform the general analysis task. It is currently being used by the FFN to help with analyses. We hope it can be extended in future works as there are very promising leads for improvement.