

PROJET DE DÉPARTEMENT  
INGÉNIERIE MATHÉMATIQUE ET INFORMATIQUE (IMI)

---

## Tracking en Natation

---

Juliette COLLIN, Elise LACOSTE, Paul-Eloi MANGION, Ambroise ODONNAT,  
Manh-Dan PHAM, Charles RAUDE

*encadrés par*  
Rémi Carmigniani & Vincent Lepetit

<https://github.com/remingtonCarmi/TrackingSwimmingENPC>

## **Remerciements**

Nous souhaitons remercier en premier lieu M. Rémi CARMIGNANI, notre tuteur entreprise, pour son implication et ses nombreux conseils qui ont contribué à alimenter notre réflexion.

Nous souhaitons également remercier M. Vincent LEPETIT pour son expertise et son aide précieuse concernant l'aspect technique de notre travail, ainsi que pour sa disponibilité.

Nous tenons aussi à témoigner notre reconnaissance aux personnes suivantes:

M. Renaud JESTER pour l'échange très enrichissant que nous avons eu au sujet des homographies en natation. Sa connaissance du sujet nous a été très précieuse.

M. Théo VINCENT pour ses explications concernant le projet de département réalisé l'an dernier.

Enfin, nous aimerais remercier la FFN, la FFH et l'INSEP ainsi que l'ANR NePTUNE (ANR-19-STHP-0004) pour toutes les vidéos et données mises à notre disposition.

## Résumé

Ce projet s'intéresse au tracking automatique en natation, principalement sur trois axes différents, dans le but d'automatiser l'obtention fastidieuse de statistiques sur les nageurs, généralement obtenues par les entraîneurs.

Dans la première partie, nous présentons un algorithme qui détecte et suit la position de la tête du nageur dans une vidéo en utilisant des techniques de Deep Learning (plus particulièrement un réseau de neurones convolutionnel). Le modèle présenté ici résout le problème en combinant régression et classification.

Les données d'entrée de ce modèle doivent être modifiées à partir de la vidéo brute en calibrant et corrigent la perspective. Ceci motive le travail réalisé en deuxième partie sur des techniques de calibration automatique déjà utilisées dans le football ou le hockey.

Deux approches sont abordées pour résoudre ce problème : une méthode déterministe et une méthode utilisant un réseau de neurones convolutionnel U-net.

Enfin, nous avons travaillé sur la détection automatique des phases de nage, que ce soit la détection automatique de la nage en elle-même (crawl, papillon, brasse,...) ou la détection automatique des cycles de bras en crawl. La détection de la nage a été traitée par des techniques de Deep Learning, tandis que des outils de régression linéaire ont été utilisés pour détecter les cycles de bras.

Par ailleurs, dans chacune des trois parties, nous présentons également comment les différentes bases de données pour entraîner les modèles ont été créées.

**Mots-clés :** natation, Deep Learning, création de bases de données, réseaux de neurones convolutionnels, U-Net, homographie, tracking automatique, calibration automatique, détection de phases, régression, classification.

## Abstract

This project focuses on automatic tracking in swimming, mainly on three different aspects, in order to automate the time-consuming task currently done by coaches to evaluate swimmer's performances.

We first introduce an algorithm that tracks the position of a swimmer's head in a video using deep learning techniques, more specifically a convolutional neural network. The model presented here deals with the problem by combining a regression problem and a classification problem.

The data taken as input for this model has to be modified from raw footage (TV footage for instance) using calibration and perspective correction. This motivates the work conducted here in a second part on automatic calibration techniques that are already used in certain sports such as football or hockey.

We present two approaches to solve the problem: a deterministic method and a method that uses a U-net deep convolutional network.

Finally, we have worked on automatic detection of swimming phases, whether it be the automatic detection of swimming techniques (crawl, butterfly, breaststroke, ...) or the automatic detection of arm cycles in crawl. Both problems have been tackled using deep learning.

At the same time, in all three parts, we also present how the different training databases created to train each model have been generated.

