ÉCOLE
CENTRALE LYON

THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON
opérée au sein de
l'École Centrale de 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 Analysis of Elite Swimming Race Videos

Analyse Automatique de Vidéos de Course de Natation Elite

Soutenue publiquement le 1er Décembre 2022, devant le jury composé de :

Jenny Benois-Pineau, Pr.	Université Bordeaux 1	Rapporteure
Peter Sturm, DR	INRIA, Rhône-Alpes	Rapporteur
Serge Miguet, Pr.	Université Lyon 2	Examinateur
Albrecht Zimmermann, MCF	Université de Caen Normandie	Examinateur
Céline Leverrier, PhD	Fédération Française de Natation	Invitée
Stefan Duffner, MCF, HDR	INSA Lyon	Directeur
Romain Vuillemot, MCF	École Centrale de Lyon	Co-Directeur

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.

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.

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 compétences d'accompagnement diverses. 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 [1, 2]. 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 ma paresse en leur présence. Promis, c'est parce que je bossais.

Je remercie pour finir l'invité surprise de la thèse, le covid 19. Il a su significativement réduire mon impact carbone durant ces trois années 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
CNNs	Convolutional Neural Networks
CAM	Class Activation Mapping
CV	Computer Vision
DL	Deep Learning
DLT	Dynamic Linear Transform
DNN	Deep Neural Network
DoG	Difference of Gaussians
FFN	Fédération Française de Natation (French Swimming Federation)
FPS	Frames per Second
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
IOU	Intersection Over Union
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 Network
PCA	Principal Component Analysis
POV	Point of View
R-CNN	Region-Based Convolutional Networks
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

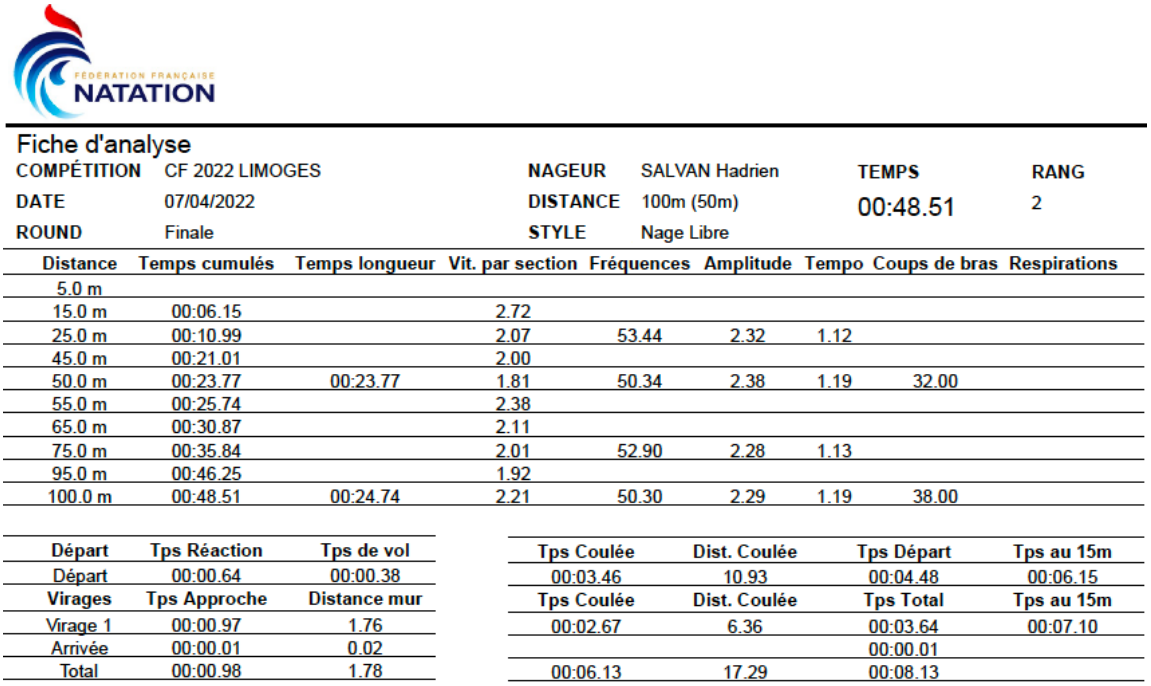
At the 2022 French Elite Swimming Championships in Limoges, after Maxime Grousset and Florent Manaudou finished in respectively first and second place in 50m butterfly, they were asked by a sports media [1] their feedback on the race. Maxime Grousset explained how he started strongly with great first 15m, then how his finish was under-optimal. Such analysis performed intuitively by a top-level swimmer (silver medal in 100m freestyle at the 2022 World championships) is extremely valuable. It allows him and his coach to know what is mastered and what are the areas of improvement.

Grousset explained his personal feelings about the race, but he could not have provided quantitative data evidence showing how ahead he was initially and how he slowed down at 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 sooner.

A quantitative analysis of the sort can only be performed by studying the entire video footage of the race. Coaches can do it manually using specific tools, but they tend to focus on the few metrics they prefer. Further, not all swimmers have either a coach to do it for them or the time and tools at their disposal. 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 had access to a summary sheet showing how they performed during a race, it could help them to progress rapidly.

This is the goal of this thesis: to create a swimming race automatic analysis method. It was made in collaboration with the Fédération Française de Natation

1. By Jonathan Cohen from the FFN https://www.youtube.com/watch?v=Qtr2VttunCQ&t=110



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

(French Swimming Federation) (FFN) through the Neptune project [2], which started in January 2020, just three months after this thesis.

1.1 Choice of Using Videos

We chose videos because coaches heavily rely on them. Cameras are transportable and can be installed next to a pool with no major requirement other than a dry area. They are also easy to manipulate even by non-experts due to their presence in our daily life. Videos are 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.

Apart from videos, it is also possible to analyze a race using other data streams (GPS, inertial sensors, *etc.*), but they are constraining. They need swimmers to either equip with intrusive sensors or to perform extensive prior calibration phases. Body sensors, for instance, are much more constraining although they

2. The Neptune project (Natation et Paranatation: Tous Unis pour Nos Elites) brings academics and swimming professionals together to prepare for the 2024 Paris Olympics. Members of the FFN explained what lacked from their current analyses and the researchers focused on these topics.

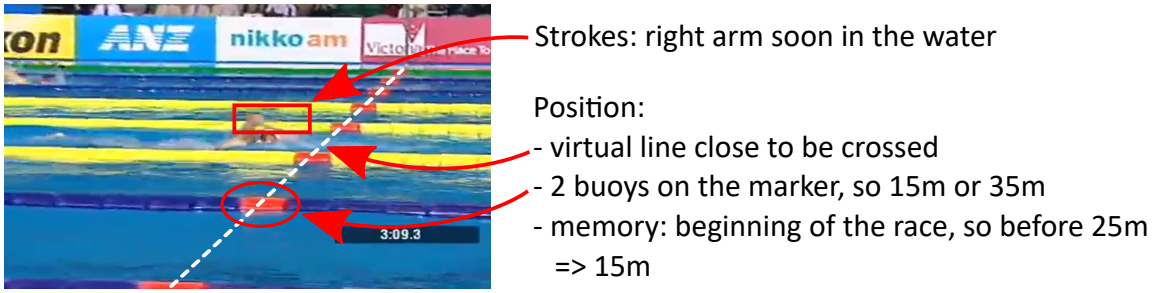


Figure 1.2 – How humans understand a pool and use their prior knowledge of the situation to infer data out of the video. These implicit challenges must explicitly tackled by CV techniques.

give a precise body position. Finally, intrusive gears are not allowed during competitions, contrary to filming material.

1.2 Manual Analysis VS Automatic Analysis

The FFN's performance division uses videos that are closely zoomed to the swimmers to fill in summary sheets, as shown in Figure 1.1. Their workflow is as follows: an expert informs the swimmers' position through time by telling when they cross each visible landmark. Their strokes are then counted during one pool length. This enables to automatically compute the other metrics in the summary sheet. Despite all being inferred from the same two values, they offer different perspectives of the race to coaches and swimmers.

Although these data contain a lot of valuable information, we argue that Computer Vision (CV) methods can enhance them. Also, this sheet is only created for 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. Finally, the bottleneck of these analyses is the human time they require. If our goal is achieved, we hope to save time for the performance team over their manual annotation process and analyze a wider range of swimmers (*e.g.* foreign races, local league competitions, *etc.*).

1.3 Computer Vision Tasks

To perform a task using CV, one must first understand the prior knowledge humans have regarding swimming and the spatial structure of a pool, as illustrated in Figure 1.2. 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. As CV algorithms do not have such prior knowledge,

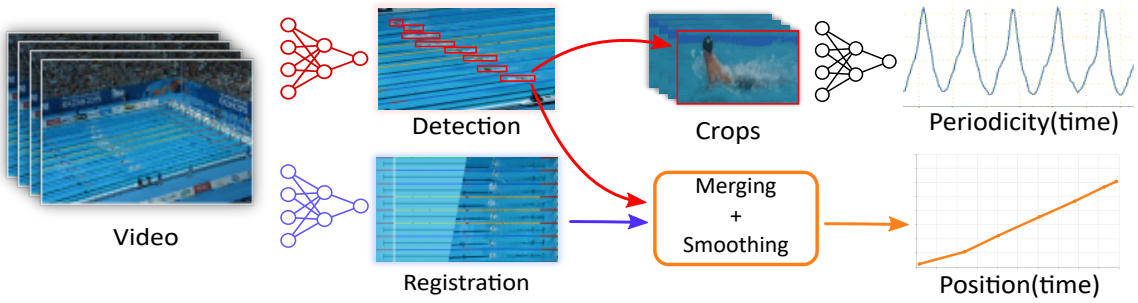


Figure 1.3 – Description of the automatic analysis method we propose. Each Neural Network (NN) model represents a different challenge of this thesis. The final objectives are to estimate the swimmers' position through time and periodicity.

they cannot study a race the same way humans do. An automatic swimming analysis must then do:

- **swimmer detection**: places the swimmers in the image.
- **video spatial calibration**: maps a position on the image to a position in the pool (*i.e.* removes the perspective).
- **periodicity estimation**: counts the number of swimming strokes a swimmer makes during a pool length, which is the coaches' main analysis metric.
- cameras temporal synchronization: if multiple cameras are used, we must adjust them so that the race starts at the same time in each.
- swimmer identification: maps a swimming lane with a swimmer.
- swimming phases segmentation: separates diving, normal swimming, and u-turn. Each has different properties and must be analyzed individually.

In this thesis, we focus on the tasks in bold, the others are left to future works, although leads are proposed in Future Works (section 6.3.7). We also aim at working with unconstrained swimming videos to be more general and less dependent on acquisition conditions. The models must thus adapt to videos that can be panning to follow the swimmers, fix to continuously film the same part of the pool, far away or close to the water, *etc.*.

The swimming race analysis method we propose is illustrated in Figure 1.3. It starts by detecting the swimmers, arguably its most important element. Once this is done, only half the swimmer's positioning part is done: depending on the camera position, the framing, and possible camera movements, mapping pixels from the image to positions in the pool is not trivial. A camera calibration step is necessary to resolve this. Combining detection and calibration gives the position of the swimmers *in the pool*. Finally, with each swimmer detected, one can crop a frame around the swimmers in the video and study the periodicity of these sub-videos.

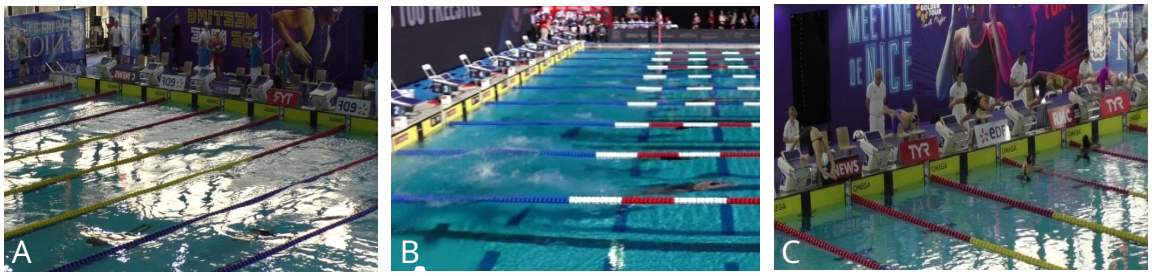


Figure 1.4 – Examples of difficult conditions in swimming race videos. Underwater swimmers are sometimes barely visible (A). Low camera angles give little view of the farthest swimming lanes (B). Swimmers of a previous race still in the pool and referees standing by the start create a difficult subjects of interest choice (C). Lighting conditions, like outdoor with bright sun (A), or indoor with back-lighting (B, C) can obfuscate swimmers.

1.4 New Challenges We Must Tackle

The CV works directly applied to swimming are rare and no pre-existing model can be used as an *ad hoc* solution. A new model specifically addressing these elements is thus necessary. A robust solution we use in this thesis is Deep Learning (DL). This type of algorithm relies on dedicated datasets, but none 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 [3, 4], these datasets do not contain examples of humans in the water. As a result, even powerful object detection models [5, 6, 7] able to detect any pedestrian, runner, or sitting person, are unable to robustly detect swimmers.

CV results also depend significantly on the video capture conditions. Difficult lighting, obfuscation, motion blur, and every other perturbation can cause important performance drops. Such problems exacerbate in swimming, as examples showcase it in Figure 1.4. 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.*. Camera placement is another problem, as they can be placed by the coaches on the poolside at different places depending on the presence of a public in the stands, the availability of a filming cell, *etc.*. Depending on the induced Point of View (POV), the video quality can vary significantly. A camera that is centered, positioned a few meters above the ground, and at a good distance from it, gives the best point of view.

These capture conditions must be best suited for the later analysis. Static cameras are easy to calibrate but are not zoomed into swimmers. The analysis is therefore more robust but limited by the digital 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 show 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 exact same way due to their very nature (either statics or following a group of swimmers). This thesis proposes solutions for both.

1.5 Contributions

Three articles have been produced during this PhD. The first, described in chapter 3, addresses swimmer detection [8]. The proposed method handles the constraints of swimming, with data augmentation and a model architecture that are specifically adapted for the task at hand.
Nicolas Jacquelin, Romain Vuillemot, Stefan Duffner. *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, presenter in chapter 4, has also been produced [9]. Its purpose is to perform registration, that is mapping the input video to a virtual top-view of known coordinate system. This gives a direct correspondence of a position (in pixels) in the image with a position (in meters) in the pool.
Nicolas Jacquelin, Romain Vuillemot, Stefan Duffner. *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, described in chapter 5, addresses strokes counting [10]. It introduces a general periodicity estimation method for videos and complex time-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.
Nicolas Jacquelin, Romain Vuillemot, Stefan Duffner. *Periodicity Counting in Videos with Unsupervised Learning of Cyclic Embeddings*. Pattern Recognition Letters, Elsevier, 2022, (hal-03738161).

In addition to these papers, two datasets have been introduced along them, namely $Swimm^{400}$, published in [8], a swimmer detection dataset, and $RegiSwim^{500}$, published in [9], a swimming pool registration dataset. We also published a benchmark dataset that groups those tasks along with non-addressed ones at the 2022 MediaEval challenge (detailed in section 6.3.8).

CHAPTER 2

BACKGROUND IN COMPUTER VISION

Contents

Computer Vision (CV) is the research field we contribute to in order to address automatic swimming video analysis. This field aims at developing methods to extract knowledge from input images. It has a long history from signal processing to data analysis and has been recently revolutionized with deep learning methods. This chapter will present general background in the domain, explaining different types of CV algorithms. The next chapters will use them to explain in further detail the State of the Art (SOTA) in their respective field (detection, registration, and periodicity counting).

The domains of application of CV are extremely varied and tend to develop with computation power increase, data accessibility, and new model ideas. The automation of vision-based tasks is tackled with CV (*e.g.* pedestrian counting, action recognition) and frequently paired with online cameras (*e.g.* video surveillance, plant monitoring, *etc.*). Monitoring tasks are indeed often addressed using CV as it gets more and more reliable and cheap. Apart from that, 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 [11]. Regarding sports analysis, as for this thesis, it can both be seen as monitoring and experts assisting. 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 also help coaches identify anomalies during a race or training. For instance, if

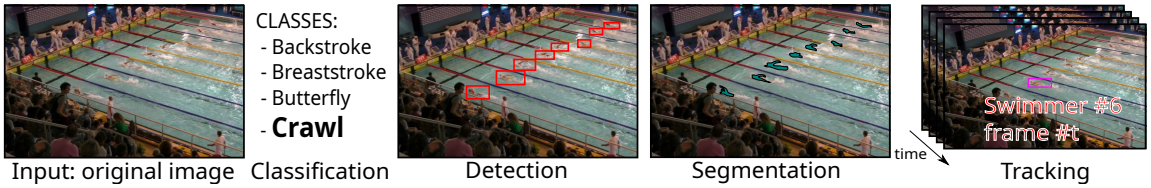


Figure 2.1 – An illustration of the different CV tasks applied to swimming race automatic analysis.

the stroke rate suddenly changes for a small period, a coach might analyze the corresponding video timecode and find out an improvement area for the swimmer. Here is an overview of common CV tasks related to our work, each illustrated in Figure 2.1.

- **image 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.
- **object detection**: objects in the image must be identified and located. Multiple objects can be present in the same image. The problem is formulated as the placement and shaping of bounding boxes framing the objects of interest.
- **instance segmentation**: this task estimates the probability of each pixel to belonging to a given set of classes. It can be understood as pixel-wise classification.
- **object tracking**: at the start of a video, an object is selected. Tracking is detecting it on each subsequent frame, even if it moves are changes. This task uses temporal information from the video to extract information from each frame.

A task is chosen because it addresses a problem, but said problem usually has other external constraints. The speed of execution 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 addressed. Depending on the use cases, one is more important than the other. Classic older algorithms described in the next section are generally 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 Network (DNN) models, orders of magnitude more computationally expensive than older algorithms, can nowadays run in real-time in the correct conditions.

Also, classic CV methods do not rely on as much data as the more recent ones. This is usually seen as the most important difference and the reason why

recent methods work better. This data dependency also has drawbacks that older approaches do not have.

2.1 Previously in Computer Vision

CV techniques have evolved a lot throughout the years. Understanding methods predating Deep Learning (DL) is required to understand the newest models. We also have to do it to understand works on automatic swimming race analysis [12, 13] that are based on these algorithms.

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). Overall, CV was much more limited than today, and as soon as an application was outside of classic calibrated tasks, it was impossible to manage the exceptions. By contrast, the majority of the modelling process of recent methods comes from the data and has better generalisation abilities. The "classic" methods mostly 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. We will only focus on the ones that were used or considered for our application.

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 pixel 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 gives richer and more meaningful results. 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) convolves through the whole image, which results in a new image of the same dimensions (if we ignore padding problems). The values of the filter are key as they define the nature of the output (*i.e.* the new resulting image). A wildly used filter is an edge detector named the Laplacian [14], that is illustrated in Figure 2.2. 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

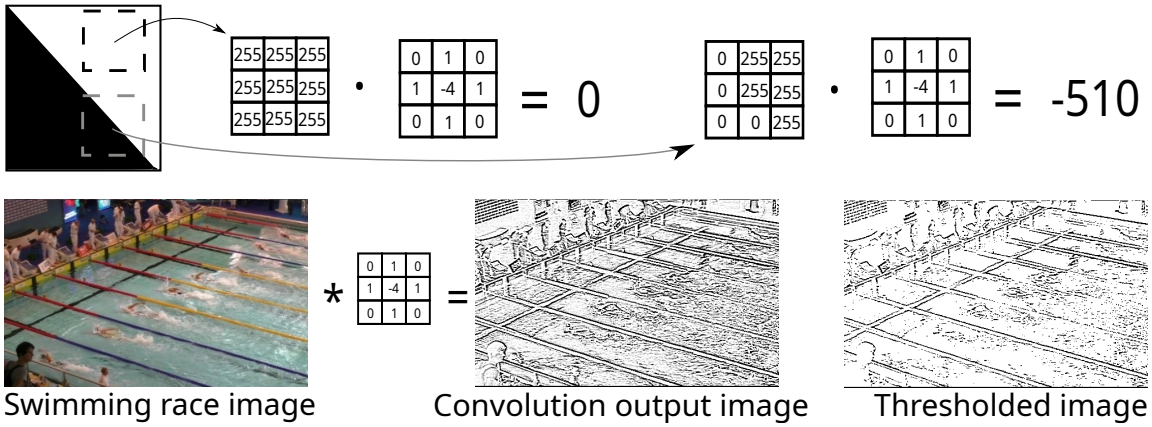


Figure 2.2 – Illustration of a 2D convolution edge-detection filter. The "*" symbol represents the convolution, the dot represents the pixel-wise matrix multiplication (*i.e.* the local behaviour of convolution), and the "•" represents element-wise multiplication. The 3×3 Laplacian filter is displayed with the result of its convolution on the input. 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.

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 visual pattern, but if the pattern is too complex, it becomes scale-dependent. 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 value in the matrix 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. Edge detection does not have these problems, as edges are not scale-dependent, so small size kernels generally give acceptable results, and the Laplacian can detect lines of any orientation.

After a convolution, one can apply a threshold to the output to binarize the result: either a pixel represents a pattern (an edge for instance) or it does not. This 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 reason 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 [15]. 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.

Hough Transform

Although convolution filters can identify local patterns, the result is still difficult to grasp for a computer. For instance in Figure 2.2, the lines are not perfectly continuous due to noise and threshold imperfection. Also, even if some pixels are identified as edges, corners, or similar local 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 [16] 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. Afterwards, 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 (x, y) image 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. If a line L of parameters (θ_L, ρ_L) contains N_L points on the source image, the position (θ_L, ρ_L) of the parametric space will have a value of N_L. 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 highly textured areas: if an edge detector outputs noise, many pixels can, by 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

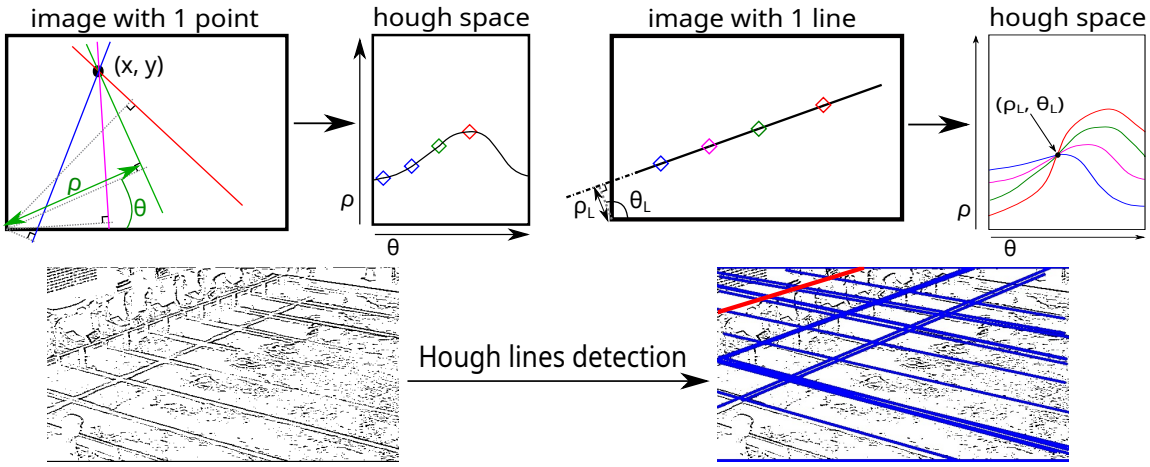


Figure 2.3 – Hough transform illustration. Top: toy examples with 1 point (left) and 1 line (right) with their respective Hough transform results. Colours are preserved in the example to map elements of on space to the other. In the point example, any line crossing the (x, y) position is represented in the Hough space using the angle (θ) and radius (ρ) as in the example. A single point thus results in a sinusoid curve. In the line example, the curves from each point of the line intersect in a single 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.

targeted content and its scale on the image. Finally, as a line has a width, it is possible to draw multiple mathematical lines from it, with similar parameters. It is thus common to have multiple highlighted points in the Hough space that come from the same line. As a result, multiple close (θ, ρ) pairs may exceed the detection threshold.

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. More recent methods significantly gained 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.

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, feature matching 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 (*e.g.* SIFT [17], SURF [18], ORB [19] or BRIEF [20], *etc.*), each with different properties. The most used is called Scale Invariant Feature Transform (SIFT). It performs discriminant 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 and have a chance 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 gradient's value is computed on 8 possible angles to normalize the possible outcomes. SIFT also ignores the magnitude of the gradients, as it is 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

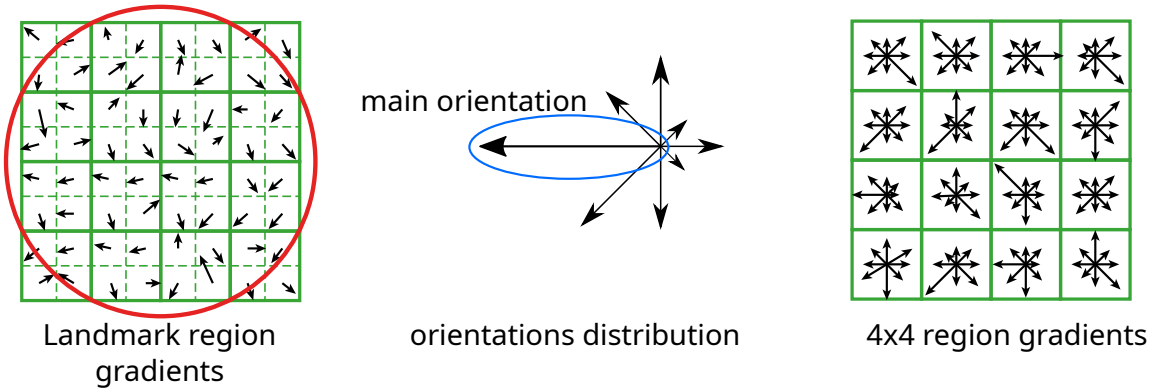


Figure 2.4 – SIFT descriptors creation. The local gradient is computed on a 16×16 region around the landmark's position. Their orientation distribution (among 8 possible angles) is computed and the dominant one is retained. Then the process is repeated for smaller 4×4 regions inside the area. These local 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.

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 [21]. 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 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 histogram 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.

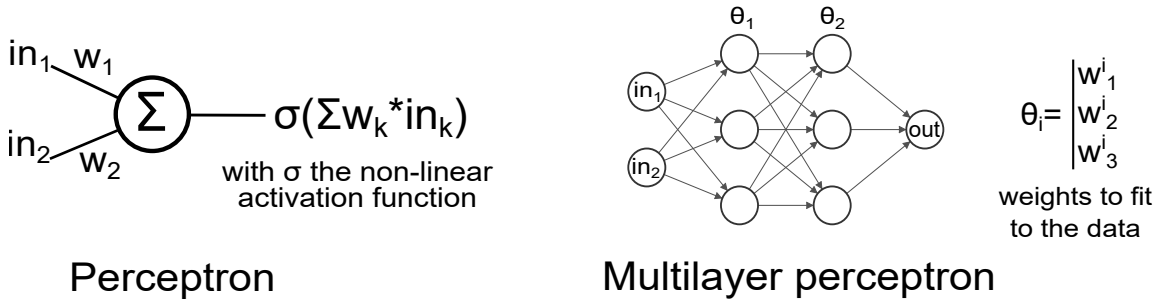


Figure 2.5 – The perceptron and a Multilayer Perceptron (MLP) architecture with 2 hidden layers.

2.1.2 Going Further with Machine Learning

ML can be defined as the algorithms which, given a set of inputs and outputs, choose the parameters of a model to map them. 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 [22], SIFT was conceived to detect these elements. If one generates SIFT descriptors for a face and studies what they represent [23], the ones centered around the eyes, the mouth, and other highly textured 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 [24]. Using varied object representations and ML can alleviate these problems.

However, despite having fewer human biases and being more powerful than many previous algorithms, ML has its problems and biases too [25]. 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, applied to a specific approach of ML, namely the Neural Network (NN).

Training Neural Networks

In the mid 50's, Rosenblatt conceptualized the perceptron [26], 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:

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

z_i and θ_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 mathematical model and a heuristics. The former integrates as many influencing elements as possible, using prior knowledge of the environment. This results in an explicit formula with highly interpretable parameters. If the system is not chaotic, estimating each parameters makes it highly predictable. The number of parameters depends only on the system and the equations used to manage it. However, a formula is not trivial to find, especially with high-level notions: mapping an image to an object class is far from being understood. On the other hand, a heuristics does not result in explicit 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, it can be obtained using learning and data, which we will detail in this chapter.

Gradient Descent [27] is the algorithm to fit the weights of a model to the data. Given a set of inputs X, outputs Y, and a NN f, the problem is to find the NN parameters θ that best map X and Y. 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:

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

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) [28] or Adam [29]. 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 [30] and substantially improved during the next decade [31, 32]. Back-propagation computes the error's gradient through the layers using the chain rule, following Equation 2.3 for the layer $l \in [1, n-1]$:

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

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

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.

Batch Training. 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. When each batch of the dataset has been used, one *epoch* is complete.

The choice of the batch size can be critical according to [33, 34]. On one side, [33] 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 ($< 10^{-3}$) and try increasing the batch size until performance decreases. On the other hand, [34] 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

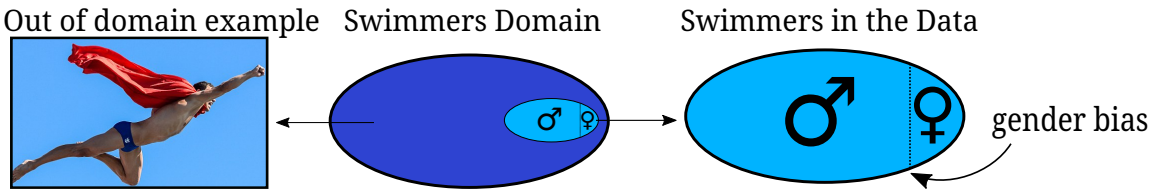


Figure 2.6 – Illustration of domain definition and data bias. The swimmer domain in the data (right) only represents a small portion of the entire swimmer domain (center). For a model trained on this data, the farther an image is from the training domain, the less likely it will be identified as a swimmer. 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.

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 afterwards, with small variations of the weights. Intuitively, it is because a model needs to understand what an image is before telling what it contains.

Data: the Solution and the Problem

The weights of a NN are adjusted to fit the data (*e.g.* an image associated with the swimmers' position). 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 [35]. 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, 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 not feasible, as there are infinite variations and

(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 [37]

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 may not fit 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 [36], 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 [37], 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 background, and ifs snow is visible, it considers it is 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 [38]: 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.

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 [39], 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. The datasets created during this thesis have limitations that will be discussed in their corresponding chapters (chapter 3 for $Swimm^{400}$and chapter 4 for RegiSwim^{500}).

2.2 Convolutional Neural Networks

Images are spatially structured in a way that a model analysing them must be translation invariant. A MLP that inputs an image, or the same image shifted by a few pixels, will output different results. This is a problem, as both represent the same general content. In 1989, inspired by the pioneer work of Fukushima [40, 41, 42], LeCun *et al.* [43] 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 type of NN, extremely well fitted for image analysis, called Convolutional Neural Network (CNN). An example is illustrated in Figure 2.10.

As seen in Section 2.1.1, 2D filters extract features from an image, resulting in what is called a feature map. For a CNN, a layer is composed of stacked 2D filters inputting and outputting different feature maps. 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 colours, edges, corners, *etc...* As the features combine through the layers, more and more abstract visual characteristics such as complex textures and shapes are 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 concepts close to faces, legs, hands, or clothes.

This section explains in more detail the use of deep models applied to CV. This domain requires specific elements and architectures of networks 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 of how this was alleviated.

2.2.1 Deep Learning

Looking at [43], one of the earliest CNN architectures, there are only 3 hidden layers. In [31], LeNet-6 architecture has 6. However, 2016's ResNet-152 [44]

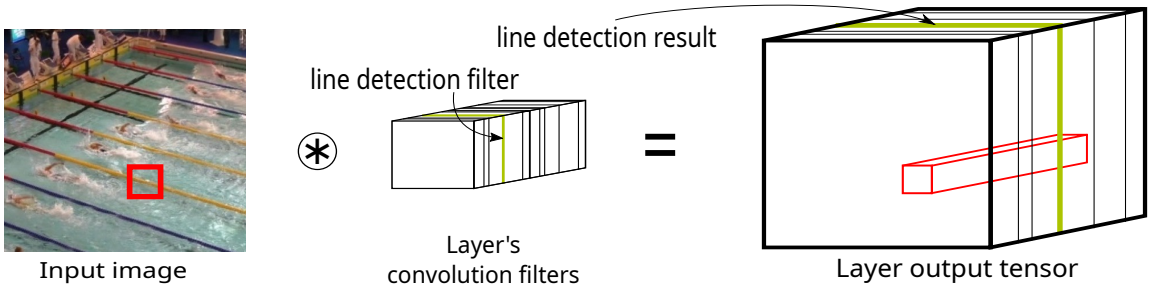


Figure 2.8 – Stacked convolutional filters forming one layer of a CNN. The "⊛" symbol represents convolution between the image and the filters. The red square contains visual features, not easy to understand for a computer. The corresponding red vector, on the layer output, contains these information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

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 [45] 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.

Better Abstraction

Understanding how CNN convert images into concise, abstract, and meaningful data is the key to grasping the interest of deep models. 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 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 of the feature.

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 [46] can 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 [47], 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 to be precise enough.

Previous Limitations

As deep CNNs can powerfully abstract images into understandable information 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 [48], 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 [49]. Although the computer was state of the art and the model very well implemented, several seconds were required to analyse 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) in [50], which claimed acceleration of 8 to 17 times. Indeed, a CNN can benefit from a GPU due to its 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 labels. 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 give them a label (*i.e.* an output value) to train a model on them. Even MNIST dataset [51] (1998, 70,000 images) and Letter Dataset [52] (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

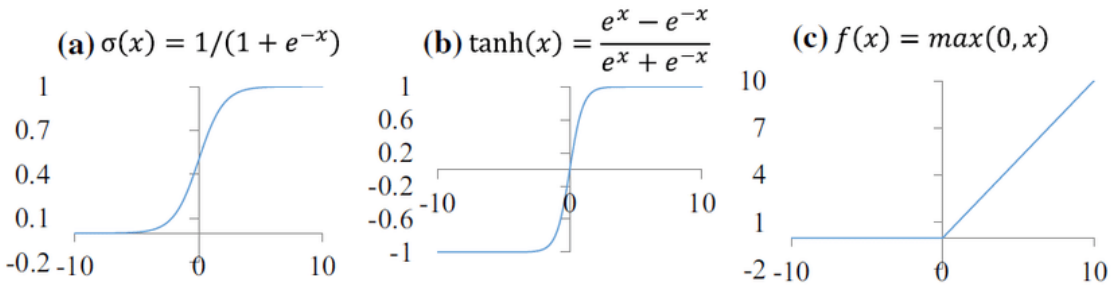


Figure 2.9 – Sigmoid (left), hyperbolic tangent (tanh) (center) and Rectified Liner Unit (ReLU) (rights) activation functions. The two firsts saturate when moving away from the origin, thus resulting in a weak gradient as their absolute value gets bigger. ReLU has a higher derivative for any positive value, enabling a better gradient back-propagation. Scheme extracted from [54].

the release of Imagenet [53] 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 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 [55], 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 behaviour 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 [56] 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.

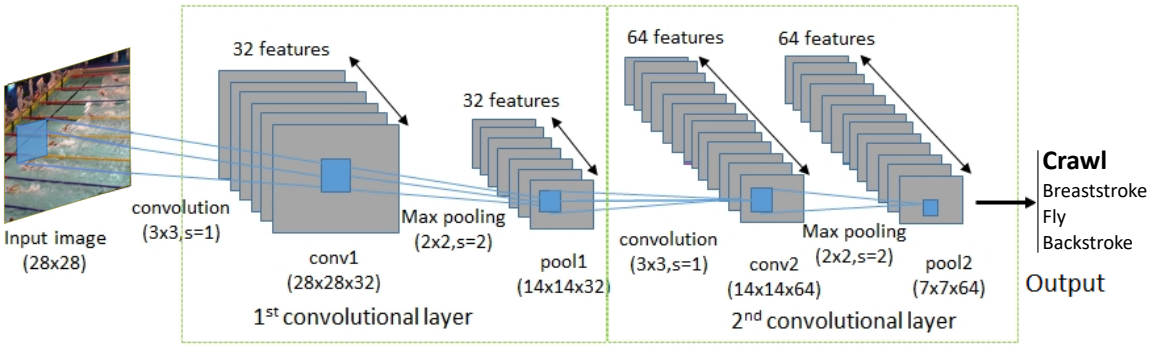


Figure 2.10 – A CNN architecture with 2 convolutions, each followed by a Max Pooling operation, completed by a classification layer at the end. The task at hand is classifying the swimming style shown in the input image.

2.2.2 CNNs Components

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

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 one 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 [57] 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 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.* [58] 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 [44]. 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 [59, 60], 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.

The most efficient existing architectures used today [44, 57, 58, 61] 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 this will be nuanced, but the core idea will remain.

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 [62]. 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 [63]. 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 downsizes a small region by only keeping its most activated channel. This prevents the previously mentioned 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 [64]. The same article explains how spatial organization is important for CV models: the feature's relative position with each other 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 [57]: 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 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 strides longer than 1.

2.2.3 CNN 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.

We also precise that Vision Transformer architectures [65] will not be explained here. They were not used during the thesis due to their needs in data and the fact that we aimed at reducing our data needs. More details on the subject are provided in the perspectives of this manuscript (see section 6.3.6).

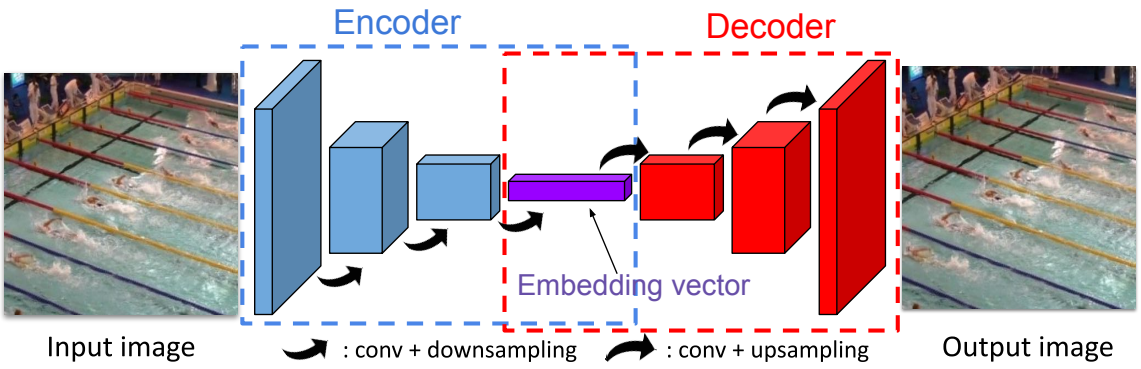


Figure 2.11 – An encoder-decoder architecture. As the input and output are the same, it is an autoencoder. The length of a block represents its number of channels. Remarkably, the bottleneck of a linear autoencoder converges into the Principal Component Analysis (PCA) representation of the data.

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 detection layers on top of the encoder. Note that encoders usually reduce the data width and height, but this is not always the case, as in [6] 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.

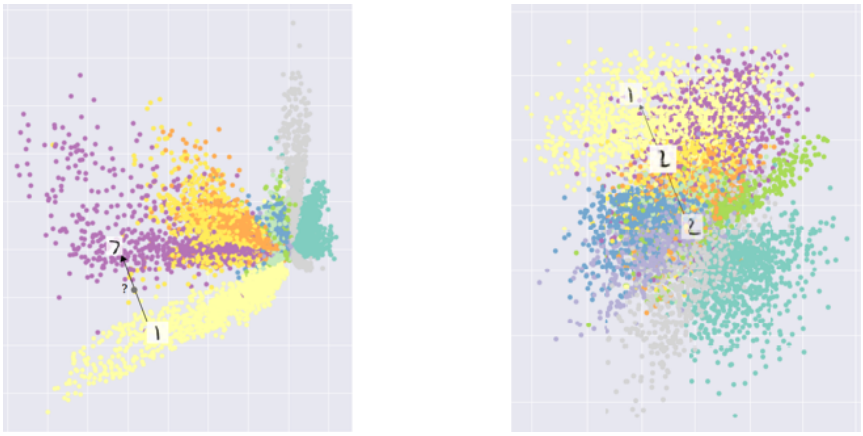


Figure 2.12 – 2D PCA visualisation of different autoencoders trained on MNIST. Colours represent classes. Left: an autoencoder with a non-continuous manifold in the latent space. Right: a VAE with a dense continuous manifold. Figure from [72].

An autoencoder [66, 67, 68] 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 [69]: 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 [70]. The MSE converges faster at first because quadratic functions penalize more the bigger 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 [71] 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 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) [73]. 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.

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 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 [57] 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 a scalable way to increase the depth of an architecture.

A solution was proposed by He *et al.* [44]. They 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.

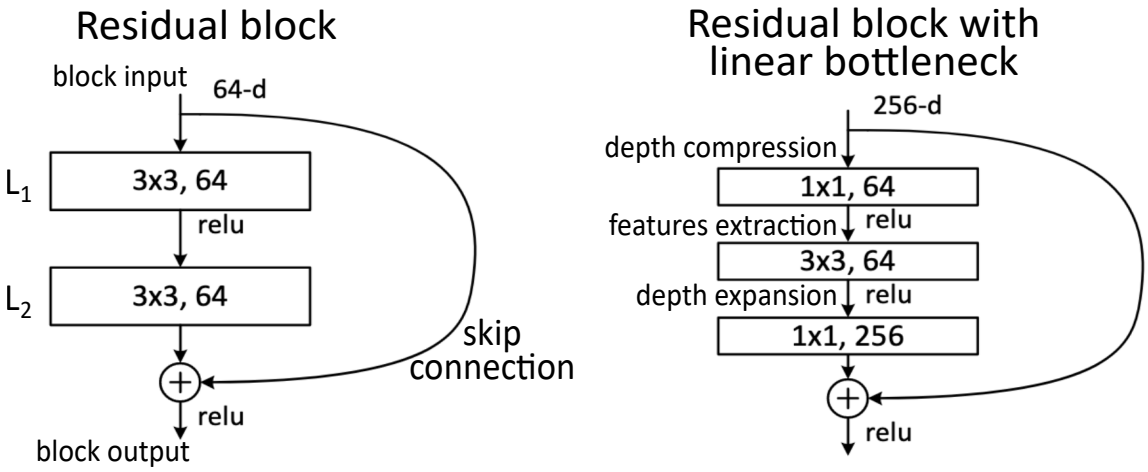


Figure 2.13 – Elementary residual blocks. Variations can be further applied to them, but the core feature is the skip connection, back-propagating the gradient directly from the block output to its input. On the right, an illustration of the linear bottleneck, useful for deeper networks. The (1×1) convolutions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

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 architectures are more accurate but slower. Variation of the ResNet architecture with 18 up to 152 layers [44], and all the other architectures that followed [58, 74, 75, 60, 76], are made of these blocks. Such models have better use of data and processing power, as showcased in [77]. For the deeper ones (>50 layers), linear bottlenecks are used to reduce the number of parameters by locally reducing the number of channels, as explained in 2.13 right.

U-Net

U-Net [61] 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 2.14. 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 2.14 is the original U-Net, 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 U-Net-like architecture.

The encoder extracts deep features describing the image. As the deep tensors are upsampled in the decoder, they are stacked with slightly lower-lever features but higher spatial precision. As a result, U-Net is extremely powerful to per-

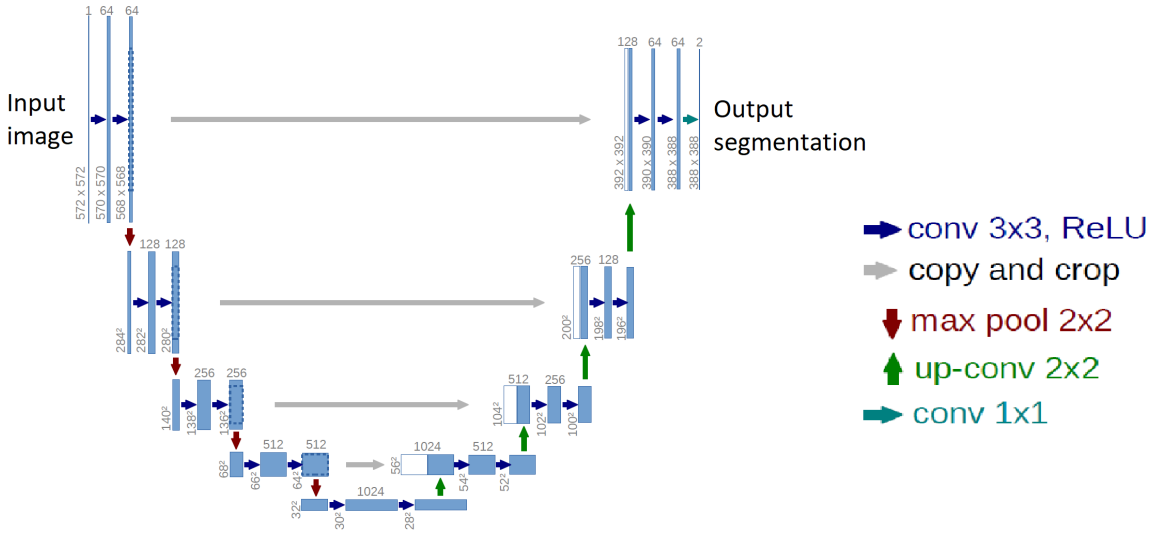


Figure 2.14 – The U-Net architecture, from [61].

form segmentation tasks, where both semantic meaning and pixel precision are required.

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 U-Net though, the earliest convolution blocks are directly linked to the latest ones symmetrically. Therefore, this architecture has a gradient flow which enables fast gradient back-propagation through each layer. U-Net was originally developed for biomedical image segmentation, where data is often lacking. It is a standard benchmark for the segmentation of organs or tumours. Due to its interesting training properties, this architecture will be used in chapters 3 and 4.

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 that data wrangling in general represents an enormous time of a ML model development (50-60% in [78, 79]). 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 must fit the nature of data.

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.

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 label. The vast majority of labels are either a set of classes [53, 51, 80], bounding boxes [3, 4], or segmentation maps [81, 82, 83], 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 [84, 85]). Bounding boxes can be encoded in several ways depending on the method itself and they will be presented in detail in Section 3.2.1. 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 is 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 [3] 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.

Data Gathering

The labelling process is very long, but gathering images can be too. With the Internet [86], 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 image 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's way of filming is extremely constant, with only a few different shot angles, and only part of them

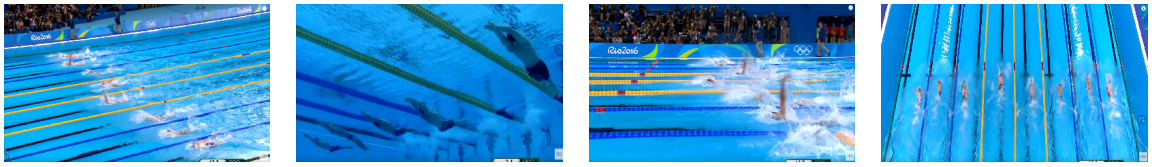


Figure 2.15 – Different views from a classic TV stream. Apart from the leftmost, they are very different from what coaches are used to.

exploitable for deep analysis. In Figure 2.15, 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: https://www.dartfish.tv/ChannelHome?CR=p153270. 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 are 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.

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 [87]. Unclean data can comprise multiple occurrences of the same image, modified data (photomontage or images with marks for instance are omnipresent in the dataset Pascal-VOC [4]), too small images, unusual ratios (there is a (500x32) pixels image in Imagenet [53]), *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 [88]. 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 [89] or using crowd-sourcing methods [53] for big datasets, or manually for small ones.