**Keywords:** swimming, deep-learning, database creation, convolutional neural networks, U-Net, homography, automatic tracking, automatic calibration, phase detection, regression, classification.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>8</b>
1.1	Contexte d'étude . . . . .	8
1.2	Données mises à disposition . . . . .	8
1.3	Nos objectifs . . . . .	8
<b>2</b>	<b>Détection de la tête du nageur</b>	<b>10</b>
2.1	Base de données . . . . .	10
2.1.1	Calibration de la vidéo . . . . .	11
2.1.2	Synchronisation . . . . .	11
2.1.3	Rappel des conventions de la base de données . . . . .	12
2.1.4	Statistiques sur la base de données finale . . . . .	12
2.2	Présentation du modèle . . . . .	12
2.2.1	Idée générale . . . . .	12
2.2.2	Description détaillée du modèle . . . . .	13
2.2.3	Loss implémentée . . . . .	14
2.2.4	Métriques d'évaluation . . . . .	15
2.2.5	Résultats . . . . .	15
2.3	Pistes d'amélioration . . . . .	18
<b>3</b>	<b>Calibration automatique sur vidéo</b>	<b>19</b>
3.1	Objectifs . . . . .	19
3.2	État de l'art . . . . .	20
3.3	Aspect Mathématique . . . . .	20
3.4	Première approche déterministe . . . . .	22
3.5	Seconde approche par le Deep Learning . . . . .	23
3.5.1	Présentation . . . . .	23
3.5.2	Construction manuelle de la base de données . . . . .	25
3.5.3	Data augmentation . . . . .	26
3.5.4	Résultats . . . . .	27
3.6	Pistes d'améliorations . . . . .	29
<b>4</b>	<b>Détection des phases de nage</b>	<b>30</b>
4.1	Détection de la nage . . . . .	30
4.1.1	Préparation de la base de données . . . . .	30
4.1.2	Le réseau de neurones . . . . .	31
4.1.3	Résultats . . . . .	32
4.2	La détection des cycles de bras en crawl . . . . .	34
4.2.1	Synchronisation des données gyrométriques avec les images . . . . .	34
4.2.2	Préparation des données synchronisées . . . . .	35
4.2.3	Apprentissage . . . . .	37
4.3	Pistes d'amélioration . . . . .	39
<b>5</b>	<b>Conclusion</b>	<b>41</b>
<b>A</b>	<b>Annexe : Conventions sur les fichiers de la base de données</b>	<b>44</b>
<b>B</b>	<b>Annexe : Arborescence de <i>TrackingSwimmingENPC-master</i></b>	<b>46</b>
<b>C</b>	<b>Annexe : Utilisation des notebooks de détection des phases de nage</b>	<b>48</b>

## Liste des tableaux

1	Accuracy en fonction de la ligne d'eau . . . . .	33
2	Exemples de fichiers contenus dans le dossier <code>Gyro</code> . . . . .	49
3	Exemple de fichier <code>.csv</code> dans le dossier <code>labels</code> . . . . .	50