Orders of Magnitude of Computer Vision Datasets

The ability to digest huge amounts of data has not reached the limit of recent architectures [90, 65] (see the different test sample numbers in [45]). Training with thousands, tens of thousands or even millions of images is frequent, and it seems to be generally beneficial to the deeper models. Indeed, before the explosion of deep models, one of the most used detection datasets was Pascal-VOC [4] 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 [53] 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 [3] 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 [91] 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 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 expected output. Due to external constraints (availability of data, mostly), they can be different. The following section describes the challenges associated with NN training in general and how they are associated with data. A scheme illustrating the relationship between data and training algorithm is shown in Figure 2.16.

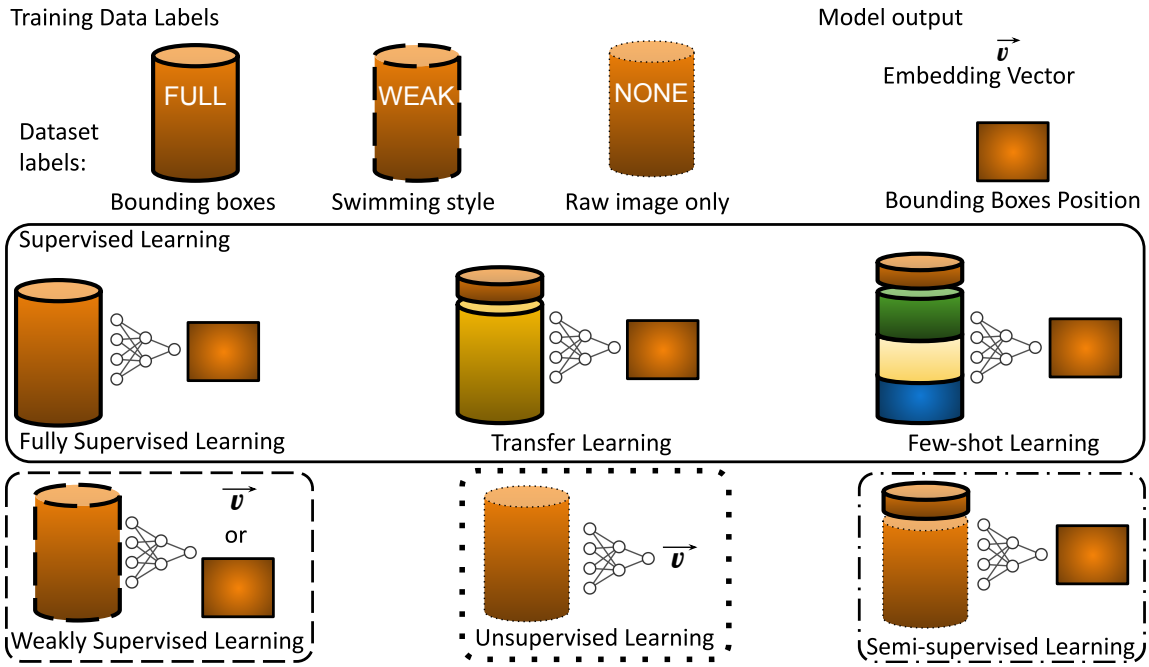


Figure 2.16 – The different levels of supervision. Datasets domains are represented each with a specific colour. There are different levels of label, here represented by the cylinders' edge. Although each type of supervision has a different complexity of training data, they all aim at performing the detection task. Note that for transfer learning, the "big" domain is distinct but close to the target domain.

Supervised Learning

Fully-supervised training is the basic level of supervision. It means the desired outputs are present in the massively available data. In this condition, learning is straightforward and no other method or trick is required to optimize the model.

Transfer learning [92, 93, 94] is frequently used when one is limited to a small dataset that has not enough samples to get acceptable results. 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 2.20, 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.

Few-shot learning tackles an even more extreme case of this "annotated data lacking" problem when one only has 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. Few-shot learning uses completely different techniques from transfer learning [95, 96, 97, 98]. 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, a suitable solution is 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 before accomplishing the task. This definition is extremely generic, though, because the existing methods addressing few-shot learning are very diverse. A widely used few-shot classification method [99] 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 [95], 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 [95]) 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 large while the distance to the known embedding vectors of S will be small. This idea is explained with Figure 2.17.

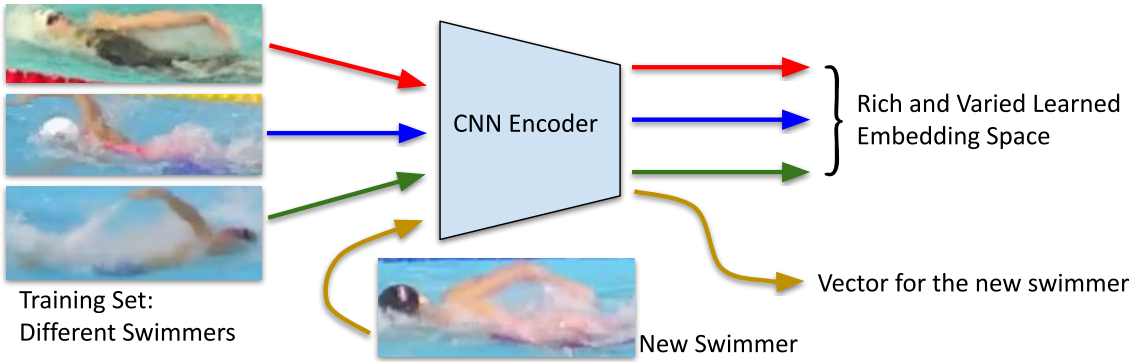


Figure 2.17 – An illustration of one-shot learning (*i.e.* the most extreme case of few-shot learning) applied to swimmers identification, inspired by [99]. The image of the new swimmer is embedded by the model. Afterwards, each new swimmer image is compared to the embedding vector of the different swimmers, including the new one.

Few-shot learning is also addressed by other approaches, such as meta-learning [97, 98], 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.

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 [100, 101, 102, 103]. 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) so weakly supervised learning is a solution [104, 105]. 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 [103], 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

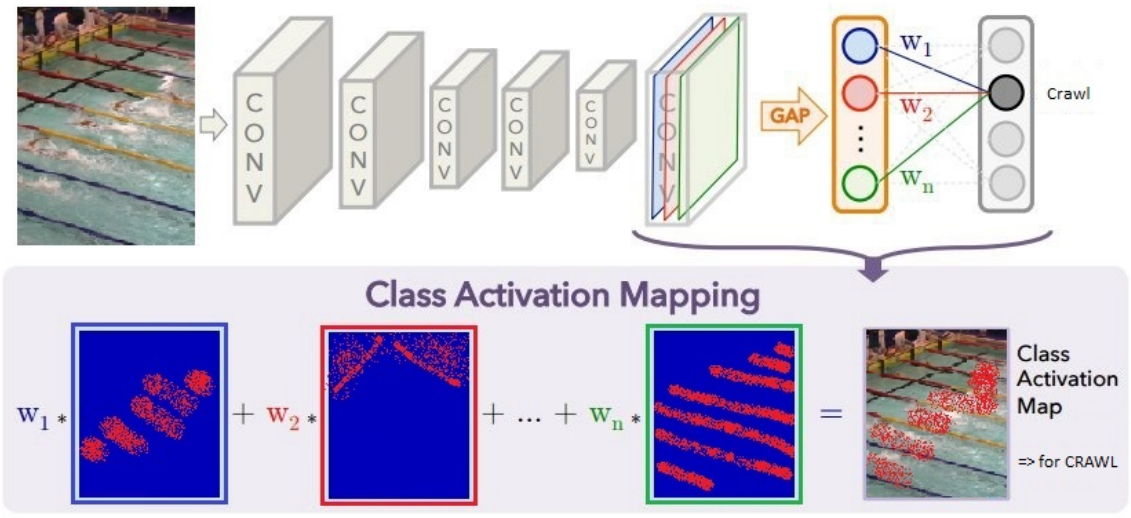


Figure 2.18 – The CAM pipeline explained (figure inspired by [103]). The objective is to detect swimmers using a model trained to classify the swimming style. Such proxy task works because the swimming style can only be correctly identified by focusing on the swimmers. Each feature map of the last convolution layer's output is weighted by the coefficient it assigns to the freestlye class (w_1, to w_n). 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.

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 2.18. 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 [101], 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 [102] 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 give the expected end result (detection for instance,

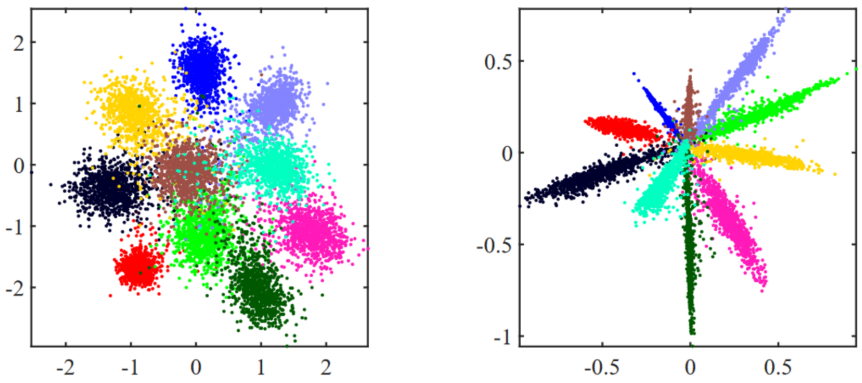


Figure 2.19 – 2D PCA of embedding vectors from encoders trained on the MNIST dataset. The colours represent the different classes. Left: the model is trained using metric learning. Right: the model is trained with an additional classification layer with softmax activation. Extracted from [111].

if weak labels are classes), it can 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 [106] or the magnet loss [107]. 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 [108]. 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 [109]. As shown in Figure 2.19, training a model on a classification task using softmax activation [110] in the end creates similar groups. However, the end distribution is very different: metric learning creates a dense manifold with smooth frontier between classes, whereas the latent space of the other is significantly sparser. Being less specific, the first is better-suited for feature encoding of new similar images.

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

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 and representation learning [112, 113]. 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 (introduced in section 2.2.3) is an unsupervised learning model. Its main interest is 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 [114, 115, 116]. 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 [117]. This helps the model to learn a better representation of the content featured in the input instead of just reciting the content of an image. Such added artificial noise can be varied: blur, (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 [118, 119]. 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 [112, 113]. 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

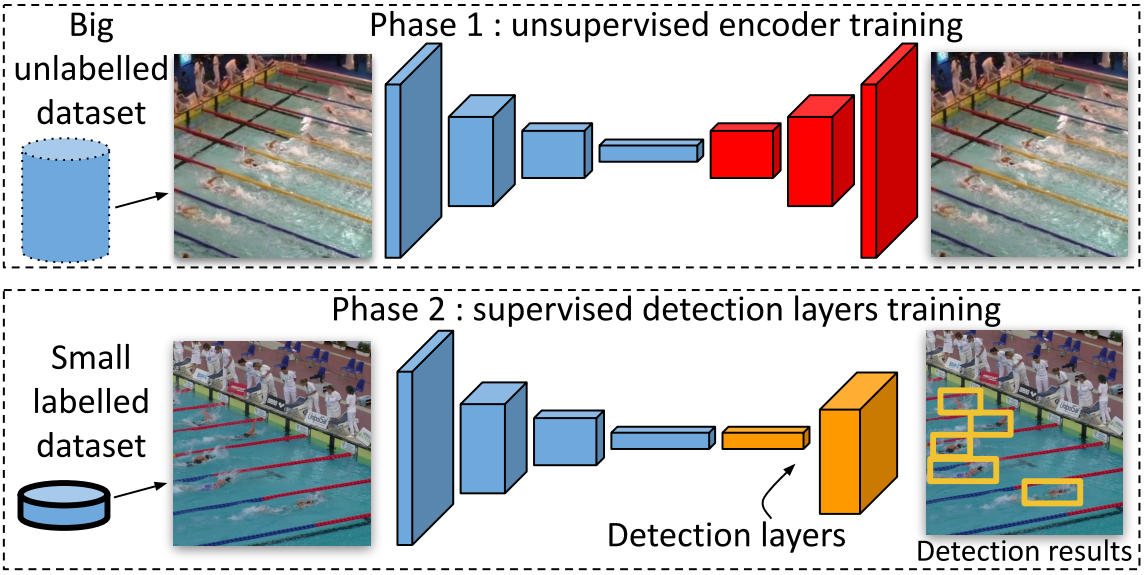


Figure 2.20 – A semi-supervised pipeline applied to detection. Phase 1: an autoencoder is trained on a big unlabelled image dataset to create a feature extractor. Phase 2 (transfer learning): the encoder's weights are frozen and layers are added on top of it and trained on a small labelled detection dataset.

to alleviate data augmentation transformation. Therefore, the resulting encoder is robust to noise and learns a good representation of the dataset. As Figure 2.19 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.

Semi-Supervised Learning

Semi-supervised learning [120] is the combination of supervised learning and unsupervised learning, illustrated in Figure 2.20. 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 [121] 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 [122, 73]. It is possible to separate the unsupervised and supervised learning [118]. 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 [119]. 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 [123, 124].

Another frequently used approach, named self-training, relies on pseudo-labels [125]. 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 [126]. Instead, it was proposed to use a fixed proportion of the best new elements [127] 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 [128, 129]. Pseudo-labels are also often used in other domains, such as weakly-supervised learning, as it does not require extra annotations [112, 113].

Semi-supervised learning has limitations. In [71], 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 feature learning. Indeed, it was shown that a good semi-supervised learning encoder actually needs a bad generator [130].

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 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 [7]. Weakly supervised learning SOTA [101] 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 [96]. 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 [131] 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 algorithms such as Faster R-CNN [132]. 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.

CHAPTER 3

SWIMMER DETECTION

Contents

Chapter abstract

This chapter tackles the problem of detecting swimmers in an image. It aims at performing accurate detection while relying on few data with unusual features. It thus does not rely on transfer learning methods, but on a new model trained from scratch for our purpose. Its results will be evaluated and important design choices will be explained through different ablation studies.

A dataset for swimmers detection will also be presented. It 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.

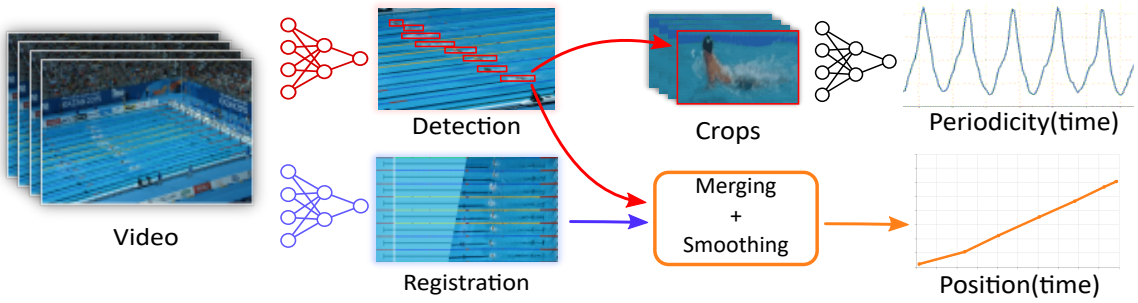


Figure 3.1 – The swimming race analysis method. The detection part is tackled in this chapter. It is arguably the most critical bloc, as both swimmers placement in the pool and periodicity analysis depend on it.

This chapter mainly concerns the work published in:

Nicolas Jacquelin, Romain Vuillemot, Stefan Duffner. *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

Swimmer detection is the process of localizing the visible parts of a swimmer's body in a picture. Its role in the general pipeline is defined in Figure 3.1. Combined with registration (see chapter 4), it gives to know the position of the swimmers in the pool, thus giving meaningful analysis information. Detecting a swimmer in an image - and by extension 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 under a surface full of reflections and diffraction, affected by unpredictable waves creating many local deformations in the image. The light on the water tends to saturate the camera sensor, or at least obfuscate the swimmers underneath. Their accurate detection is thus harder than for a normal human. An entirely different model has to be created to detect swimmers during competitions and training.

Contrarily to daily-life objects, 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 know a swimmer's perfectly fitted boxing. In this chapter, we show an architecture and a training method that limits the impact of this problem.

Apart from that, even recent deep learning methods [5, 6, 133] usually require a large amount of carefully labelled images. Many datasets of the sort exist [3, 53],

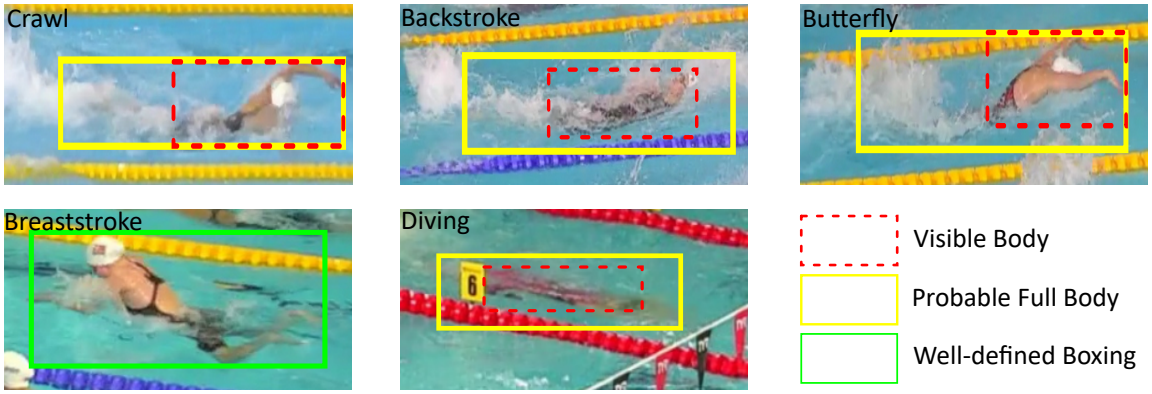


Figure 3.2 – Representative examples of the edge fuzziness problem with the different styles and the diving. Although for breaststroke the framing is generally well-defined, for other contexts it is not. The swimmers create waves and splashes keeping an observer from knowing the exact boxing of their body. Even with an unexcited water surface, diffraction deforms the observations and shifts the visible position of a swimmer from their actual position.

but in our context, they show major drawbacks. First, training a model on such data collections is computationally expensive and time-consuming, which we want to avoid. This is one of the main emphases of this thesis: the feasibility and accessibility of our method. Second, the class "swimmer" is present in no public datasets. Although the class "person" is very well represented, from a Computer Vision (CV) perspective, they look rather different. One could simply not use it for swimmer detection. Third, although fine-tuning a robust generic model may enable the training of a new well-performing model with few data, it is not suited here. The same applies to more recent approaches of domain adaptation and few-shot learning. They work well if the image distribution is not too different from the daily-life context they were trained on, but this is not the case for swimming. Due to the many local deformations the pool environment creates, common existing models perform poorly. Regions Of Interest (ROI) detection and image embedding models are not suited 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.

Our 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 3.4). It also can easily be applied to other swimming-based sports like water polo (see Figure 3.10). Our main contributions in this chapter are the following:

- a model for automatic and robust swimmer detection in competition videos,
- an annotated swimmer detection dataset,
- a method to easily train the model which reaches high performance with few data.

3.2 State Of The Art

This section starts with an overview of modern CV object detection techniques and then presents recent works applying them to swimmer detection.

3.2.1 General Object Detection

Recent object detection approaches can be divided into two groups: single-shot and multi-shot. Generally, the first is faster and the second is more accurate. This section will illustrate them by explaining representative architectures of the domain. It is also possible to use segmentation as a proxy for detection, which will be addressed in section 3.3.2.

Multi-Shot Object Detection with R-CNN

Multi-shot object detection algorithms input several times a ROI (*i.e.* a small patch of the image). This refinement over an initial detection increases the precision. Searching at multiple scales and creating several queries individually analysed is the foundation of multi-shot object detection. We present in detail what is arguably its most influential architecture: Faster Region-Based Convolutional Networks (R-CNN). The R-CNN family consists of 3 main elements, illustrated in Figure 3.3, left: (i) the backbone (*i.e.* the feature extractor), (ii) the Region Proposal Network (RPN), and (iii) the classifier.

The objective of the RPN, illustrated in Figure 3.3 right, is to find ROI, and (if it does) adjust their coordinates around the object they frame. The RPN inputs the features from the entire image (extracted by the backbone) and outputs a set of heatmaps, each associated to an anchor box of a given size and aspect ratio. Once the RPN is trained, the heatmaps activate at the barycenter of the different objects. More precisely, only the heatmap matching the best the area of a box is activated. This gives a first rough estimation of the object's position and dimensions. The features corresponding to these positions are fed to a refinement layer, regressing more precise spatial information to fit the found object of interest. These are the ROI. Finally, patches of the feature map corresponding to the ROI are extracted. They are separately fed to the classifier, which outputs a probability of presence for each class inside of the ROI. If none is above a defined threshold, it means the box is a false positive and it is discarded. If not, we obtain the exact position of a bounding box and its corresponding class.

Faster R-CNN is a very powerful object detection model. Its small anchors make it great at detecting small objects, a common problem in object detection. However, it is very slow, 3-4 Frames per Second (FPS) on average, because of the many ROI extractions and analysis by the classifier. The backbone inference is also

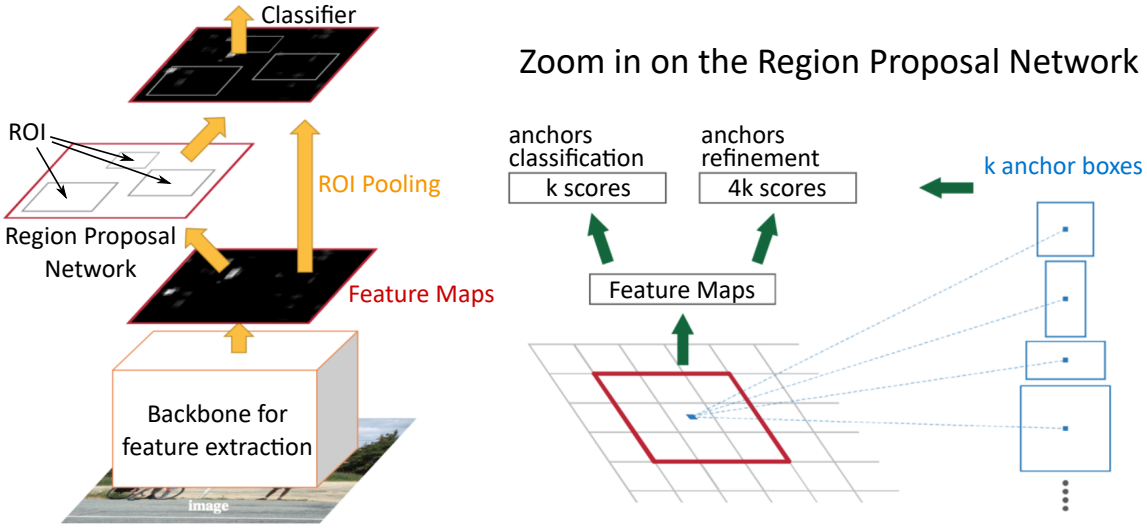


Figure 3.3 – The Faster R-CNN algorithm. Left: the entire architecture. The extracted features are used by the RPN and by the classifier. Right: detail of the RPN. It uses the features both to classify each area as ROI or background and to refine its detections. Anchors refinement generates 4k values (*i.e.* 4 values per anchor) that correspond to (width, height, x shift, y shift) $\times$ k anchor boxes. Figure adapted from [134].

heavy: it is better to use big images (600 $\times$ 1000 pixels in the paper) to have a higher output resolution for the RPN and thus be more precise.

Single-Shot Object Detection with YOLO

Single-shot detectors input the image once, with no sub-patches division or multiple refinements over the same area. The two first main algorithms explicit it in their name: YOLO [135, 136, 5] and SSD (Sing-Shot Detector) [133]. They both rely on the same underneath technique: a feature extraction backbone, directly connected to regression and classification layers. Their main difference is in the architecture, SSD stacking different scales of feature maps to get more precise results, but at the cost of time. YOLO will be explained in this section, as it is an extremely popular algorithm. It performs detection and classification at once using anchor box adjustment.

Its behaviour, illustrated in Figure 3.4, is straightforward. A backbone extracts features from the image. One can see each tensor cell (*i.e.* spatial dimension of the tensor) as a region's deep representation. For each cell, YOLO regresses k bounding boxes, each associated with an objectness score (*i.e.* probability that an object the size of the anchor is present), a class vector, and positional information (to place the box exactly around the object). This results in (too many) boxes one can position on the image, as illustrated in Figure 3.4, "generated bounding boxes". The anchors with a high enough objectness score are kept. A greedy NMS is applied to filter the boxes of the same class overlapping too much, likely

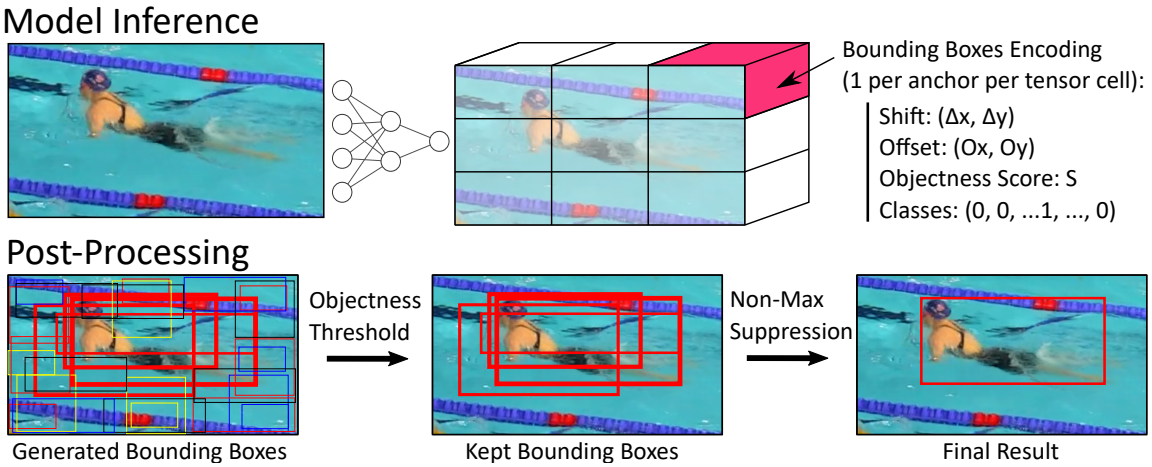


Figure 3.4 – An illustration of You Only Look Once (YOLO). The output tensor is only (3×3) for simplicity. A box colour represents its class. Objectness score is represented by box thickness. The model outputs an encoding for each anchor of each sub-region. Only the ones with an objectness score superior to a threshold are kept. Non-Maximum Suppression (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.

representing the same instance. This results in the boxes of interest, as in Figure 3.4, "final result"

YOLO is extremely popular and simple to use. However, its precision is lower than Faster R-CNN and other slow models. The algorithm is also limited in the density of its detection as there is a fixed amount of anchors per cell.

3.2.2 Recent Advances on Swimmer Detection

First, we need to mention the work of Benarab *et al.* [12, 137, 138], who recently proposed several techniques pursuing the same detection goal as ours. Their computer vision algorithms are based on low-level techniques and in particular Joint Transform Correlation. They transform the pool image and swimmer head reference images into a 2D complex spectrum and use 2D convolution to find correlations between them. The reference images' variety and choice are key: they must be representative of the swimmer's head domain to work. They repeat this process at different scales and orientations to be more robust. Although it allows a faster inference and low computation needs, this choice leads to hand-crafted, pool-specific thresholds. 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.* [139] 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 [5] model, where classes describe the swimmer state (normal swimming, diving, u-turning...). Their model reaches an AP25 (Average Precision (AP) definition in Section 3.4) of 0.7.

Hall *et al.* [140] 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 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 is fed 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 release their dataset or model, so the comparison is impossible.

3.3 Proposed Approach

The first step in detecting swimmers is to create a dataset for the task. This section describes the data acquisition and labelling process and explains its properties. We wanted to focus on a detection model trained from scratch using few data. This did not allow us to use classic methods such as YOLO [135] or Faster-RCNN [6]. Instead, we drew our inspiration from medical imaging, where data is always missing. This is because the images come from costly medical imaging instruments and each label is made by a specialist. 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 a small dataset. Moreover, they have interesting spatial properties this section will explain.

3.3.1 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 total of 3121 bounding boxes (7.7 per image on average). Some competitions were inside, others outside, with varying lighting conditions. For each competition, races were selected to represent

the 4 main swimming styles and the 2 genders (49% ♂, 51% ♀). The freestlye 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, called $Swimm^{400}$, is composed of 80 test images and 323 train images.

3.3.2 Detection Through Segmentation

The segmentation and detection tasks have different objectives, 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.

Bounding box regression vs. segmentation

Bounding box regression-based approaches [5, 6, 133] 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 which requires large amounts of labelled training images to get stable.

On the other hand, fully-convolutional models like U-Net [61] 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 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 excludes the parts of the surrounding background.

Tiny-U-Net

We propose a variant of the well-known U-Net architecture [61] 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 each downsampling (in the encoder) or upsampling (in the decoder). The following modifications have been performed: instead of 3 convolution layers between each sampling operation, only one is performed. The filters are also smaller, increasing from 8 up to 128 instead of 64 to 1024 for the original U-Net. A side-by-side scheme comparison of both is given in Figure 3.5 We will designate our model by tiny-U-Net. Due to its shallow architecture and low filters number, it runs at 260 FPS on a GTX 1080

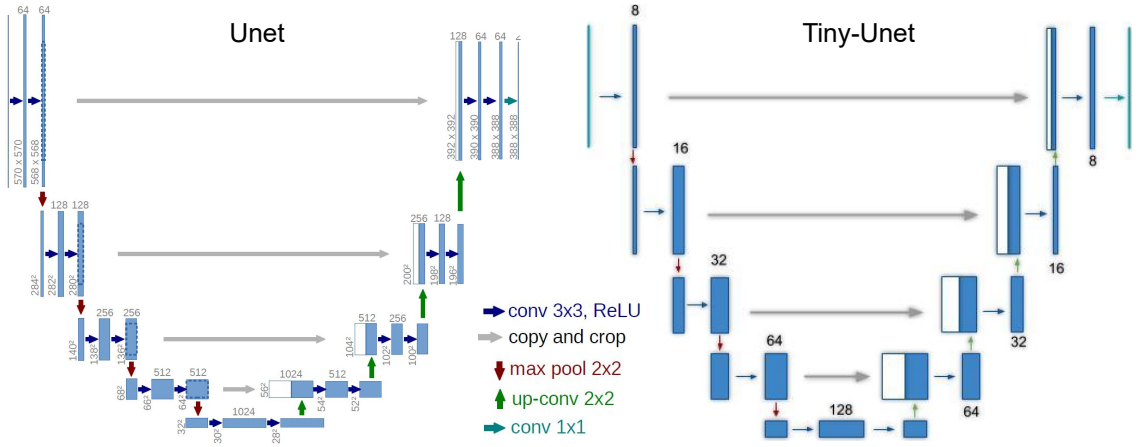


Figure 3.5 – Comparison of the classic U-Net architecture and our tiny-U-Net. The latter is much more compact, each level having both fewer convolution layers and fewer filters per convolution. It is thus significantly faster (5×). Despite this complexity reduction, Tiny-U-Net is still complex enough to isolate swimmers.

NVIDIA GPU with (256 × 256) pixels images, whereas U-Net runs at 50 FPS in the same conditions. Tiny-U-Net 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 3.4, each of them creating bounding boxes.

Box-to-Segmentation-Map Transformation: the U-Net model requires segmentation heatmaps for training. 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 binary masks or Gaussian values (close to zero as the pixel is far from the center). As shown in Table 3.1, 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.

3.3.3 Data Augmentation

Training Tiny-U-Net is direct as it is a simple forward model without hyper-parameters. To allow batch training, the images were all resized to (256 × 256)

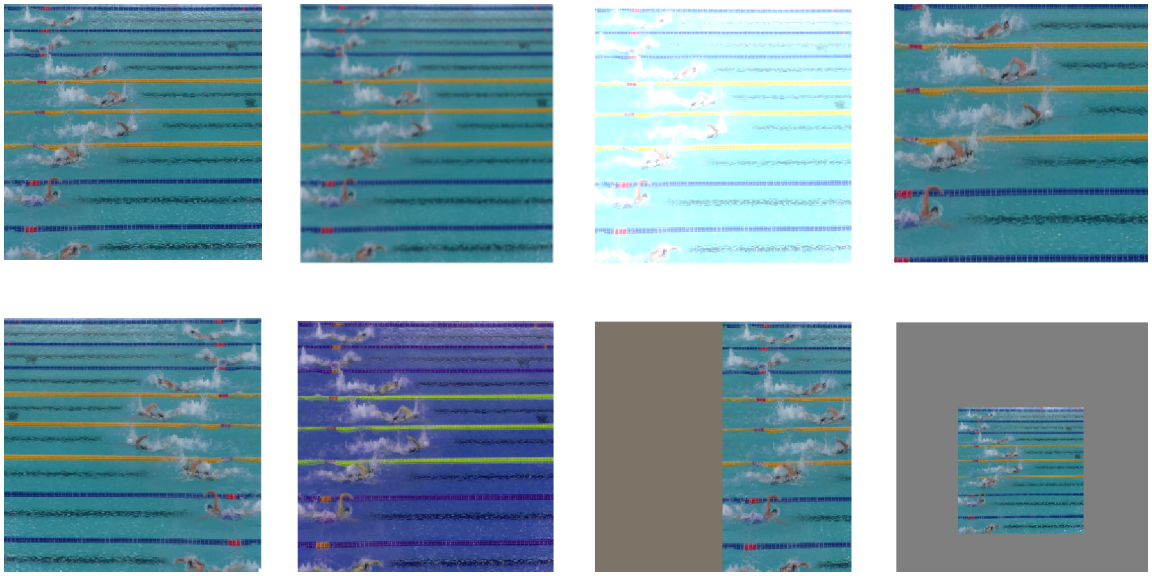


Figure 3.6 – The data augmentations used to train the model. From left to right, top to bottom: original image, blur, contrast and brightness change, crop, horizontal flip, hue change, side switch, zoom out.

pixels. Data augmentation is used to increase the trained model's performance. It will also compensate for the low quantity of images in $Swimm^{400}$.

Zoom-in / zoom-out: the main augmentation was the zoom in and out (Fig 3.6, 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 3.6, 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 from only detecting instances at the center. For fully-convolutional networks, this is less of a problem as they are mostly translation invariant.

Others: apart from these two transforms, other more common methods were used to train the model. The random left-right flip generalizes the swimmer's 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 skins, pools and water colours. The contrast and brightness random variations adapt the model to the many lighting conditions that can happen during 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 colour 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.

3.4 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.

3.4.1 Metrics

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

$$AP\ 25 = \frac{1}{N}\sum_i \frac{\#Good\ Detections_i}{\#Positives_i}\ , \quad AR\ 25 = \frac{1}{N}\sum_i \frac{\#Good\ Detections_i}{\#True\ Positives_i}\ , \quad (3.1)$$

on $image_i$, $\#Good\ Detections_i$ being the number of 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 $\#True\ Positives_i$ being the total number of boxes *labelled*. N is the number of images in the benchmark set.

To get a better idea of what these metrics represent, consider the different detection scenarios presented in Figure 3.7. 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

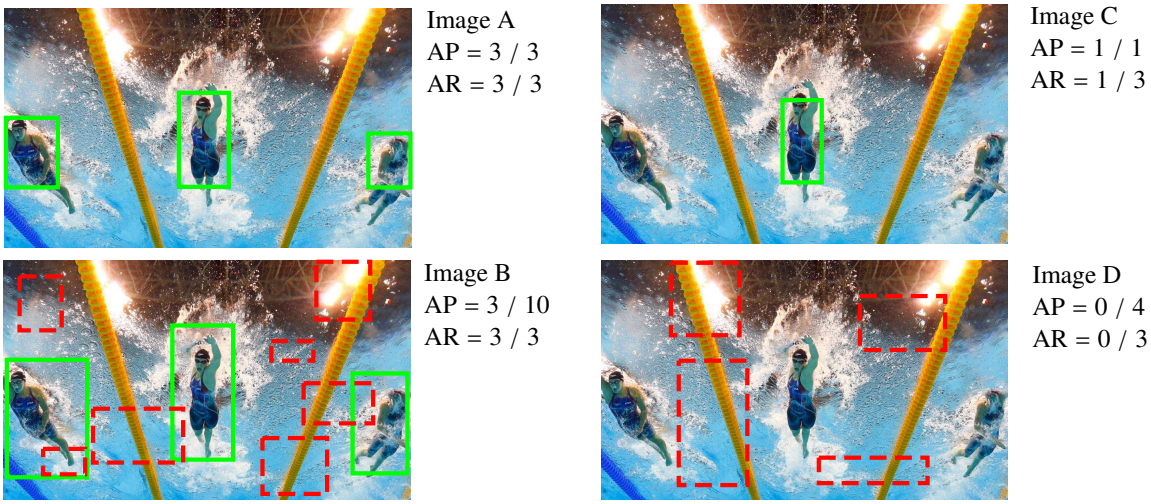


Figure 3.7 – Different detection scenarios on the same image. In green and continuous: boxes with more than 25% IOU with a true box (*i.e.* true positives). In red and dashed lines: incorrect detections (*i.e.*: false positives).

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) [141], 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 for pixel-perfect precision, which is not adapted for swimmers with blurry edges under the water. AP50 is the standard metric and threshold for object detection. It considers valid a box at the correct position and with the correct general size and proportion. In our case, though, the width and height of a bounding box are fuzzy and imprecise by nature, even during annotation. For this reason, we chose a smaller threshold of 25. AP25 and AR25 are closer to what real-world applications seek, such as extracting a sub-region around swimmers, where estimating the boxes' barycenter and general size is enough.

3.4.2 Ablation Study

First, we trained our tiny-U-Net model on different variants of the box-to-segmentation map strategy. The results are shown in Table 3.1.

Table 3.1 – Detection performance of our model trained with different swimmer heuristic shape on the target heatmaps. The input image size is (256 × 256) pixels. * 72 in the original paper, improved since.

Training Data	AP 25	AR 25
Ellipse binary mask	**76***	**60**
Rectangle binary mask	60	28
Ellipse Gaussian mask	21	5
Rectangle Gaussian mask	13	3

This table clearly shows the superiority of shapes with hard edges. Smoothed ones tend to reduce the model convergence during training. Finally, filling an 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. 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-U-Net with current state-of-the-art methods, we trained two variants of Yolo on $Swimm^{400}$. The first version is YoloV3 [5], 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, 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 [135] Section 2.2 is followed. From Table 3.3, it appears that our tiny-U-Net 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-U-Net does not require such a transformation. This result is confirmed by testing the 3 models on the same videos: the tiny-U-Net gives the best results. Moreover, on a video, tiny-U-Net appears much more stable between one frame and the next.

We further explored different domains to get a better knowledge of this model's abilities and limits. The first experiment we did was to measure its speed. It is significantly impacted by the image size, also affecting performance. Table 3.2 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 swimmer to get a good detection. On the other hand, too big images tend to contain many false positives, even if

Table 3.2 – Speed and performances results as a function of input size (left), and AP25/AR25 as a 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.

Size	FPS	AP25/AR25	0.30	0.45	0.60	0.75	0.90
128^2	**490**	46/13					
256^2	260	**76**/60	69/59	**76/60**	69/54	65/50	64/50
512^2	80	55/**67**					
1024^2	20	28/63					

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 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 $Swimm^{400}$. In Table 3.2 (rightmost part), we observe that our optimum is fairly flat between 0.3 and 0.6, which proves the model's stability. The optimum value is 0.45.

3.4.3 Comparative Results

Also shown in Table 3.3, the Yolo model trained on 25,000 images by Woinoski *et al.* [139] gives results comparable to the tiny-U-Net 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-U-Net is extremely efficient in terms of scalability. Being designed to be trained with small datasets, it is quite shallow and can run extremely fast (see Table 3.2) 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 each swimmer in a race faster than in real-time, so faster than an expert by a huge margin.

Table 3.3 – 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 U-Net architecture gives significantly worse results compared to tiny-U-Net, as it overfits on the few data.

Model	AP 25	AP 50
Yolo (from [139])	70	-
Yolo	24	12
Yolo-tiny	31	20
U-Net	39	25
Tiny-U-Net	**76**	**60**

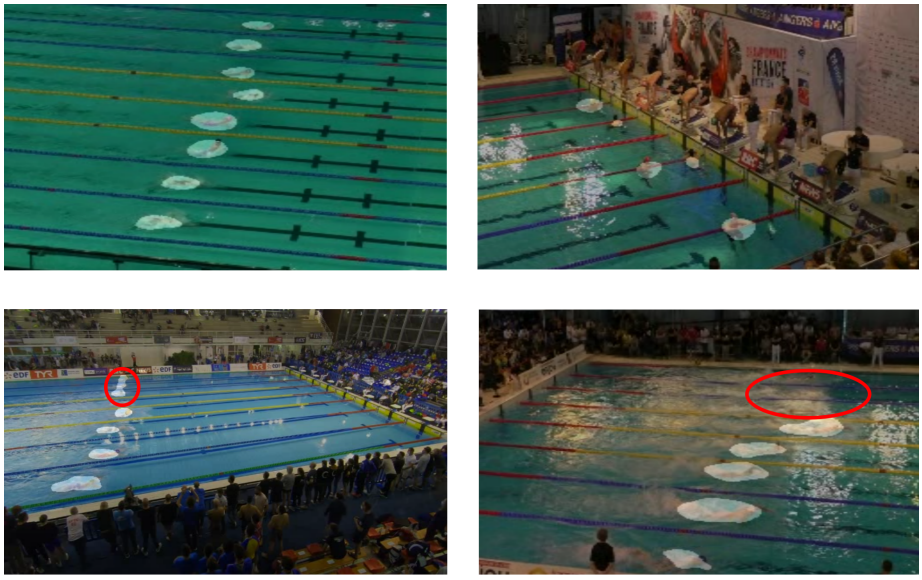


Figure 3.8 – The thresholded segmentation output overlaid on the input image with a size of (256 × 256) pixels. Circled in red are mistakes to focus on.

3.5 Visual and Qualitative results

We also provide a few visual examples of the results. These are not cherry-picked: they have been selected because they are representative of the global behaviour of our model.

3.5.1 Swimming Races

First, it is important to study the model in its normal context. Figure 3.8 displays a few of them, illustrating different behaviours 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

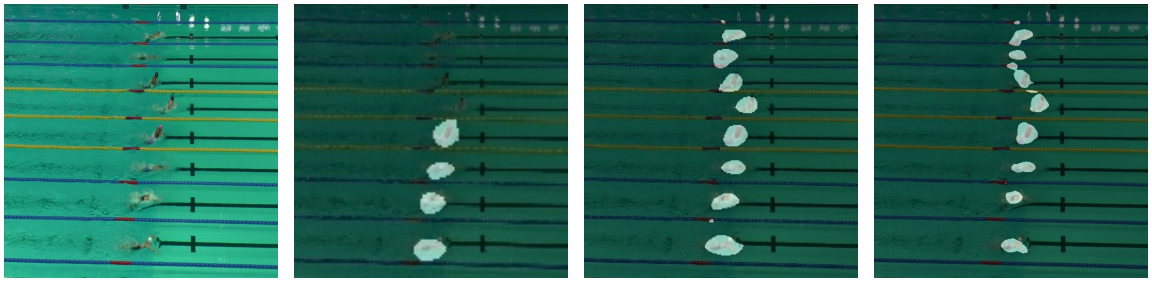


Figure 3.9 – Classic use-case image overlapping with the segmentation heatmap output by the model with different input sizes. From left to right, the input sides are 128 pixels, 256 pixels and 512 pixels.

the others. Although one could not state 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 is the lack of detection of swimmers about to dive, even if a few examples of such cases 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 people in the water who are not swimming. In our dataset, there is no occurrence of such "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 could be addressed by isolating each lane, as explained in a later chapter (see section 6.2.1). One could also segment the buoy lines and mask them out of the heatmap to divide the merged blobs. Finally, the bottom-right image shows the other side of the problem with swimmers too far from a too low camera: the lack of detection (*i.e.* false negatives). Usually, this is caused by a threshold that is too high, but lowering it would thus create false positives. Again, this will directly be addressed in a later chapter (section 6.2.1): 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.

In Table 3.2 is 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 3.9, 3.10 and 3.11 all highlight this phenomenon on their own domain. The general observation is that as the input grows, swimmer segmentation improves. In Figure 3.9, 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

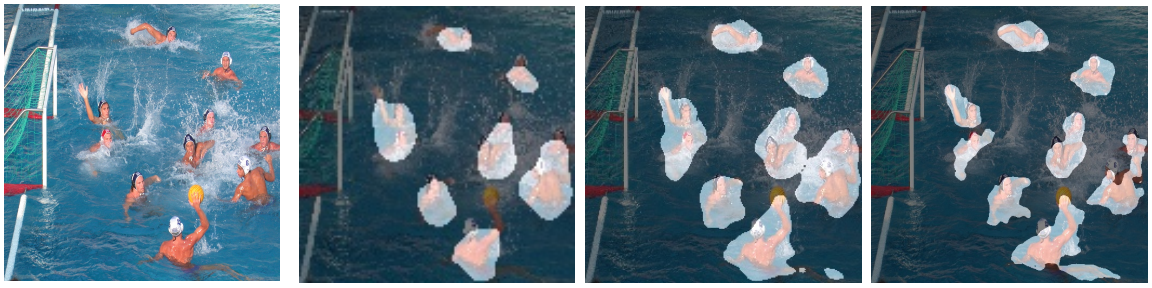


Figure 3.10 – Model Segmentation of a water-polo image, slightly out of the training domain. From left to right, the input sides are 128 pixels, 256 pixels and 512 pixels.

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 initial data of lower level than the output. One could arguably follow this lead 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 detected when they occupy a big enough part of the image. The opposite scenario arises with smaller images: 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.

3.5.2 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 3.10 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 being 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.

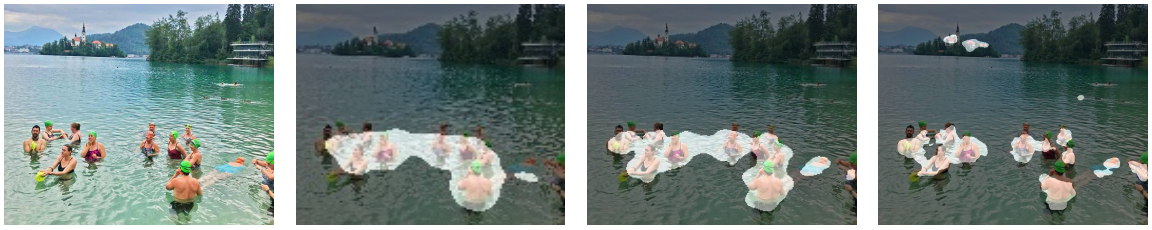


Figure 3.11 – Segmentation results of persons in a lake, which is significantly out of the training domain. From left to right, the input sides are 128 pixels, 256 pixels and 512 pixels.

One step can be considered missing, which is the blobs splitting. 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 elaborate further on this.

Finally, 3.11 shows an example result completely out of the training domain, as the persons are not swimming and not in a pool. This background and environment are 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 observations can be made. With low resolution, first, the group of close-enough persons is essentially well detected, with imprecise edges and some background water (*i.e.* false positives) segmentation. With higher resolution, though, results are more refined, the different blobs follow decently the persons' 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 a waste of time (or simply visual checking) than false negatives, *i.e.* we do not detect a drowning 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 can detect persons 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.

3.6 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.