## Table des figures

1	Exemple d'image où il est difficile d'avoir une idée de la position de la tête du nageur. L'année dernière, ce type d'image n'était pas pointé. Grâce aux données à disposition cette année, ce type d'image peut être pointé. . . . .	10
2	Exemple des données obtenues grâce au fil tendu sur le nageur. . . . .	10
3	Images avant correction (gauche) et après correction (droite). Après correction de la perspective, l'apparence de la piscine ne dépend plus du point de vue de la caméra, ce qui facilite la détection du nageur. . . . .	11
4	Schéma des transformations successives des coordonnées de la tête du nageur . . . . .	11
5	Exemple d'image à partir de laquelle les sous-images sont générées . . . . .	13
6	Exemples de sous-images générées à partir d'une image entière de ligne d'eau. Ici trois sous-images de largeur 150 pixels sont générées. . . . .	13
7	Schéma représentant l'architecture du modèle implémenté pour des images en entrée de taille (108 × 30) . . . . .	14
8	Interprétation de la MAE dans le cas où la MAE vaut 20 pixels . . . . .	15
9	Courbes d'accuracy et de loss sur l'ancienne base de données. Les courbes sont en dent de scies, peut-être à cause d'une certaine inconsistance de la base de données. . . . .	16
10	Courbes d'accuracy et de loss sur la nouvelle base de données. Les courbes sont plus lisses et les résultats obtenus sont similaires à la figure précédente. . . . .	16
11	Courbes d'accuracy et de loss sur la base de données complète avec un <i>learning rate</i> initial de $2 \times 10^{-5}$ pour l' <i>optimizer Adam</i> . . . . .	17
12	Courbes des MAE respectivement sur l'ancienne base de données (à gauche), la nouvelle base (à droite) et la base mixte (en bas) . . . . .	17
13	Images avant correction (gauche) et après correction (droite) . . . . .	19
14	Logiciel de pointage manuel implémenté et utilisé dans le cadre du projet précédent . . . . .	19
15	Schéma d'autocalibration de caméra appliquée au football . . . . .	20
16	. . . . .	21
17	Normes internationales sur les dimensionnements des bassins Olympiques . . . . .	21
18	Détection de contours par méthode de Canny . . . . .	22
19	Repérage des lignes d'eau . . . . .	23
20	Repérage de ligne d'eau . . . . .	23
21	Ensemble des classes des marqueurs de couleurs sur un bassin Olympique . . . . .	24
22	Exemple de carte de probabilité pour la classe n°7 . . . . .	24
23	Architecture du réseau U-Net . . . . .	25
24	Interface graphique de pointage manuel . . . . .	26
25	Data augmentation . . . . .	27
26	Valeurs de loss obtenues avec 3 epochs pour 1145 données avec des batch de tailles 64 . . . . .	28
27	Courbes de loss pour 60 itérations sur l'ensemble d'entraînement (a) et 30 itérations sur l'ensemble de validation (b) . . . . .	28
28	Visualisation de l'image X1, du label 0 et de la prediction associée . . . . .	28
29	Saturation de la RAM après chargement de la base de données . . . . .	29
30	Vidéo backstroke_m_left - Ligne 1 - Frame n°2140 . . . . .	30
31	Image croppée de l1_f2140 de la vidéo backstroke_m_left . . . . .	31
32	Architecture de notre réseau de neurones . . . . .	32
33	loss en fonction du nombre d'itérations . . . . .	32
34	Echantillon d'images de la ligne d'eau 1 . . . . .	33
35	Echantillon d'images de la ligne d'eau 5 . . . . .	33
36	Echantillon d'images de la ligne d'eau 7 . . . . .	33
37	Visualisation des données gyrométriques lors de l'aller d'un nageur . . . . .	34
38	Les différentes étapes de la calibration d'une vidéo . . . . .	35
39	. . . . .	35
40	Synchronisation des données gyrométriques avec les images . . . . .	35
41	Explication de la labellisation souhaitée par une gaussienne . . . . .	36
42	Labels avec $g_1$ . . . . .	36
43	Labels avec $g_2$ . . . . .	37
44	Labels avec $g$ . . . . .	37

45	Repérage de cycles à partir des labels . . . . .	38
46	Comparaison entre labels attendus et prédicts sur une vidéo test . . . . .	39
47	Repérage de cycles à partir des labels . . . . .	39

# 1 Introduction

## 1.1 Contexte d'étude

L'analyse des performances des nageurs est indispensable pour que les nageurs soient compétitifs à très haut niveau. Les indicateurs pertinents sont par exemple la vitesse, la position du nageur, la fréquence des coups de bras, etc. Actuellement, ces statistiques sont relevées manuellement par les entraîneurs ce qui est lent et fastidieux. On cherche donc à automatiser ce processus. C'est un projet à long terme, qui a notamment été l'objet d'un projet de département en 2020. Le but final est d'obtenir automatiquement les statistiques de nageurs à partir d'une vidéo. Ce but peut être décomposé en plusieurs objectifs intermédiaires. Nous avons donc décidé de traiter trois axes dans le cadre de ce projet : la calibration automatique du bassin, la détection automatique de la position de la tête du nageur, la détection des cycles de bras ainsi que le type de nage nageée. Notre équipe étant constituée de six personnes, nous nous sommes répartis en trois équipes de deux.

Chacun de ces sujets demandant un travail conséquent et long, chaque équipe a travaillé séparément et indépendamment des résultats des autres. On pourrait envisager dans le futur de fusionner les algorithmes de ces trois parties pour automatiser chacune des étapes du processus : on prend une vidéo en entrée qui va se calibrer automatiquement pour ensuite être utilisée dans l'algorithme de détection automatique de la position de la tête et enfin dans la détection automatique de chaque cycle de bras. On obtiendrait ainsi la position, la vitesse instantanée ainsi que la durée de chaque cycle de bras du nageur.

## 1.2 Données mises à disposition

Dans le cadre de notre étude, la Fédération Française de Natation (FFN), l'Institut National du Sport, de l'Expertise et de la Performance (INSEP) ainsi que l'Ecole Nationale des Ponts et Chaussées (ENPC, grâce à Charlie Pretot et Rémi Carmignani dans le cadre de l'ARN NePTUNE) ont mis à notre disposition plusieurs vidéos d'épreuves de natation de différentes catégories (100m Femme, 100m Homme, 100m brasse, 100m crawl, etc) ainsi que des vidéos de nageurs dans le bassin de l'INSEP filmées avec des angles de vue différents. Nous nous concentrons sur les vidéos prises avec une caméra fixe. Dans l'objectif de détecter et suivre la tête des nageurs, nous uniformisons les différents points de vues en utilisant des techniques de vision par ordinateur (voir les sections suivantes) à l'aide des normes de dimensionnement des bassins olympiques (voir [1]).

En complément des vidéos des nageurs filmées à l'INSEP, nous avons aussi à notre disposition des données de "Fil Tendu" et des données gyrométriques. En effet, des nageurs de l'INSEP et de la Fédération Française d'Handisport (FFH) ont fait des longueurs avec des capteurs mesurant leur vitesse et leur position (grâce à un fil attaché au nageur) ainsi que la vitesse angulaire de leurs bras (grâce à un gyromètre). Ces données ont ainsi été utilisées pour la constitution de la base de données de la détection de la position de la tête (partie 2.) ainsi que pour celle de la détection des nages et des cycles de bras en crawl (partie 4.).

## 1.3 Nos objectifs

Dans le cadre de la **détection de la tête du nageur**, l'objectif principal était d'**étoffer la base de données** à notre disposition afin de pouvoir entraîner un **réseau de neurones convolutionnel** et obtenir des résultats plus convaincants que dans le cadre du projet de département de l'an dernier. En particulier, l'objectif est de **limiter le phénomène de sur-apprentissage** rendant le modèle entraîné peu généralisable.

Pour ce faire, nous avons repris la même architecture que le modèle de l'année dernière, qui considère le problème de détection de la tête du nageur dans une image de ligne d'eau comme deux sous-problèmes complémentaires. Le premier **problème de classification binaire** permet de dire si une tête est présente ou non dans une image. Le second problème est un **problème de régression** qui permet de prédire la position de la tête dans l'image. Afin de pouvoir évaluer les résultats obtenus avec la base de données étoffées, nous avons réalisé une étude comparative en entraînant le même modèle sur trois bases de données différentes : la base de données à disposition initialement, la base de données constituée dans le cadre de ce projet de département et finalement la base de données regroupant les deux bases de données précédentes.

L'objectif de la deuxième phase de notre travail, à savoir l'autocalibration de caméra, est **d'automatiser la phase de calibration des vidéos** qui permet de rendre indépendant de l'angle de vue de la caméra le tracking des nageurs. Cela nécessite de maintenir constantes les distances entre les lignes d'eau ainsi que les couleurs des marqueurs présents sur les images. Nos prédecesseurs se servaient d'un pointage manuel fastidieux et soumis à la précision de celui qui pointe.