3.6.1 Improvements and Future Works

The swimmers' coordinates output by the model 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 just lasts a second. To alleviate this problem, one could create another small dataset with only the swimmer head annotated. By training a model to detect the head only, we will obtain higher robustness. This might be incorporated into a 2-stage detector similar to Faster-RCNN [6]: 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 resizing and a new model inference. To implement a faster version, one could have a U-Net-like model with 2 decoder branches, following ProstAttention-Net ([142]). 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 3.8, is the merging of different blobs. 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 a refinement step. Another model would input only a crop around the blobs and it would have to segment them following a Gaussian distribution (instead of the current flat distribution). By only detecting the local maxima on this second step, instead of a threshold and blobs approach, we could differentiate the swimmers. As previously mentioned, Gaussian masks do not perform as well as binary masks for full-image detection. However, this refinement task is different as there are only a few swimmers to segment in a close space.

As seen in the previous section, the model also has trouble 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 an inherent part of Tiny-U-Net's nature. The network is shallow and simple by design, which inevitably creates this limit. However, this complexity / speed trade-off will

be addressed in a later chapter (section 6.2.1) 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 a low cost without large computational and human resources.

3.6.2 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 sports 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...).

3.7 Conclusion

This chapter proposes a method to detect swimmers in a frame with very few constraints. Its main advantage is its simplicity: it is easy to recreate 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, possibly freeing an enormous amount of time for coaches who performed this task mostly manually. It allows them to focus only on the athletic part of their role in a swimmer formation.

CHAPTER 4

POOL REGISTRATION

Contents

Chapter abstract

This chapter presents the problem of pool 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. It is complementary with detection to obtain higher-level information like the position and speed of swimmers. Existing methods usually first create a rough projection estimation and then use a 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. In particular, speed is important, mainly in the context of live broadcast

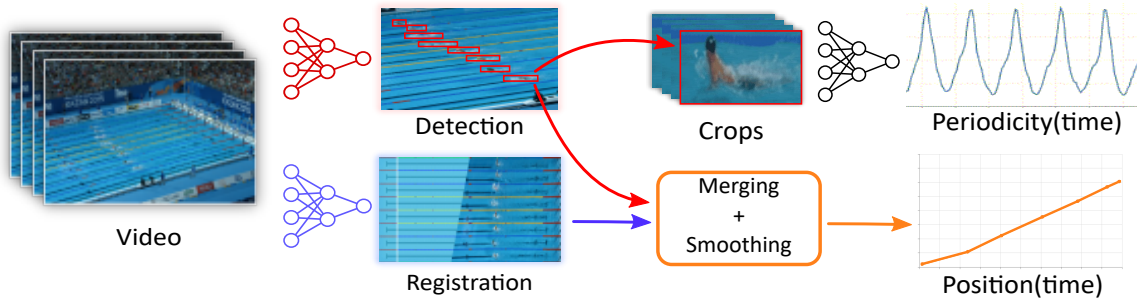


Figure 4.1 – The swimming race analysis method. The registration part is tackled in this chapter. It does not depend on any other method as it directly inputs the raw video frames. Used with swimmer detection (in the image), it gives their position in the pool and all the data coming from it (speed,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

TV and sports analysis. A new automatic field registration method is introduced in this chapter, achieving robust performance on the WorldCup Soccer benchmark, while neither depending on specific visible landmarks nor refinement steps, resulting in a very high execution speed and a good generalization. Finally, to complement the widely used 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, Romain Vuillemot, Stefan Duffner. *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. Its role in the automatic analysis pipeline is shown in Figure 4.1. As sports fields are planar and we consider lens distortions negligible, the registration is performed using a linear projection called homography.

Manual calibration is long (at least a dozen seconds 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 longer, thus an efficient solution for automatic field registration is crucial.

Automatic methods [143, 144, 145, 146, 147] 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 concave. Without a good initial estimation, no refinement step can 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 accurate results (see results in Section 4.5.2). 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 4.4, template). Our approach can be applied to any type of 2D sports field with TV streams or side stadium views. While maintaining high precision on the WorldCup Soccer benchmark [148], 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 for our application if we want athletes to get quick feedback on their performance shortly after a race (the other tasks also have to be quickly ready).

WorldCup Soccer benchmark [148] is the only public dataset that has been widely used in the literature, although some private datasets have been introduced for registration [145, 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 4.3) 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 sport and reaches high execution speed and State of the Art (SOTA) precision.

4.2 State Of The Art

This section describes how registration works. It first focuses on the creation of the homography matrix in practice, then presents manual approaches, and finally describes the SOTA for sports field registration.

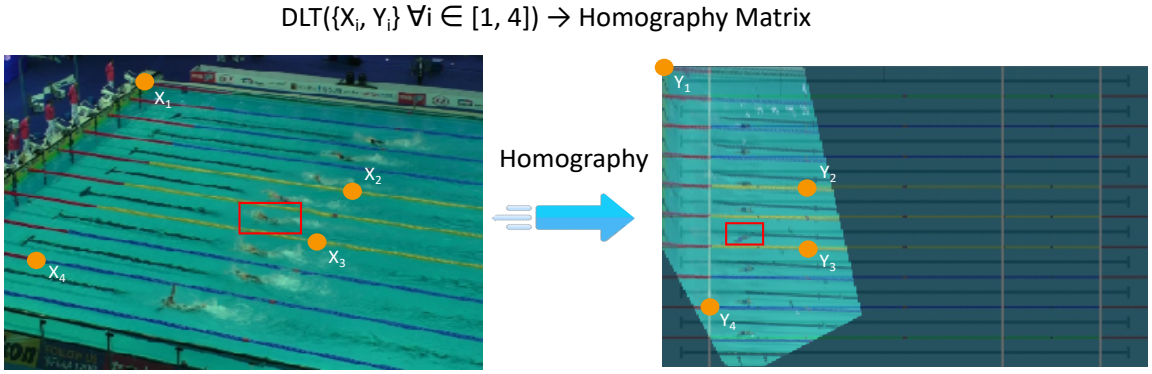


Figure 4.2 – An illustration of the registration. The Dynamic Linear Transform (DLT) operation uses known matches of positions (X_i with Y_i) to compute the homography matrix. This matrix can then be used to match a position from one coordinate system to the other. Therefore, the bounding box from the image can be positioned in the pool with this method.

4.2.1 Registration Background

In the context of registration, a pair of points is defined by a point on the source image (X) and a point in the absolute 3D coordinate system (Y), linked by the equation: $HX = Y$ 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 [149], as illustrated in Figure 4.2. 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 like RANSAC [150] to determine the most likely output.

In the case of homography matrix generation with many pairs, among which some errors, RANSAC behaves 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, then the new loss, *etc.*. 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 (thousands), which increases the robustness of stochastic algorithms as explained in [151].

4.2.2 Semi-Manual Approaches

Although associating pairs of points is a challenging task, associating points from a similar view is common using Scale Invariant Feature Transform (SIFT) [17] 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 is 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 [152, 153] in a similar way. 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 they would likely benefit from the newest approaches. Instead of relying on SIFT and RANSAC, newer methods [154, 155] input two subsequent frames and directly regress the matrix. The framework SuperPoint [156] 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 [154, 155]: as they are too different, no matching feature appears resulting in unusable output.

4.2.3 Recent Advances in Sport Field Registration

The first sports field registration methods [157, 158] relied on lines and circle detection using Hough Transforms [16]. The detected patterns were used as keypoints and, combined with RANSAC [150], enabled to compute a homography giving the absolute position of the camera view on a soccer field.

Using more recent deep learning approaches, fully automated robust methods appeared. Homayounfar *et al.* [148] 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 [143, 144, 145] 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. 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.* [147] used a model to directly estimate the image homography. They then used another one to refine the matrix by comparing the image and a template projected to the same point of view. Other approaches are based on field keypoint detection. Citraro *et al.* [159] 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 the whole 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

Compared to a soccer field, a swimming pool contains harder patterns to correctly identify and associate with a position on a pool. In Figure 4.3, 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 circled in red), but even in this case, they are distinct (with 4 different angles here). Further, the Soccer WorldCup dataset contains very few zoom variations and the camera is always placed close to the edge center. As such, it is always possible to distinguish enough landmarks to know without ambiguity the correct projection of the image in the field.

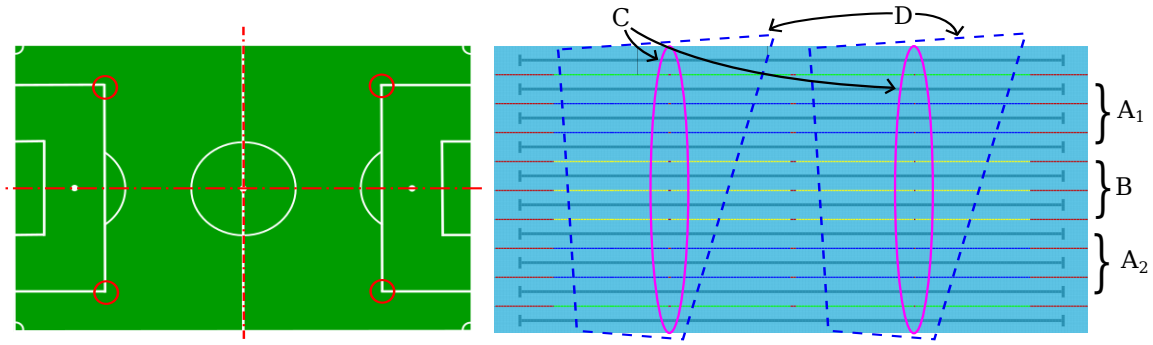


Figure 4.3 – Local appearance ambiguities comparison of a soccer field and a swimming pool. $A_{1,2}$: 4 identical lines at different places. B: 3 identical lines in the middle. Both A and B create line mismatch problems. C: the 15m and 35m markers are identical. D: an example of 2 different camera view projections on the pool that display the exact same content, despite being at two completely separate places of the pool. Best viewed in colour.

A pool, however, contains many more challenges (see Figure 4.3, right), namely positioning along the Y axis (A, B), positioning along the X axis (C, D) - both due to landmarks repetitions - and unstable background (*e.g.* wavelets, reflections, light saturation, 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.

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 captured by the Fédération Française de Natation (French Swimming Federation) (FFN) from the stands and their purpose is to frame the swimmers. They are included in the dataset 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

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

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 from a standard benchmark perspective, but it allows also sequential model evaluation. The dataset is available at https://github.com/njacquelin/sports_field_registration.

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 4.4. The main emphases of this work are computational efficiency and generalization. Other methods [145] 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 unremarkable, meaningless grass areas.

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 4.4, "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 4.4, "Depth").

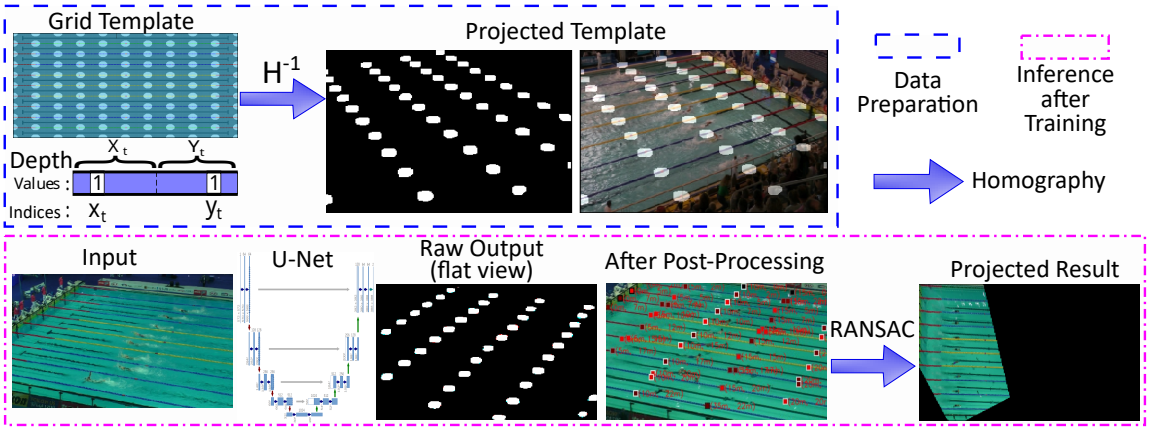


Figure 4.4 – Top: data preparation. A generic template with regularly spaced keypoints is created. The template's depth encodes the keypoints' position in the top-view frame. For each image in the dataset, a corresponding projection of the template is created. As a result, the landmarks spatially refer to their position in the image and semantically encode their position in the top-view coordinates system. Bottom: inference. The model generates a heatmap of keypoints. Using RANSAC, the homography matrix can be estimated, giving the final projection of the input image in the top-view frame. Best viewed in colour.

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 (Figure 4.4, "Projected Template"). With such projection, only semantic information has to be inferred.

Our approach relies on a U-Net architecture [61], 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 ensure that the keypoints are in 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

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 each channel to 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 $Homography\ Matrix$

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

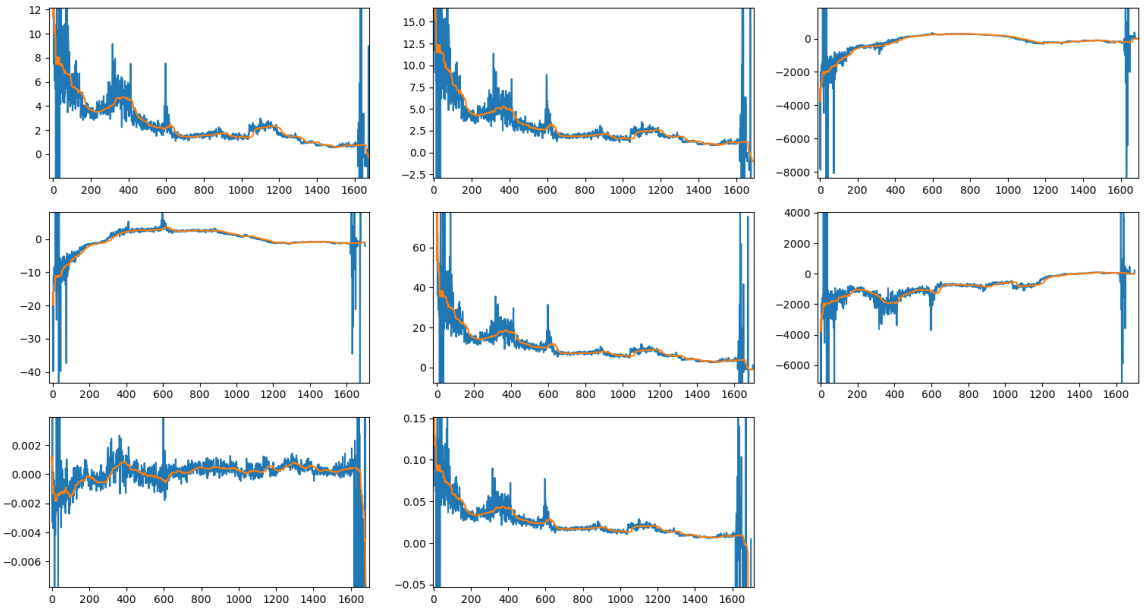


Figure 4.5 – Smoothing of the 8 parameters of the homography matrix estimated for a race (the parameter at position (3,3) is always equal 1). The values are represented through time. The outliers being orders of magnitude different from the truth, they have been cut out in the figure's frame. Blue: the original signal. Orange: the smoothed signal.

template. Based on these $((x^m, y^m), (x_t^m, y_t^m))$ pairs, RANSAC [150] can be used to compute the homography matrix. This is formally described in the Algorithm 4.1.

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, registration is inconsistent throughout the video. 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 4.5. 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 4.5. It would also be possible to complexify the smoothing process using the 2/3 Power Law [160], 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 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 an 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 post-processing methods is not possible. However, a qualitative appreciation of them on different registered videos is possible. Further, visualizations are presented in Figure 4.5 to showcase the interest of this post-processing. Imprecision in the homography estimation can be understood as noise on the parameter's 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 [29]. 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. In the literature, the standard 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 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 estimate its importance. We did not extensively search hyperparameters' precise values, 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 it represents the balance between the markers' classification loss and position loss. As the markers mask (serving to the position loss) is made of the maximum of each channel, it pushes all the channels to 1 at the places of interest. On the contrary, a unique channel must be activated for a low classification loss. Antagonist behaviours naturally emerge from them. It is thus important to know how to balance them.

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

With a value of $\lambda = 2$, the results are the best by a significant margin. 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 much 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 the image to be detected in the first 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 [161, 162]. Looking at the details, one can even argue that our model is faster than Sha *et al.* [145] 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 [163] to train a more condensed, shallower and faster version of U-Net. 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 4.3) 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

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

Method	Benchmark	IOU^{avg}_{part}	IOU^{med}_{part}	IOU^{avg}_{whole}	IOU^{med}_{whole}	FPS	Memory - GPU
Citraro *et al.* [159]	WorldCup	93.9	95.5	-	-	9	NA - Titan RTX
Sha *et al.* [145]	WorldCup	94.2	95.4	83.2	84.6	<u>250</u>	**48GB - Titan RTX**
Chen *et al.* [144]	WorldCup	94.5	96.1	89.4	**93.8**	2	16GB - NA
Jiang *et al.* [147]	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

mean and median results 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 4.3). 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 much more difficult than the standard Soccer WorldCup benchmark.

4.5.3 Failure Cases

In various situations, 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.

Wrong Classification

First, there are frames where a majority of the detected landmarks are not correctly classified. They are rarely 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 4.6, center.

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 4.6, 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

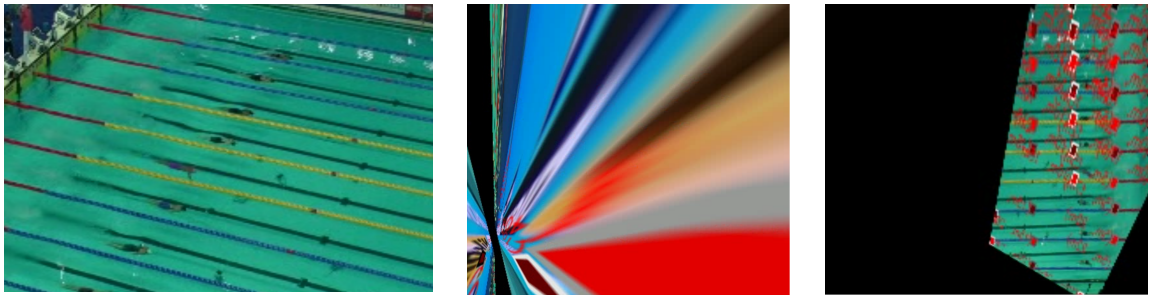


Figure 4.6 – Examples of failure cases. The results were obtained from two models trained under different conditions. The Red squares represent the detected landmarks. Despite the mirror failure looking consistent, its IOU with the ground truth is 0.

results or not, as it appears correct without the original image. It can be the cause of important errors.

General Geometric Misunderstanding

Both of these failure cases 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 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.

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 enough 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 sufficient. One can only say what the camera does *not* frame. To circumvent such problem, one must not zoom too much on the pool, 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 Computer Vision (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 dependent on the U-Net model we chose (here, the original one in [61]). 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 U-Net model's size to improve the results, if that were more critical than speed. One could even use a few refinement steps for 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.

To create and optimize a tool, one must consider its actual needs in precision and 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 an 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 field registration, which reaches very good performance and real-time inference speed. It is very well suited 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 speed in the context of such method was discussed. In CV, speed and performance are both important aspects to consider. Developing a tool, a user must always select which one to prioritize over the other.

CHAPTER 5

PERIODICITY

Contents

Chapter abstract

Knowing the position of swimmers in a pool allows one to study them specifically, and in particular to count their swimming strokes. To that extent, this chapter introduces a context-agnostic unsupervised method to count periodicities in videos. Current methods are limited to a specific type of application (e.g. *repetitive human motion). We propose a novel approach 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 Neural Network (NN) model that is trained completely unsupervised. More specifically, it is trained to transform the periodic temporal data into a 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 reliably detects and counts periods in complex time series. Despite being unsupervised and facing supervised methods with complex architectures,*

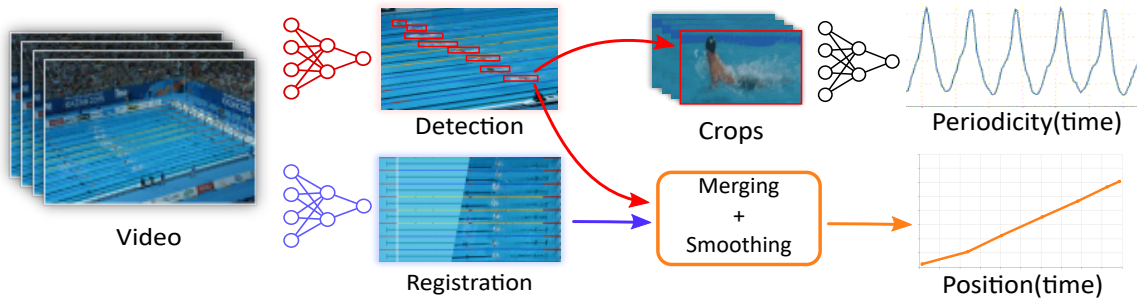


Figure 5.1 – The swimming race analysis pipeline. The periodicity counting part is tackled in this chapter. This part is built on top of swimmer detection because it relies on sub-video crops around swimmers. It can output the number of cycles per pool length or the duration of cycles. Both metrics are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

our experimental results demonstrate that our approach reaches State of the Art (SOTA) performance for periodicity counting on the challenging QUVA video benchmark. Its remaining limits will be addressed by an additional method based on supervised training and an annotated dataset. This chapter is based on the work from:

Nicolas Jacquelin, Romain Vuillemot, Stefan Duffner. *Periodicity Counting in Videos with Unsupervised Learning of Cyclic Embeddings*. Pattern Recognition Letters, Elsevier, 2022, (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 rate (defined in 5.5) is one of the most important metrics to determine a race quality and infer other statistics (*e.g.* stroke amplitude, rate etc.). Combined with the swimmers' position in the pool, it provides important analytical data to a coach. In the automatic swimming races analysis pipeline, it occurs after the detection, 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). 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 (*e.g.* the left and right arm strokes for freestlye). Furthermore, the notion of periodicity is context-dependent: the same event in two different

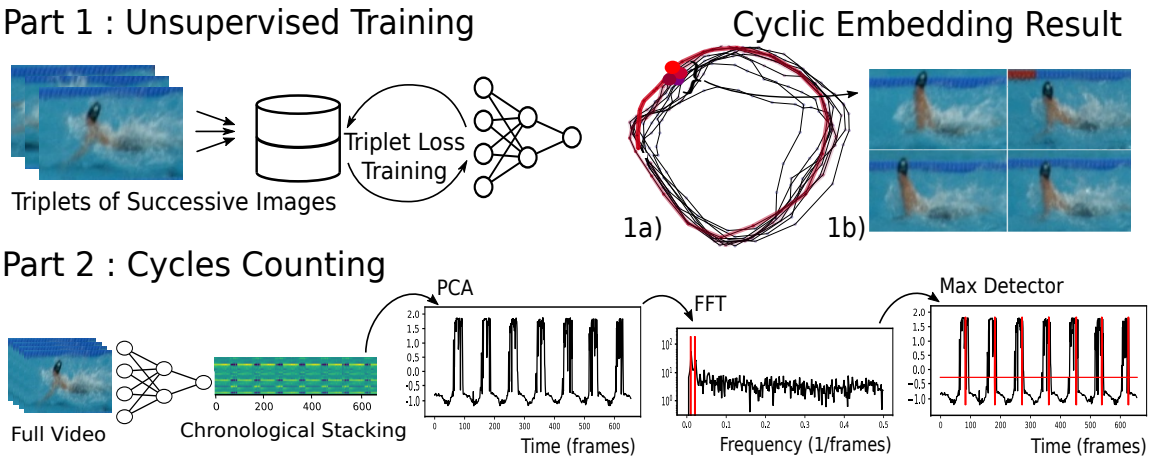


Figure 5.2 – The periodicity counting framework. In Part 1, a Convolutional Neural Network (CNN) is first trained in an unsupervised fashion on the test data, as described in section 5.3.1. Then, it is used to extract an embedding for each image of a video. (1a) shows an example of the 2D PCA projection 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.

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 spatial 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, those models often require large amounts of training data [53, 3, 164]. In the case of swimming periodicity counting, such dataset does not exist. Being able to adapt to any domain also prevents from using pre-training methods or context-specific datasets such a Kinetics [165], as many other techniques of the SOTA do [166, 167, 168, 169]. In particular, a NN trained in daily-life context does not transfer well in the new domain of a pool. 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 [170, 171, 172], 4D MRI videos (*i.e.* 3D images through time) of breathing lungs [173], active brains [174] or beating hearts [175]. For these reasons, it is important to have a domain-agnostic method.

This chapter presents a new domain-agnostic 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 multi-variate 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 testing different time scales or using 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 [166] or multimodal fusion models [168].

Further, during this thesis, we kicked off 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. Our main contributions in this chapter are the following:

- An unsupervised method to train a NN with the triplet loss to encode any kind of video showing a periodic phenomenon (section 5.3.1).
- A framework for automatic periodicity counting in videos, based on the analysis of a learnt embedding (section 5.3.2).
- a swimming periodicity dataset and a method to train a model on it (section 5.5).

5.2 Related Work

To specifically address periodicity counting in daily life videos, Levy and Wolf [176] 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) 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 [177] 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 [178] 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.* [179], introduced another method to convert a video into a 1D signal. They studied the main direction of the foreground's optical flow 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.* [166] proposed a complex architecture mixing CNN and transformers [90], trained in a fully-supervised fashion on the Countix dataset which they introduced. 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-dependent. Using the Countix dataset, Zhang *et al.* [168] proposed a multi-modal approach relying on sound and sight to improve the SOTA on Countix benchmark. They did no evaluation of it on QUVA, however.

The work of Yin *et al.* [169] 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 Kinetics [165], a large annotated video dataset, in a supervised way. As such, they can only analyze conventional videos of 2D images and the learnt visual features are domain dependent, 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.* [167] 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 [165], 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.* [180] 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 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.

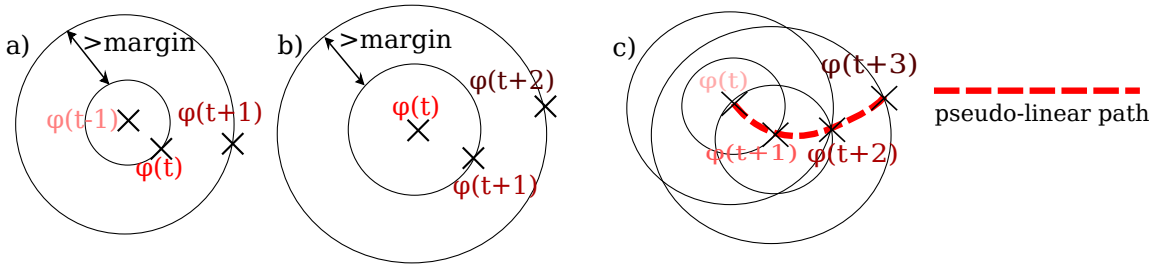


Figure 5.3 – Unsupervised learning of the pseudo-linear path using the triplet loss. The anchor is at the center, the positive is on the smaller circle (not necessarily the same size each time), and the negative is outside of the bigger circle. a) The anchor is $\phi(t-1)$, so $\phi(t)$ and $\phi(t+1)$ are separated. b) The anchor is $\phi(t)$, so $\phi(t)$ and $\phi(t+1)$ are drawn together. When the training starts, the negative can be on 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 both the attraction and repulsion constraints imposed by the loss.

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. For 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 described 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 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 the 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 "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 5.3. 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 [38]. 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 video into a latent space, 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.

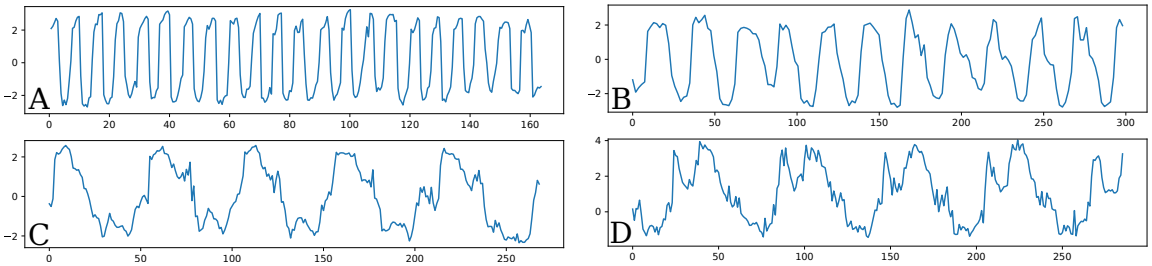


Figure 5.4 – Examples of 1D PCA projections of embeddings chronologically stacked. The A, B, and C curves show the embedding results for different cycles duration, from 8 to 50 frames per cycle (on average). D shows a more complex pattern containing 2 distinct local maxima. In such cases, our *Max Detector* might count 2 cycles per pattern, resulting in a false result, like mentioned in section 5.4.3.

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.

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 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 5.4, 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:

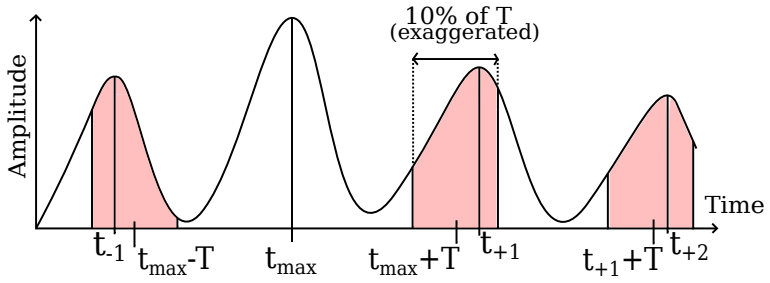


Figure 5.5 – Illustration of *Max Detector*. Starting from the global maximum's index t_{max}, the algorithm shifts by one period T and finds the maximum's index t_{+1} in a window of 10% of T (in red, exaggerated for a better understanding). This window makes *Max Detector* robust to period variations. 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 $t = 0$.

once forward towards the end, and once backwards 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 its 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 local maximum 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. This is a simple heuristic, but it proved to be efficient.

The list with the highest score is kept, whose number of elements represents 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 [179] and Countix [166] 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,

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 $MaxList$

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 [179], 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.

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 significantly 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.* [182]	0.307$\pm$ 0.408	0.51
flow + F=4	**0.231$\pm$ 0.326**	**0.64**

5.4.1 CNN Architecture

During our test phase, we did not notice a significant performance difference using different CNN architectures (we tried VGG19 and VGG11 [57], 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 [55], each layer doubling the number of filters (starting at 4, finishing at 128) and 2×2 max pooling [181] 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 [29]. Under these conditions, the training took about 1.1 times the total duration of a video using an 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.* [183], 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 [53] 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 anything else, especially the temporal input signal). The results were the same for the different splits: between 4 and 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 [182] 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, our unsupervised method gives an OBOA better than Zhang *et al.* [168] and is

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±σ ↓	Q: OBOA ↑	C: MAE±σ ↓	C: OBOA ↑
Levy and Wolf [176]		0.482± 0.615	0.45	-	-
Yin *et al.* [169]		**0.199** ± 0.335	-	-	-
Dwibedi *et al.* [166]		0.322	**0.66**	0.364	**0.697**
Zhang *et al.* [168]		-	-	**0.307**	0.511
Pogalin *et al* [184]	✓	0.389± 0.376	0.49	-	-
Runia *et al* [179]	✓	0.232± 0.344	0.62	-	-
Our method, F=4	✓	0.231± **0.326**	0.64	0.495 ± 0.769	0.517
Our method, F=2	✓	0.291± 0.445	0.59	0.419 ± 0.496	0.545

only outperformed by Dwibedi *et al.* [166]. 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", that are 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 exists, 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 methods' 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.

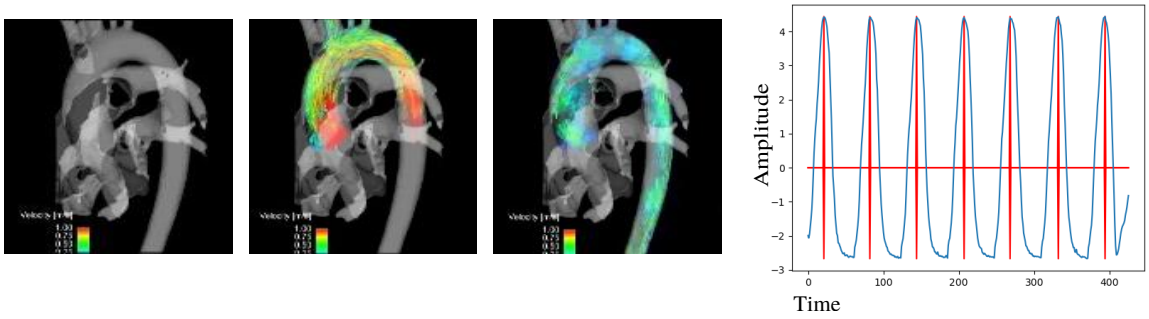


Figure 5.6 – 4D MRI video analyzed by our method. This is a proof of concept of the method's generalisation to different input types. Left: 2D slices of 3D input images (for display purposes) at different moments. The blood pulses through the artery. Right: the 1D PCA (blue) and peak detection of our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

However, SOTA 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 [185], 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 generalizes well to other types of data.

5.5 Going Further with Supervision

In our first discussions with the Fédération Française de Natation (French Swimming Federation) (FFN), 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

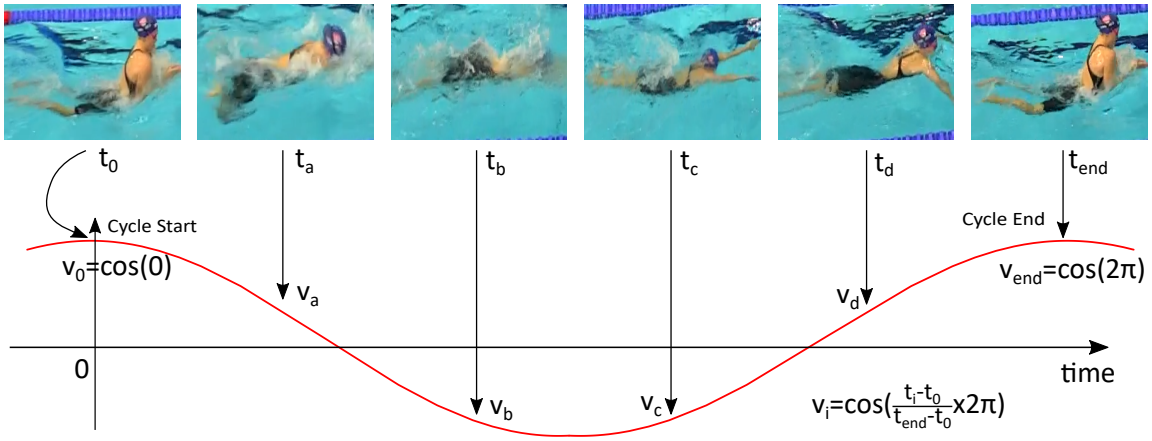


Figure 5.7 – Value associated to each frame according to their phase in the cycle. The arrow point at the value $v \in [-1, 1]$ taken by the frame on the sinusoid. Both extremity frames are associated to 1.

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 using a specific 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 Swimmer Strokes Detection

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 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 members of the Neptune project. Coaches analyzed the filmed races and we were able to synchronize their data with our video. It was therefore possible to create a labelled swimming strokes dataset with no extra annotation. Furthermore, as long as the FFN keeps analyzing videos filmed by our team, we can incorporate their data and 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 be added to 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 framerates between 25 and 60. In both cases, the proportion of frames associated with the value 1 is under-represented, despite arguably 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 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 in a good direction.

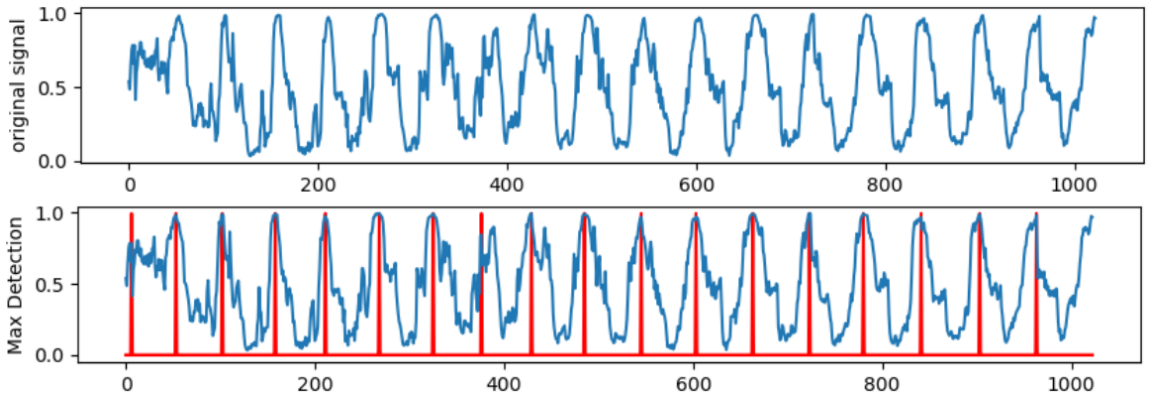


Figure 5.8 – Crops of a race around a swimmer, analyzed by our periodicity regression model. Top: the raw output signal. Bottom, maxima detection by the *Max Detector* algorithm. As maxima correspond to cycles extremities, the algorithm outputs the cycles extremities' timecodes.

5.6 Discussion and Perspectives

The methods proposed in this chapter show promising results for our swimming application, but also for general periodicity counting in videos.

Good Unsupervised Results

Despite being unsupervised and based on a shallow model, our method gives results comparable to SOTA 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.

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.

Important Limitation to Address

Periodicity counting in the case of swimming, however, has an important limitation that cannot be addressed by any of the two described methods. It is the dependency on swimmer 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 crop 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 crop 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 tasks, computation-wise: it is an order of magnitude slower than a U-Net model inference. As such, periodicity counting based on crop extraction is the temporal bottleneck of the pipeline.

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 swimmer 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) U-Net-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.

5.7 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 directly done on the test data. We believe that such an unsupervised approach may be of increasing importance in the future for different applications, to reduce the need for big datasets and complex architectures.

CHAPTER 6

CONCLUSION AND PERSPECTIVES

Contents

Chapter abstract

This chapter sums up the different contributions we presented in this manuscript. They have their individual limitations that we discuss here. We also describe an important future work that aims at improving the methods and reducing their limits by combining them all. This is a proposition of strategy for fully automating the video analysis. We also address more general perspectives and limitations of the thesis. Unaddressed challenges are mentioned, with leads to tackle them.

6.1 Summary of the Contributions

This thesis introduced multiple contributions for automatic analysis of swimming race videos. They brought different domains of Computer Vision (CV) together, like object detection / segmentation, registration, and unsupervised

learning. Two datasets were also introduced to help the community moving forward and creating new methods.

Swimmer Detection from Unconstrained Videos

The first contribution we made tackled swimmer detection in [8]. The $Swimm^{400}$ dataset, also presented in the paper, contains 400 labelled images of swimmers with around 3100 bounding boxes. To the best of our knowledge, no publicly available swimmer detection dataset existed when $Swimm^{400}$ was released and we hope is will push the sports analysis community forward. The dataset is varied, containing several environment, pools, viewpoints, angles, *etc.*. It can be used for swimmer detection without external constraints like camera positioning or type of pool.

$Swimm^{400}$is well-crafted, but it is orders of magnitude smaller than the usual detection datasets [3, 53]. Consequentially, classic object detection techniques [5, 132] do not perform well on it. Swimmer detection is thus addressed using a variation of the classic segmentation architecture U-Net. Our tiny-U-Net 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.

Real-Time Pool Registration

Our second contribution addresses automatic sports field registration techniques, in [9]. This task is also fundamental for swimming analysis, as once it is combined with swimmer detection, it allows the positioning of swimmers in the pool. Automatically addressing this task gives more adaptability to our analyses, as we do not have to rely exclusively on static videos that must be manually calibrated. As such, videos filmed before this thesis 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 matrix 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. Thus, it enables to train more adaptable models, which do not have to rely on specific filming properties.

The model relies on the U-Net 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).

Periodicity Estimation on Cropped Videos

In this thesis, we also studied periodicity counting and strokes end detection in chapter 5 and in [10]. 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 video. 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.

6.2 Limitations and Proposed Solutions

The methods we presented in this thesis could be improved, as explained in their respective chapter. The way the videos are filmed also presents problems for the later analysis. In this section, we address their limitations and present ideas of solution.

6.2.1 Benefits from Combining the Models

Some performance limitations of our models could directly be addressed by putting them together. By integrating them inside of a unified system, we could maximize their positive interactions in order to increase precision. Figure 6.1 illustrates an idea of such a unification (note that it changed compared to the previous chapters).

Registration on a Race Video

The current registration model is not accurate enough to give useable results if the cameras are panning and following the swimmers. Indeed, it needs spatial context, which can only be obtained by filming at least half of the pool at all time. For this purpose, the Neptune project uses two static cameras each filming half of

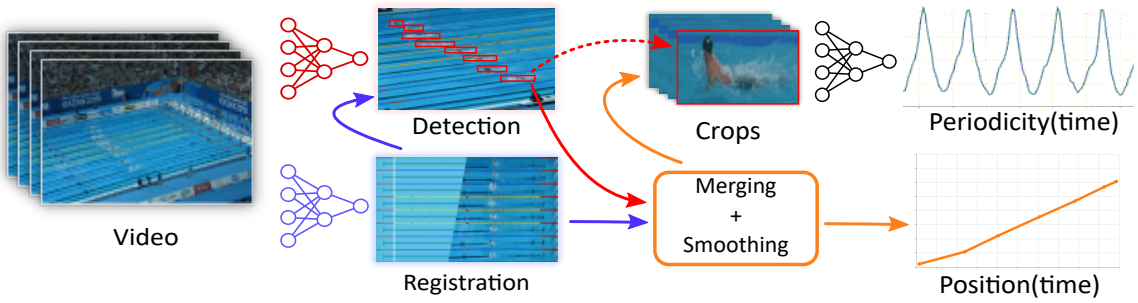


Figure 6.1 – A final illustration of our swimming models unified in a single system. The dotted arrow is a connection that would no longer exist with the proposed architecture. Following this idea, the registration result would be part of the detection process, and the smoothing / merging bloc would improve on the crops extraction.

the pool. However, even then, the model gives different results in function of the input frame, as shown in Figure 6.2. This is likely due to small variations from the waves and the reflections that change the local distribution of pixels. This creates important coherence issues, as the same position on two frames from a static camera would be mapped to different places in the pool.

We could circumvent the problem by finding a single matrix that is used for the entire video. To accelerate the process, we could only apply our registration model on a few frames. The challenge is to find the best homography matrix among several that would be estimated by the model. We could use DBSCAN [186] to cluster them. We can assume that although many images can have a wrong registration, they tend to fail in different ways (see the bottom line of Figure 6.2). However, as there is only one way to be correct, we could find the optimal solution by simply selecting the most numerous cluster of homography matrices. We would thus obtain one group of similar homography matrices (good regression) and several significantly smaller groups of very different matrices (failure cases). This would alleviate the weakness of the registration model. Within the selected group, each parameter could be averaged to get the optimal homography matrix.

An extra module could also be added to judge the quality of registration. 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. If the result on a video were below a threshold, this would trigger a warning addressed to the user. The threshold could be manually defined by the user depending on the level of precision they want.

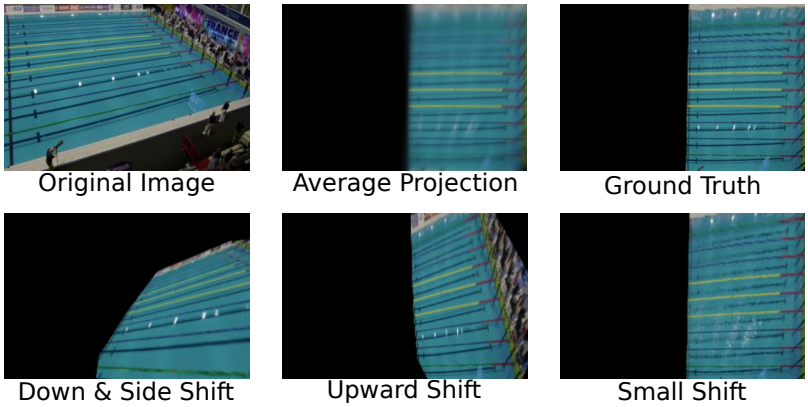


Figure 6.2 – Top-views obtained using the same model from a static camera filming a race. Different types of visible errors exist. There are also many small shifts, almost indistinguishable from the ground truth, causing flickering to the video. Both the raw projection and the clustered-based projection (using a unique matrix) of the video are available at: https://drive.google.com/drive/folders/1oXKgDIzy3vTd0UHE9TaFjyozOeiR9gTt?usp=sharing.

Detection Aided by Registration

We could use a new detection pipeline as illustrated in Figure 6.3, relying on the top-view to detect the blobs on the heatmap. With this idea, we would first reconstruct the entire pool using the two camera's registration, and isolate the different swimming lanes. In these lanes, we could locate the blob of highest probability. This would alleviate the weak detection on the top (*i.e.* farthest) lines, where a threshold of 0.45 (optimal as shown in chapter 3) is too high for this specific area. The problem of swimmers' blobs merging would also be addressed by the lanes' separate analysis.

A second-stage swimmer detector could be added to confirm that a given blob actually contains a swimmer. It could be used to select the best blob of a lane as an alternate to the heuristic of the blob with highest probability. However, this would increase computation time significantly, as the blobs' area should be extracted and resized, and then fed to the model.

A post-processing step could finally be added to refine the swimmers' positioning. With a unique swimmer detected on each frame in each lane, their position through time becomes obtainable. It could be smoothed out and outliers could be removed using simple signal processing methods. This would result in a monotonously growing curve, corresponding to the position of the swimmer starting at 0m and finishing at 50m.

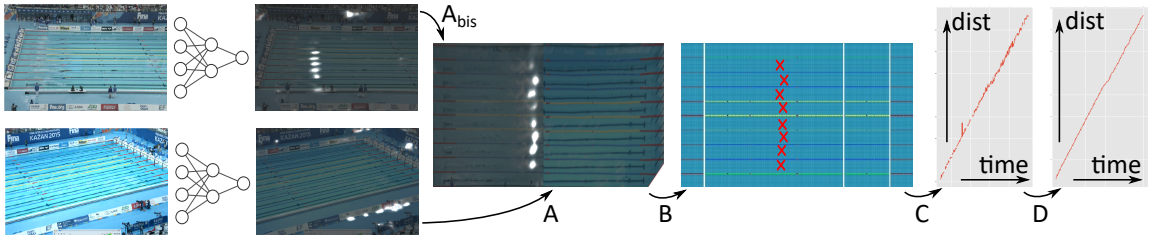


Figure 6.3 – The detection pipeline proposed in this section. After the model inference on the right and left videos, there are 4 steps. A: top-view projection and fusion of the raw heatmaps. B: by-lane threshold, resulting in a position for each swimmer for a given frame. C: temporal aggregation of the positions through the entire race. D: position smoothing and gap-filling giving the trajectory of each swimmer through time. Note that although the probability maximum of the topmost swimmer's blob is under 0.45, our by-lane threshold can detect it. The detection false positives are transparently discarded thanks to the use of registration.

Stabler Crops for Periodicity Estimation

A major limitation of periodicity counting is its need for robust swimmer crops extraction. We could use the smoothed positioning coming from the combination of detection and registration to extract stable, not shaking sub-videos around each swimmer. The periodicity model would thus have almost no external perturbation and could function optimally.

6.2.2 Increasing the Acquisition Speed

Before analyzing a video, one must have access to it, which is slow using the current way the races are filmed during competitions. It could be changed to address real-time (or close) constraints, and allow for faster feedback to the swimmers.

Towards a Real-Time Video Analysis Acquisition Process

Currently, cameras film different races and their memory cards are regularly switched with empty ones. The full cards are moved to the Fédération Française de Natation (French Swimming Federation) (FFN) performance division cell and their content is uploaded to a computer where it can finally be analyzed. In both cases, there is a delay between a race and the moment an analysis can be completed. This filming 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 processing computer. The analysis could thus start during the race and be over soon after it, as the method can operate fast. It would allow

for direct feed-backs from the analysis system to the coaches and swimmers, but also from them to us, to improve our tool. Indeed, such faster speed to obtain results would encourage both sides to interact more, which is mutually beneficial.

Acquisition Resolution

Races are shot in 4K. However both image loading in memory time and image resizing time increases with input size. Filming images of lower resolution, such as 720p, would decrease substantially this time. As the images fed to the models are resized to (256×256) pixels anyway (as shown in Table 3.2 in chapter 3), it would not change the analysis accuracy. For periodicity counting, videos cropped around a swimmer do not require 4K to perform well. If, in the future, other models needing a close-up of the swimmer (*e.g.* for swimmer pose estimation) are used, higher quality videos may be required. For now though, it is not the case.

6.3 Perspectives and New Challenges

This work proposed different new methods, ideas, models, and datasets. As such, it unlocked some domains of swimming analysis, that can be explored in future works. There are also domains we could have explored, that we ended up not addressing by lack of time. This section describes what these works could be and what they would have brought.

6.3.1 Guided Annotation Tool

Creating an extensive dataset is time consuming, even with a good labelling tool. To alleviate this problem, we worked on a tool of our own to help swimmer annotation, illustrated in Figure 6.4. This tool relies on an already trained detection model, like the one presented in chapter 3. It works like a regular annotation tool, but bounding boxes are already placed by the model. The user could quickly adjust these boxes if they are almost correct, delete the false positive, and create new boxes. During the conception of this tool, our priority was labelling speed, as it is the most cumbersome problem of annotation. Other than that, the tool's key feature concerned the threshold of the model, which has a different optimum in function of the level of zoom and other similar variations. With distant swimmers, it is generally beneficial to have a low threshold. Therefore, while annotating a race distant from the camera, the user could set the threshold to a low value 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

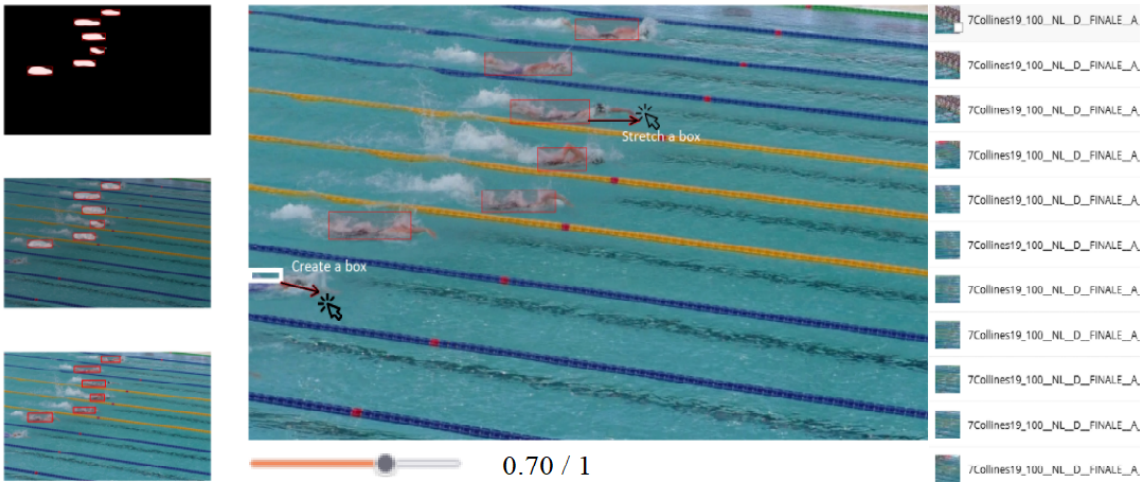


Figure 6.4 – An illustration of the annotation tool we built, with bounding boxes already generates by the model (in red) and bounding boxes created from scratch by the user (in white). On the left, different views of the detection. The threshold slider is at the bottom.

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.

6.3.2 Temporal Data for a Better Context Understanding

In this thesis, detection and registration were based on models performing frame-by-frame operations, but no temporal data were used. One could enhance them using either past or future frames to detect the swimmers. Such approach would probably give significantly better results, as it is not rare that waves and diffraction temporally occlude swimmers, making them almost impossible to detect using only one frame. A temporal approach could also be used for registration. Some points of view lack information to allow for robust and confident homography estimation. However, using the landmarks visible in the previous frames would circumvent this problem. Freeing us from the constraint of static videos would significantly increase the generalisation of the model.

This could be implemented by feeding the models several successive images stacked together instead of only one. The first layers of the model could be replaced with 3D convolutions to better handle this new input. No extra annotation would be required, as the output could still concern only one frame. However,

we would have to find the frames preceeding the annotate ones in the videos that served to create our datasets.

6.3.3 More Data for Better Models

Additional data can be created by labelling fix videos as they are captured in competition with static cameras. New data are always appreciated by the community and they can increase substantially the performance on different tasks. In particular, u-turns and underwater swimmers could be better detected than they currently are as they are not very frequent in $Swimm^{400}$. Therefore, the most interesting images to label are both angles more similar to our use context and "rare" swimmers positions.

6.3.4 Weakly Supervised Learning for a Multitask Model

The model fed with crops of swimmers giving information on their phase could be extended using weakly-supervised learning algorithms. Instead of crops, whole images could be the new input and the model would give phase information about only the areas with swimmers. If a model like this were created, it would unify swimmer detection with periodicity analysis. In fact, the three models could be merged into a single general U-Net architecture outputting registration, detection, and periodicity information all at once. This would increase speed by merging the different models and removing the costly crop operations.

6.3.5 Swimmers Pose Estimation

Swimmers pose estimation was not addressed at all during this thesis as many intermediate works were required before it. However, based on our existing swimmer detection method and on progresses in the domain, it could be feasible in the near future. However one must be realistic about the limitation of underwater joints detection in uncontrolled filming condition. 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.

6.3.6 Vision Transformers

Vision Transformer architectures [65] might give better results for registration tasks. One could try using them instead of Convolutional Neural Networks (CNNs). Indeed, this task relies on landmarks that must be identified in context with the entire image. As transformer models are considered better than CNNs to

study images globally, this could lead to better results. One could also use transformers to study the videos as sequences and introduce continuity in the analysis. However, this architecture currently needs significantly more data than CNNs [187], even though recent advances reduce this limitation [188] (at heavy computational costs). As the positioning of our works concerns small datasets and computationally light methods, transformers are incompatible.

6.3.7 Unaddressed Challenges

There are tasks mentioned in the Introduction 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 our automatic analysis method. These tasks are swimming phase segmentation, swimmer identification, and cameras temporal synchronization. They are developed in this section.

Swimming Phase Segmentation

Phase segmentation consists in knowing what specific action the swimmer is performing (diving, normal, swimming, u-turning, *etc.*). Currently, when a swimmer is detected, we only extract crops to study their periodicity. The model does not informs if the swimmer is breathing, or if they are touching the edge of the pool, although these are very important analytics. We could create a new model that would be fed crops around the swimmers and output the swimming phase. Transfer learning based on the swimming periodicity model can be used to train it with few data. This would be a classification task where the states (*i.e.* classes) can comprise diving, breathing, touching the edge, the swimming style, and other aspects of a swimmer that can be extracted from a crop.

Swimmer Identification

Identifying a swimmer during a race is very challenging, especially with videos that are not particularly zoomed-in on them. Face recognition is irrelevant, as the swimmer's head is underwater most of the time. One could however use classification techniques that would input the entire body. The swimmer identification would thus not concern only face identification, but the body shape and general swimming look. This method was illustrated in chapter 2 with Figure 2.17.

Cameras Temporal Synchronization

As we propose to film the pool with two cameras, they must be temporally synchronized to create consistent analyses. To that extent, we could rely on the

audio. The soundtracks from the cameras could be correlated with each other: the highest peack should correspond to the time shift between the two recordings.

6.3.8 MediaEval Challenge

Finally, we wanted to call for the CV community to help with the models. Combining the ideas from multiple sources, each improving on the others, is a great improvement perspective. To that extent, we published a swimming image analysis challenge at the 2022 MediaEval Workshop:

Nicolas Jacquelin, Théo Jaunet, Romain Vuillemot, Stefan Duffner. *SwimTrack: Swimmers and Stroke Rate Detection in Elite Race Videos*. Working Notes Proceedings of the MediaEval 2022 Workshop.

It introduces a dataset created by merging the FFN analysis data with videos we recorded, manually calibrated, and synchronized. Coaches thus provide analytics in the pool coordinates system and we project them back into our videos to create rich annotations. This allows us to create temporal data for swimming detection and periodicity counting. We also have access to the official times and ranking of the races. The resulting challenge we created addresses these tasks, listed as follows:

- **Swimmer Tracking**: the objective is to find swimmers an image. Contrarily to the dataset *Swimm*[400], here, the head is labelled and there are temporal data, thus a tracking model can be used.
- **Strokes Detection**: the input is a video cropped around a swimmer and the task is to tell a which the frames the cycles end.
- **Pool Registration**: using the previously introduced RegiSwim[500].
- **Score Boards Reading**: images of score boards are provided. The purpose is to read them, that is to output the name, ranking, and time corresponding to each line.
- **Start Buzzer Detection**: the audio from different races have been recorded and the challenge is to detect the start buzzer, which is labelled with a $1/100^{th}$ of a second precision.

Although we already addressed the three first tasks, the new source, nature, and amount of data offer new opportunities of technical innovation. More information are available at https://multimediaeval.github.io/editions/2022/tasks/swimtrack/.

6.4 Conclusion

This thesis tackled the challenge of automatically analyzing swimming races using videos. The created methods have 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 method was divided in 3 tasks: detection, registration, and periodicity counting. This resulted in different models, each with its strengths and weaknesses. We hope it will be extended and used by swimmers and coaches.

BIBLIOGRAPHY

[1] Internet. *Syndrome de Stokholm*. Wikipedia. 2022. URL: https://fr.wikipedia.org/wiki/Syndrome_de_Stockholm (cit. on p. vii).

[2] Internet. *Influence Sociale*. Wikipedia. 2022. URL: https://fr.wikipedia.org/wiki/Influence_sociale (cit. on p. vii).

[3] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV] (cit. on pp. 5, 32, 34, 46, 83, 102).

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html (cit. on pp. 5, 32–34).

[5] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV] (cit. on pp. 5, 46, 49, 51, 52, 57, 102).

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587 (cit. on pp. 5, 27, 46, 51, 52, 63).

[7] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel M. Ni, and Heung-Yeung Shum. *DINO: DETR with Improved DeNoising Anchor Boxes for End-to-End Object Detection*. 2022. URL: https://arxiv.org/abs/2203.03605 (cit. on pp. 5, 43).

[8] Nicolas Jacquelin, Stefan Duffner, and Romain Vuillemot. "Detecting Swimmers in Unconstrained Videos with Few Training Data". In: *Machine Learning and Data Mining for Sports Analytics*. Ghand, Belgium, Sept. 2021. URL: https://hal.archives-ouvertes.fr/hal-03358375 (cit. on pp. 6, 102).

[9] Nicolas Jacquelin, Romain Vuillemot, and Stefan Duffner. "Efficient One-Shot Sports Field Image Registration with Arbitrary Keypoint Segmentation". In: *IEEE International Conference on Image Processing*. Bordeaux, France, Oct. 2022. URL: https://hal.archives-ouvertes.fr/hal-03738153 (cit. on pp. 6, 102).

[10] Nicolas Jacquelin, Romain Vuillemot, and Stefan Duffner. "Periodicity Counting in Videos with Unsupervised Learning of Cyclic Embeddings". In: *Pattern Recognition Letters* (2022). URL: https://hal.archives-ouvertes.fr/hal-03738161 (cit. on pp. 6, 103).

[11] Mark Hitchman. *From AI-assisted imaging to AI-assisted diagnosis*. medtechnews. 2022. URL: https://www.med-technews.com/medtech-insights/ai-in-healthcare-insights/from-ai-assisted-imaging-to-ai-assisted-diagnosis/ (cit. on p. 7).

[12] Djamel Benarab, Thibault Napoléon, Ayman Alfalou, Antoine Verney, and Philippe Hellard. "Optimized swimmer tracking system based on a novel multi-related-targets approach". In: *Optics and Lasers in Engineering* 89 (May 2016) (cit. on pp. 9, 50).

[13] David Napoleon Simbana Escobar. "Variabilité de la technique de nage : adaptabilité aux contraintes et performances en natation". Theses. Normandie Université, Feb. 2018. URL: https://tel.archives-ouvertes.fr/tel-01793215 (cit. on p. 9).

[14] David Marr and E. Hildreth. "Theory of Edge Detection". In: *Proceedings of the Royal Society of London Series B* 207 (1980), pp. 187–217 (cit. on p. 9).

[15] John Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986), pp. 679–698 (cit. on p. 11).

[16] Richard O Duda and Peter E Hart. "Use of the Hough transformation to detect lines and curves in pictures". In: *Communications of the ACM* 15.1 (1972), pp. 11–15 (cit. on pp. 11, 70).

[17] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157 (cit. on pp. 13, 69).

[18] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: Speeded Up Robust Features". In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof, and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417 (cit. on p. 13).

[19] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571 (cit. on p. 13).

[20] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. "BRIEF: Binary Robust Independent Elementary Features". In: *Computer Vision – ECCV 2010*. Ed. by Kostas Daniilidis, Petros Maragos, and Nikos Paragios. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792 (cit. on p. 13).

[21] Jan C. van Gemert, Cor J. Veenman, Arnold W.M. Smeulders, and Jan-Mark Geusebroek. "Visual Word Ambiguity". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.7 (2010), pp. 1271–1283 (cit. on p. 14).

[22] Marco Mangini and Irving Biederman. "Making the ineffable explicit: Estimating the information employed for face classifications". In: *Cognitive Science* 28 (Mar. 2004), pp. 209–226 (cit. on p. 15).

[23] Di Liu, Dong-mei Sun, and Zheng-ding Qiu. "Bag-of-Words Vector Quantization Based Face Identification". In: *2009 Second International Symposium on Electronic Commerce and Security*. Vol. 2. 2009, pp. 29–33 (cit. on p. 15).

[24] Carl Vondrick, Hamed Pirsiavash, Aude Oliva, and Antonio Torralba. "Learning visual biases from human imagination". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015 (cit. on p. 15).

[25] Simone Fabbrizzi, Symeon Papadopoulos, Eirini Ntoutsi, and Ioannis Kompatsiaris. *A Survey on Bias in Visual Datasets*. 2021. URL: https://arxiv.org/abs/2107.07919 (cit. on p. 15).

[26] F. Rosenblatt. *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory, Jan. 1957 (cit. on p. 15).

[27] CAUCHY A. "Methode generale pour la resolution des systemes d'equations simultanees". In: *C.R. Acad. Sci. Paris* 25 (1847), pp. 536–538 (cit. on p. 16).

[28] J. Kiefer and J. Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function". In: *The Annals of Mathematical Statistics* 23.3 (1952), pp. 462 –466 (cit. on p. 16).

[29] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980 (cit. on pp. 16, 76, 91).

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536 (cit. on p. 17).

[31] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on pp. 17, 20).

[32] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48 (cit. on p. 17).

[33] Ibrahem Kandel and Mauro Castelli. "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset". In: *ICT Express* 6.4 (2020), pp. 312–315. URL: https://www.sciencedirect.com/science/article/pii/S2405959519303455 (cit. on p. 17).

[34] *How to Control the Stability of Training Neural Networks With the Batch Size*. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (cit. on p. 17).

[35] Tianlu Wang, Jieyu Zhao, Mark Yatskar, Kai-Wei Chang, and Vicente Ordonez. "Balanced Datasets Are Not Enough: Estimating and Mitigating Gender Bias in Deep Image Representations". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019 (cit. on p. 18).

[36] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. *A Survey on Bias and Fairness in Machine Learning*. 2019. URL: https://arxiv.org/abs/1908.09635 (cit. on p. 19).

[37] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016. URL: https://arxiv.org/abs/1602.04938 (cit. on p. 19).

[38] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. "Shortcut learning in deep neural networks". In: *Nature Machine Intelligence* 2.11 (Nov. 2020), pp. 665–673. URL: https://doi.org/10.1038%2Fs42256-020-00257-z (cit. on pp. 19, 87).

[39] Suorong Yang, Weikang Xiao, Mengcheng Zhang, Suhan Guo, Jian Zhao, and Furao Shen. *Image Data Augmentation for Deep Learning: A Survey*. 2022. URL: https://arxiv.org/abs/2204.08610 (cit. on p. 20).

[40] Sei Miyake and Kunihiko Fukushima. "Self-Organizing Neural Networks with the Mechanism of Feedback Information Processing". In: *Dynamic Interactions in Neural Networks: Models and Data*. Berlin, Heidelberg: Springer-Verlag, 1988, 107–119 (cit. on p. 20).

[41] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202 (cit. on p. 20).

[42] K. Fukushima. "A Hierarchical Neural Network Model for Selective Attention". In: *Neural Computers*. Berlin, Heidelberg: Springer-Verlag, 1989, 81–90 (cit. on p. 20).

[43] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551 (cit. on p. 20).

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778 (cit. on pp. 20, 25, 29, 30).

[45] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners*. 2020. URL: https://arxiv.org/abs/2005.14165 (cit. on pp. 21, 34).

[46] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: *Distill* (2017). https://distill.pub/2017/feature-visualization (cit. on p. 22).

[47] Desh Raj. *Theory of Deep Learning: Role of Depth*. github. 2018. URL: https://desh2608.github.io/2018-07-28-deep-learning-theory-3/ (cit. on p. 22).

[48] A G Ivakhnenko and V G Lapa. In: *Cybernetic predicting devices*. Ed. by New York: CCM Information Corp. 1965 (cit. on p. 22).

[49] Yann LeCun. *Convolutional Network Demo from 1993*. Youtube. 2014. URL: https://www.youtube.com/watch?v=FwFduRA_L6Q (cit. on p. 22).

[50] Tze-Yui Ho, Ping-Man Lam, and Chi-Sing Leung. "Parallelization of Cellular Neural Networks on GPU". In: *Pattern Recogn.* 41.8 (Aug. 2008), 2684–2692. URL: https://doi.org/10.1016/j.patcog.2008.01.018 (cit. on p. 22).

[51] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142 (cit. on pp. 22, 32).

[52] Peter W. Frey and David J. Slate. "Letter Recognition Using Holland-Style Adaptive Classifiers". In: *Mach. Learn.* 6.2 (Mar. 1991), 161–182. URL: https://doi.org/10.1023/A:1022606404104 (cit. on p. 22).

[53] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on pp. 23, 32–34, 46, 83, 92, 102).

[54] Zhiqiang Teng, Shuai Teng, Jiqiao Zhang, Gongfa Chen, and Fangsen Cui. "Structural Damage Detection Based on Real-Time Vibration Signal and Convolutional Neural Network". In: *Applied Sciences* 10 (July 2020), p. 4720 (cit. on p. 23).

[55] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Madison, WI, USA: Omnipress, 2010, 807–814 (cit. on pp. 23, 91).

[56] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, 1097–1105 (cit. on p. 23).

[57] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2014. URL: https://arxiv.org/abs/1409.1556 (cit. on pp. 24–26, 29, 91).

[58] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. *Rethinking the Inception Architecture for Computer Vision*. 2015. URL: https://arxiv.org/abs/1512.00567 (cit. on pp. 25, 30).

[59] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. 2014. URL: https://arxiv.org/abs/1409.4842 (cit. on p. 25).

[60] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". In: (2018). URL: https://arxiv.org/abs/1801.04381 (cit. on pp. 25, 30).

[61] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV] (cit. on pp. 25, 30, 31, 52, 73, 80).

[62] NVIDIA. *GPU Performance Background User's Guide*. NVIDIA. 2020. URL: https://desh2608.github.io/2018-07-28-deep-learning-theory-3/ (cit. on p. 25).

[63] NVIDIA. *Memory-Limited Layers User's Guide*. NVIDIA. 2020. URL: https://desh2608.github.io/2018-07-28-deep-learning-theory-3/ (cit. on p. 25).

[64] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 26).

[65] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929 (cit. on pp. 26, 34, 109).

[66] Matthias Scholz, Martinand Fraunholz, and Joachim Selbig. "Nonlinear Principal Component Analysis: Neural Network Models and Applications". In: *Principal Manifolds for Data Visualization and Dimension Reduction*. Ed. by Balázs Gorban Alexander N.and Kégl, Donald C. Wunsch, and Andrei Y. Zinovyev. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 44–67 (cit. on p. 28).

[67] Geoffrey E. Hinton, David E. Rumelhart, and James L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362 (cit. on p. 28).

[68] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets". In: *Neural Comput.* 18.7 (July 2006), 1527–1554. URL: https://doi.org/10.1162/neco.2006.18.7.1527 (cit. on p. 28).

[69] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507. eprint: https://www.science.org/doi/pdf/10.1126/science.1127647. URL: https://www.science.org/doi/abs/10.1126/science.1127647 (cit. on p. 28).

[70] James A Hanley, Lawrence Joseph, Robert W Platt, Moo K Chung, and Patrick Belisle. "Visualizing the Median as the Minimum-Deviation Location". In: *The American Statistician* 55.2 (2001), pp. 150–152. eprint: https://doi.org/10.1198/000313001750358482. URL: https://doi.org/10.1198/000313001750358482 (cit. on p. 28).

[71] Gustav Grund Pihlgren, Fredrik Sandin, and Marcus Liwicki. *Improving Image Autoencoder Embeddings with Perceptual Loss*. 2020. URL: https://arxiv.org/abs/2001.03444 (cit. on pp. 28, 42).

[72] Jeremy Jordan. *Variational autoencoders*. https://www.jeremyjordan.me/variational-autoencoders/ (cit. on p. 28).

[73] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2013. URL: https://arxiv.org/abs/1312.6114 (cit. on pp. 29, 42).

[74] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. *Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning*. 2016. URL: https://arxiv.org/abs/1602.07261 (cit. on p. 30).

[75] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. URL: https://arxiv.org/abs/1704.04861 (cit. on p. 30).

[76] Mingxing Tan and Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: (2019). URL: https://arxiv.org/abs/1905.11946 (cit. on p. 30).

[77] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. *An Analysis of Deep Neural Network Models for Practical Applications*. 2016. URL: https://arxiv.org/abs/1605.07678 (cit. on p. 30).

[78] *Data Engineering, Preparation, and Labeling for AI 2019*. https://www.cognilytica.com/document/report-data-engineering-preparation-and-labeling-for-ai-2019/ (cit. on p. 31).

[79] *Data Scientists Spend Most of Their Time Cleaning Data*. https://whatsthebigdata.com/2016/05/01/data-scientists-spend-most-of-their-time-cleaning-data/ (cit. on p. 31).

[80] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: (2009), pp. 32–33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf (cit. on p. 32).

[81] Sergi Caelles, Jordi Pont-Tuset, Federico Perazzi, Alberto Montes, Kevis-Kokitsi Maninis, and Luc Van Gool. "The 2019 DAVIS Challenge on VOS: Unsupervised Multi-Object Segmentation". In: *arXiv:1905.00737* (2019) (cit. on p. 32).

[82] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on p. 32).

[83] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 (cit. on p. 32).

[84] Mengde Xu, Zheng Zhang, Han Hu, Jianfeng Wang, Lijuan Wang, Fangyun Wei, Xiang Bai, and Zicheng Liu. "End-to-End Semi-Supervised Object Detection with Soft Teacher". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2021) (cit. on p. 32).

[85] Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. "Knowledge Distillation: A Survey". In: *International Journal of Computer Vision* 129.6 (Mar. 2021), 1789–1819. URL: http://dx.doi.org/10.1007/s11263-021-01453-z (cit. on p. 32).

[86] *Welcome to the Internet*. https://i.kym-cdn.com/entries/icons/original/000/006/877/707538ef3afa883c1d146b42cf01bac2.jpg (cit. on p. 32).

[87] Ihab F Ilyas and Xu Chu. *Data cleaning*. Morgan & Claypool, 2019 (cit. on p. 33).

[88] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on p. 33).

[89] Junghoon Chae, Shang Gao, Arvind Ramanthan, Chad A. Steed, and Georgia D. Tourassi. "Visualization for Classification in Deep Neural Networks". In: 2017 (cit. on p. 34).

[90] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (cit. on pp. 34, 85).

[91] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. *COCO-Stuff: Thing and Stuff Classes in Context*. 2016. URL: https://arxiv.org/abs/1612.03716 (cit. on p. 34).

[92] Rich Caruana. "Learning Many Related Tasks at the Same Time with Backpropagation". In: *Advances in Neural Information Processing Systems*. Ed. by G. Tesauro, D. Touretzky, and T. Leen. Vol. 7. MIT Press, 1994 (cit. on p. 35).

[93] Yoshua Bengio, Frédéric Bastien, Arnaud Bergeron, Nicolas Boulanger–Lewandowski, Thomas Breuel, Youssouf Chherawala, Moustapha Cisse, Myriam Côté, Dumitru Erhan, Jeremy Eustache, Xavier Glorot, Xavier Muller, Sylvain Pannetier Lebeuf, Razvan Pascanu, Salah Rifai, François Savard, and Guillaume Sicard. "Deep Learners Benefit More from Out-of-Distribution Examples". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 164–172. URL: https://proceedings.mlr.press/v15/bengio11b.html (cit. on p. 35).

[94] Yoshua Bengio. "Deep Learning of Representations for Unsupervised and Transfer Learning". In: *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*. Ed. by Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver. Vol. 27. Proceedings of Machine Learning Research. Bellevue, Washington, USA: PMLR, July 2012, pp. 17–36. URL: https://proceedings.mlr.press/v27/bengio12a.html (cit. on p. 35).

[95] Qi Fan, Wei Zhuo, Chi-Keung Tang, and Yu-Wing Tai. "Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector". In: *CVPR*. 2020 (cit. on p. 36).

[96] Gongjie Zhang, Zhipeng Luo, Kaiwen Cui, and Shijian Lu. *Meta-DETR: Image-Level Few-Shot Object Detection with Inter-Class Correlation Exploitation*. 2021. URL: https://arxiv.org/abs/2103.11731 (cit. on pp. 36, 43).

[97] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. 2017. URL: https://arxiv.org/abs/1703.03400 (cit. on pp. 36, 37).

[98] Xiaopeng Yan, Ziliang Chen, Anni Xu, Xiaoxi Wang, Xiaodan Liang, and Liang Lin. "Meta r-cnn: Towards general solver for instance-level low-shot learning". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2019, pp. 9577–9586 (cit. on pp. 36, 37).

[99] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, koray kavukcuoglu, and Daan Wierstra. "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016 (cit. on pp. 36, 37).

[100] Kunpeng Li, Ziyan Wu, Kuan-Chuan Peng, Jan Ernst, and Yun Fu. "Tell Me Where to Look: Guided Attention Inference Network". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018 (cit. on p. 37).

[101] Zhongzheng Ren, Zhiding Yu, Xiaodong Yang, Ming-Yu Liu, Yong Jae Lee, Alexander G. Schwing, and Jan Kautz. "Instance-Aware, Context-Focused, and Memory-Efficient Weakly Supervised Object Detection". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. Computer Vision Foundation / IEEE, 2020, pp. 10595–10604. URL: https://openaccess.thecvf.com/content_CVPR_2020/html/Ren_Instance-Aware_Context-Focused_and_Memory-Efficient_Weakly_Supervised_Object_Detection_CVPR_2020_paper.html (cit. on pp. 37, 38, 43).

[102] Junsuk Choe, Seungho Lee, and Hyunjung Shim. "Attention-Based Dropout Layer for Weakly Supervised Single Object Localization and Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.12 (2021), pp. 4256–4271 (cit. on pp. 37, 38).

[103] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. *Learning Deep Features for Discriminative Localization*. 2015. URL: https://arxiv.org/abs/1512.04150 (cit. on pp. 37, 38).

[104] Yan Xu, Jun-Yan Zhu, I Eric, Chao Chang, Maode Lai, and Zhuowen Tu. "Weakly supervised histopathology cancer image segmentation and classification". In: *Medical image analysis* 18.3 (2014), pp. 591–604 (cit. on p. 37).

[105] Gang Xu, Zhigang Song, Zhuo Sun, Calvin Ku, Zhe Yang, Cancheng Liu, Shuhao Wang, Jianpeng Ma, and Wei Xu. "Camel: A weakly supervised learning framework for histopathology image segmentation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 10682–10691 (cit. on p. 37).

[106] R. Hadsell, S. Chopra, and Y. LeCun. "Dimensionality Reduction by Learning an Invariant Mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 1735–1742 (cit. on p. 39).

[107] Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. *Metric Learning with Adaptive Density Discrimination*. 2015. URL: https://arxiv.org/abs/1511.05939 (cit. on p. 39).

[108] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification." In: *Journal of machine learning research* 10.2 (2009) (cit. on p. 39).

[109] Andrew Zhai and Hao-Yu Wu. "Classification is a strong baseline for deep metric learning". In: *arXiv preprint arXiv:1811.12649* (2018) (cit. on p. 39).

[110] John Bridle. "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters". In: *Advances in Neural Information Processing Systems*. Ed. by D. Touretzky. Vol. 2. Morgan-Kaufmann, 1989. URL: https://proceedings.neurips.cc/paper/1989/file/0336dcbab05b9d5ad24f4333c7658a0e-Paper.pdf (cit. on p. 39).

[111] Shota Horiguchi, Daiki Ikami, and Kiyoharu Aizawa. "Significance of Softmax-Based Features in Comparison to Distance Metric Learning-Based Features". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.5 (2020), pp. 1279–1285 (cit. on p. 39).

[112] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. "A Simple Framework for Contrastive Learning of Visual Representations". In: *arXiv preprint arXiv:2002.05709* (2020) (cit. on pp. 40, 42).

[113] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. "Big Self-Supervised Models are Strong Semi-Supervised Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 22243–22255. URL: https://proceedings.

neurips.cc/paper/2020/file/fcbc95ccdd551da181207c0c1400c655-Paper.pdf (cit. on pp. 40, 42).

[114] Garrison W Cottrell. "Extracting features from faces using compression networks: Face, identity, emotion, and gender recognition using holons". In: *Connectionist Models*. Elsevier, 1991, pp. 328–337 (cit. on p. 40).

[115] Chen Xing, Li Ma, and Xiaoquan Yang. "Stacked denoise autoencoder based feature extraction and classification for hyperspectral images". In: *Journal of Sensors* 2016 (2016) (cit. on p. 40).

[116] Qinxue Meng, Daniel Catchpoole, David Skillicom, and Paul J Kennedy. "Relational autoencoder for feature extraction". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 364–371 (cit. on p. 40).

[117] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and Composing Robust Features with Denoising Autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. New York, NY, USA: Association for Computing Machinery, 2008, 1096–1103. URL: https://doi.org/10.1145/1390156.1390294 (cit. on p. 40).

[118] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, June 2014, pp. 1278–1286. URL: https://proceedings.mlr.press/v32/rezende14.html (cit. on pp. 40, 42).

[119] Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. "Semi-supervised Learning with Deep Generative Models". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/d523773c6b194f37b938d340d5d02232-Paper.pdf (cit. on pp. 40, 42).

[120] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. "Why Does Unsupervised Pre-training Help Deep Learning?" In: *Journal of Machine Learning Research* 11.19 (2010), pp. 625–660. URL: http://jmlr.org/papers/v11/erhan10a.html (cit. on p. 41).

[121] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. *Semi-Supervised Learning*. 2006 (cit. on p. 41).

[122] Carl Doersch. *Tutorial on Variational Autoencoders*. 2016. URL: https://arxiv.org/abs/1606.05908 (cit. on p. 42).

[123] Mehran Mehralian and Babak Karasfi. "RDCGAN: Unsupervised Representation Learning With Regularized Deep Convolutional Generative Adversarial Networks". In: *2018 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium*. 2018, pp. 31–38 (cit. on p. 42).

[124] Augustus Odena. *Semi-Supervised Learning with Generative Adversarial Networks*. 2016. URL: https://arxiv.org/abs/1606.01583 (cit. on p. 42).

[125] S. Fralick. "Learning to recognize patterns without a teacher". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 57–64 (cit. on p. 42).

[126] Olivier Chapelle, Alexander Zien, and Bernhard Schölkopf. *Semi-supervised learning*. MIT Press, 2006 (cit. on p. 42).

[127] Paola Cascante-Bonilla, Fuwen Tan, Yanjun Qi, and Vicente Ordonez. *Curriculum Labeling: Revisiting Pseudo-Labeling for Semi-Supervised Learning*. 2020. URL: https://arxiv.org/abs/2001.06001 (cit. on p. 42).

[128] Yang Zou, Zhiding Yu, B.V.K. Vijaya Kumar, and Jinsong Wang. "Unsupervised Domain Adaptation for Semantic Segmentation via Class-Balanced Self-Training". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018 (cit. on p. 42).

[129] Bowen Zhang, Yidong Wang, Wenxin Hou, Hao Wu, Jindong Wang, Manabu Okumura, and Takahiro Shinozaki. "FlexMatch: Boosting Semi-Supervised Learning with Curriculum Pseudo Labeling". In: *CoRR* abs/2110.08263 (2021). URL: https://arxiv.org/abs/2110.08263 (cit. on p. 42).

[130] Zihang Dai, Zhilin Yang, Fan Yang, William W. Cohen, and Ruslan Salakhutdinov. "Good Semi-Supervised Learning That Requires a Bad GAN". In: NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, 6513–6523 (cit. on p. 42).

[131] Fangyuan Zhang, Tianxiang Pan, and Bin Wang. "Semi-supervised Object Detection with Adaptive Class-Rebalancing Self-Training". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.3 (June 2022), pp. 3252–3261. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20234 (cit. on p. 43).

[132] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf (cit. on pp. 43, 102).

[133] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. "SSD: Single Shot MultiBox Detector". In: *Lecture Notes in Computer Science* (2016), 21–37 (cit. on pp. 46, 49, 52).

[134] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2015. URL: https://arxiv.org/abs/1506.01497 (cit. on p. 49).

[135] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. URL: http://www.poker-edge.com/stats.php (cit. on pp. 49, 51, 57).

[136] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6517–6525 (cit. on p. 49).

[137] Djamel Benarab, Thibault Napoléon, Ayman Alfalou, Antoine Verney, and Philippe Hellard. "Optimized swimmer tracking system by a dynamic fusion of correlation and color histogram techniques". In: *Optics Communications* 356 (Dec. 2015), pp. 256–268 (cit. on p. 50).

[138] Djamel Benarab, Thibault Napoléon, Ayman Alfalou, Antoine Verney, and Philippe Hellard. "A novel multitracking system for the evaluation of high-level swimmers performances". In: Baltimore, United States, May 2014 (cit. on p. 50).

[139] Timothy Woinoski and Ivan V. Bajić. *Swimmer Stroke Rate Estimation From Overhead Race Video*. 2021. arXiv: 2104.12056 [eess.IV] (cit. on pp. 51, 58, 59).

[140] Ashley Hall, Brandon Victor, Zhen He, Matthias Langer, Marc Elipot, Aiden Nibali, and Stuart Morgan. "The detection, tracking, and temporal action localisation of swimmers for automated analysis". In: *Neural Computing and Applications* 33 (June 2021), pp. 1–19 (cit. on p. 51).

[141] COCO. *COCO detection metric*. 2021. URL: https://cocodataset.org/#detection-eval (cit. on p. 56).

[142] Audrey Duran, Gaspard Dussert, Olivier Rouvière, Tristan Jaouen, Pierre-Marc Jodoin, and Carole Lartizien. "ProstAttention-Net: A deep attention model for prostate cancer segmentation by aggressiveness in MRI scans". In: *Medical Image Analysis* 77 (2022), p. 102347. URL: https://www.sciencedirect.com/science/article/pii/S1361841521003923 (cit. on p. 63).

[143] Rahul Anand Sharma, Bharath Bhat, Vineet Gandhi, and C. V. Jawahar. "Automated Top View Registration of Broadcast Football Videos". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018, pp. 305–313 (cit. on pp. 66, 70).

[144] Jianhui Chen and James J. Little. "Sports Camera Calibration via Synthetic Data". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2019, pp. 2497–2504 (cit. on pp. 66, 70, 78).

[145] Long Sha, Jennifer Hobbs, Panna Felsen, Xinyu Wei, Patrick Lucey, and Sujoy Ganguly. "End-to-End Camera Calibration for Broadcast Videos". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 13624–13633 (cit. on pp. 66, 67, 70, 72, 77, 78).

[146] Xiaohan Nie, Shixing Chen, and Raffay Hamid. "A Robust and Efficient Framework for Sports-Field Registration". In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2021, pp. 1935–1943 (cit. on pp. 66, 67, 70, 73, 78, 79).

[147] Wei Jiang, Juan Higuera, Baptiste Angles, Weiwei Sun, Mehrsan Javan Roshtkhari, and Kwang Yi. "Optimizing Through Learned Errors for Accurate Sports Field Registration". In: Mar. 2020, pp. 201–210 (cit. on pp. 66, 70, 78).

[148] Namdar Homayounfar, Sanja Fidler, and Raquel Urtasun. "Sports Field Localization via Deep Structured Models". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 4012–4020 (cit. on pp. 67, 70).

[149] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004 (cit. on p. 68).

[150] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), 381–395. URL: https://doi.org/10.1145/358669.358692 (cit. on pp. 68, 70, 75).

[151] Julius R. Blum. "Approximation Methods which Converge with Probability one". In: *The Annals of Mathematical Statistics* 25.2 (1954), pp. 382 –386. URL: https://doi.org/10.1214/aoms/1177728794 (cit. on p. 69).

[152] Elan Dubrofsky and Robert J. Woodham. "Combining Line and Point Correspondences for Homography Estimation". In: *Advances in Visual Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 202–213 (cit. on p. 69).

[153] Ankur Gupta, James J. Little, and Robert J. Woodham. "Using Line and Ellipse Features for Rectification of Broadcast Hockey Video". In: *2011 Canadian Conference on Computer and Robot Vision*. 2011, pp. 32–39 (cit. on p. 69).

[154] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. *Deep Image Homography Estimation*. 2016. URL: https://arxiv.org/abs/1606.03798 (cit. on p. 69).

[155] Hoang Le, Feng Liu, Shu Zhang, and Aseem Agarwala. "Deep Homography Estimation for Dynamic Scenes". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on p. 69).

[156] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "SuperGlue: Learning Feature Matching with Graph Neural Networks". In: *CVPR*. 2020. URL: https://arxiv.org/abs/1911.11763 (cit. on p. 69).

[157] Fei Wang, Lifeng Sun, Bo Yang, and Shiqiang Yang. "Fast Arc Detection Algorithm for Play Field Registration in Soccer Video Mining". In: *2006 IEEE International Conference on Systems, Man and Cybernetics*. Vol. 6. 2006, pp. 4932–4936 (cit. on p. 70).

[158] Hyunwoo Kim and Ki Sang Hong. "Soccer video mosaicing using self-calibration and line tracking". In: *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*. Vol. 1. 2000, 592–595 vol.1 (cit. on p. 70).

[159] Leonardo Citraro, Pablo Márquez-Neila, Stefano Savarè, Vivek Jayaram, Charles Dubout, Félix Renaut, Andrés Hasfura, Horesh Shitrit, and Pascal Fua. "Real-Time Camera Pose Estimation for Sports Fields". In: (Mar. 2020) (cit. on pp. 70, 78).

[160] Francesco Lacquaniti, Carlo Terzuolo, and Paolo Viviani. "The law relating the kinematic and figural aspects of drawing movements". In: *Acta Psychologica* 54.1 (1983), pp. 115–130. URL: https://www.sciencedirect.com/science/article/pii/0001691883900276 (cit. on p. 75).

[161] *1080 Ti vs RTX 2080 Ti vs Titan RTX Deep Learning Benchmarks with TensorFlow - 2018 2019 2020*. https://bizon-tech.com/blog/gtx1080ti-titan-rtx-2080-ti-deep-learning-benchmarks. Accessed: June 2022 (cit. on p. 77).

[162] *Deep Learning GPU Benchmarks 2021*. https://www.aime.info/en/blog/deep-learning-gpu-benchmarks-2021/. Accessed: June 2022 (cit. on p. 77).

[163] Geoffrey Hinton, Jeff Dean, and Oriol Vinyals. "Distilling the Knowledge in a Neural Network". In: Mar. 2014, pp. 1–9 (cit. on p. 77).

[164] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. *Deep Learning Scaling is Predictable, Empirically*. 2017. arXiv: 1712.00409 [cs.LG] (cit. on p. 83).

[165] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, Mustafa Suleyman, and Andrew Zisserman. *The Kinetics Human Action Video Dataset*. 2017. arXiv: 1705.06950 [cs.CV] (cit. on pp. 83, 85).

[166] Debidatta Dwibedi, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, and Andrew Zisserman. "Counting Out Time: Class Agnostic Video Repetition Counting in the Wild". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020 (cit. on pp. 83–85, 89, 93).

[167] Huaidong Zhang, Xuemiao Xu, Guoqiang Han, and Shengfeng He. *Context-aware and Scale-insensitive Temporal Repetition Counting*. 2020. arXiv: 2005.08465 [cs.CV] (cit. on pp. 83, 85).

[168] Yunhua Zhang, Ling Shao, and Cees G. M. Snoek. "Repetitive Activity Counting by Sight and Sound". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2021, pp. 14070–14079 (cit. on pp. 83–85, 92, 93).

[169] Jianqin Yin, Yanchun Wu, Chaoran Zhu, Zijin Yin, Huaping Liu, Yonghao Dang, Zhiyi Liu, and Jun Liu. "Energy-Based Periodicity Mining With Deep Features for Action Repetition Counting in Unconstrained Videos". In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), pp. 4812–4825 (cit. on pp. 83, 85, 93).

[170] Jan Ubbo van Baardewijk, Sarthak Agarwal, Alex S. Cornelissen, Marloes J. A. Joosen, Jiska Kentrop, Carolina Varon, and Anne-Marie Brouwer. "Early Detection of Exposure to Toxic Chemicals Using Continuously Recorded Multi-Sensor Physiology". In: *Sensors* 21.11 (2021). URL: https://www.mdpi.com/1424-8220/21/11/3616 (cit. on p. 83).

[171] Erik Vavrinsky, Jan Subjak, Martin Donoval, Alexandra Wagner, Tomas Zavodnik, and Helena Svobodova. "Application of Modern Multi-Sensor Holter in Diagnosis and Treatment". In: *Sensors* 20.9 (2020). URL: https://www.mdpi.com/1424-8220/20/9/2663 (cit. on p. 83).

[172] Gyorgy Kolumban-Antal, Vladko Lasak, Razvan Bogdan, and Bogdan Groza. "A Secure and Portable Multi-Sensor Module for Distributed Air Pollution Monitoring". In: *Sensors* 20.2 (2020). URL: https://www.mdpi.com/1424-8220/20/2/403 (cit. on p. 83).

[173] Geoffrey D. Hugo, Elisabeth Weiss, William C. Sleeman, Salim Balik, Paul J. Keall, Jun Lu, and Jeffrey F Williamson. "Data from 4D Lung Imaging of NSCLC Patients". In: (2016) (cit. on p. 83).

[174] Spyridon Bakas, Mauricio Reyes, András Jakab, Stefan Bauer, Markus Rempfler, Alessandro Crimi, Russell Shinohara, Christoph Berger, Sung Ha, Martin Rozycki, Marcel Prastawa, Esther Alberts, Jana Lipkova, John Freymann, Justin Kirby, Michel Bilello, Hassan Fathallah-Shaykh, Roland Wiest, Jan Kirschke, and Bjoern Menze. "Identifying the Best Machine Learning Algorithms for Brain Tumor Segmentation, Progression Assessment, and Overall Survival Prediction in the BRATS Challenge". In: (Mar. 2019), p. 38 (cit. on p. 83).

[175] Catalina Tobon-Gomez, Arjan J. Geers, Jochen Peters, Jürgen Weese, Karen Pinto, Rashed Karim, Mohammed Ammar, Abdelaziz Daoudi, Jan Margeta, Zulma Sandoval, Birgit Stender, Yefeng Zheng, Maria A. Zuluaga, Julian Betancur, Nicholas Ayache, Mohammed Amine Chikh, Jean-Louis Dillenseger, B. Michael Kelm, Saïd Mahmoudi, Sébastien Ourselin, Alexander Schlaefer, Tobias Schaeffter, Reza Razavi, and Kawal S. Rhode. "Benchmark for Algorithms Segmenting the Left Atrium From 3D CT and MRI Datasets". In: *IEEE Transactions on Medical Imaging* 34.7 (2015), pp. 1460–1473 (cit. on p. 83).

[176] Ofir Levy and Lior Wolf. "Live Repetition Counting". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. USA: IEEE Computer Society, 2015, 3020–3028 (cit. on pp. 84, 93).

[177] Ramprasad Polana and Randal Nelson. "Detection and Recognition of Periodic, Nonrigid Motion". In: *International Journal of Computer Vision* 23 (June 1997), pp. 261–282 (cit. on p. 84).

[178] Jing Yang, Hong Zhang, and Guohua Peng. "Time-domain period detection in short-duration videos". In: *Signal, Image and Video Processing* 10 (2016), pp. 695–702 (cit. on p. 84).

[179] T. F. H. Runia, C. G. M. Snoek, and A. W. M. Smeulders. "Real-World Repetition Estimation by Div, Grad and Curl". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 9009–9017 (cit. on pp. 85, 89, 90, 93).

[180] Bruno Ferreira, Pedro M. Ferreira, Gil Pinheiro, Nelson Figueiredo, Filipe Carvalho, Paulo Menezes, and Jorge Batista. "Deep learning approaches for workout repetition counting and validation". In: *Pattern Recognition Letters* 151 (2021), pp. 259–266 (cit. on p. 85).

[181] Dominik Scherer, Andreas Müller, and Sven Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition". In: *Artificial Neural Networks – ICANN 2010*. Ed. by Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101 (cit. on p. 91).

[182] Felix Scholkmann, Jens Boss, and Martin Wolf. "An Efficient Algorithm for Automatic Peak Detection in Noisy Periodic and Quasi-Periodic Signals". In: *Algorithms* 5 (Nov. 2012), pp. 588–603 (cit. on pp. 91, 92).

[183] Brady Zhou, Philipp Krähenbühl, and Vladlen Koltun. "Does computer vision matter for action?" In: *Science Robotics* 4.30 (May 2019), eaaw6661. URL: http://dx.doi.org/10.1126/scirobotics.aaw6661 (cit. on p. 91).

[184] E. Pogalin, A. W. M. Smeulders, and A. H. C. Thean. "Visual quasi-periodicity". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*. 2008, pp. 1–8 (cit. on p. 93).

[185] Susanne Schnell, Pegah Entezari, S. Chris Mahadewia Riti J.and Malaisrie, Patrick M. McCarthy, Jeremy D. Collins, James Carr, and Michael Markl. "Improved Semiautomated 4D Flow MRI Analysis in the Aorta in Patients With Congenital Aortic Valve Anomalies Versus Tricuspid Aortic Valves". In: *Journal of Computer Assisted Tomography* 40 (Jan. 2016) (cit. on p. 94).

[186] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, 226–231 (cit. on p. 104).

[187] Stéphane d'Ascoli, Hugo Touvron, Matthew Leavitt, Ari Morcos, Giulio Biroli, and Levent Sagun. *ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases*. 2021. URL: https://arxiv.org/abs/2103.10697 (cit. on p. 110).

[188] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. *Training data-efficient image transformers amp; distillation through attention*. 2020. URL: https://arxiv.org/abs/2012.12877 (cit. on p. 110).