Nous avons choisi de traiter ce problème par **une méthode déterministe de traitement d'images** fondée sur les caractéristiques visuelles des bassins homologués (voir [1]) et les méthodes de détection de contours et de lignes présentes dans la littérature.

Puis, par souci de généralisation - notre première méthode ne s'appliquant pas à un large panel d'images - nous avons décidé d'utiliser une approche par le Deep Learning. Nous avons choisi un **réseau de neurones convolutionnel de type UNet** (voir [2]) et construit une base de données adaptée à notre problème. Les features sont les images de piscine tandis que les labels sont des classes de marqueurs de couleur présents sur les lignes d'eau au nombre de 8 (voir 3.5.1).

Concernant la **détection de nage**, l'objectif est, pour une vidéo donnée, de pouvoir détecter la nage sur toute la vidéo, et plus particulièrement détecter les changements de nage. Cette détection de la nage n'est pas forcément utile de prime abord à l'acquisition de statistiques pour le nageur, mais pourrait servir dans le futur lors de divers processus d'automatisation. Nous avons donc entraîné un **classifieur à quatre classes**, chaque classe correspondant à une nage.

Pour le classifieur, nous aurions pu utiliser des méthodes de Machine Learning (comme un arbre de décision par exemple), mais nous avons privilégié l'utilisation d'un **réseau de neurones** pour plusieurs raisons. D'une part, nos données étant des images, un **réseau de neurones convolutionnel** semblait être la méthode la plus adaptée. D'autre part, l'équipe précédente a obtenu de bons résultats en utilisant un réseau de neurones convolutionnel, sur le même type de données que nous (des images de nageurs), alors qu'ils avaient eu des résultats moins convaincants avec des méthodes de Machine Learning. Ces résultats nous ont alors incités à choisir une méthode reposant sur le Deep Learning.

Enfin, pour la **détection des cycles de bras**, l'objectif est, sur une vidéo, de repérer les instants auxquels les bras sont en position levée (la durée des cycles de nages se déduisant alors de la différence de deux instants de positions de bras levé). Ce problème se ramène à un problème de régression plus qu'à un problème de classification. Nous nous sommes alors penchés sur **une méthode de régression linéaire**, qui nous semblait alors la méthode la plus rapide et la plus adaptée à notre problème.

## 2 Détection de la tête du nageur

### 2.1 Base de données

La première partie de notre travail concernant le suivi automatique des nageurs a été consacrée à l'enrichissement de la base de données construite lors du projet de département de l'année dernière. Contrairement à cette dernière, le pointage effectué cette année peut être qualifié de semi-automatique. En effet, la base de données brute fournie par M. Carmigniani comporte à la fois des vidéos et des données sur la position ainsi que la vitesse du nageur (ces dernières sont obtenues grâce à un fil attaché sur le nageur). Grâce à ces deux types de données, nous pouvons alors labeliser des images de manière quasi-automatique en réalisant au préalable deux étapes complémentaires :

1. la calibration de la vidéo (correction de la perspective) ;
2. la synchronisation entre la vidéo et les données de position et vitesse.

Une fois les phases de synchronisation et de calibration finies, la base de données peut être générée de manière automatique sans intervention humaine, d'où notre qualification de méthode semi-automatique. Il faut tout de même noter que même si l'utilisateur n'a plus besoin d'intervenir, il doit tout de même surveiller l'exécution du programme pour s'assurer qu'un décalage n'intervienne pas entre la vidéo et la position de la tête calculée à partir du fil tendu. Pour des raisons détaillées plus tard dans la partie calibration de la vidéo, il faut tout de même noter que la calibration manuelle de la vidéo doit être effectuée pour les deux techniques de génération de la base de données. Finalement, grâce à cette méthode, l'utilisateur n'est plus obligé de pointer la tête du nageur image par image dans une vidéo.

Ce pointage semi-automatique est aussi plus précis que le pointage manuel image par image dans le sens où il permet de labeliser des images où il est difficile pour l'oeil humain de détecter de manière claire la tête du nageur.



FIGURE 1 – Exemple d'image où il est difficile d'avoir une idée de la position de la tête du nageur. L'année dernière, ce type d'image n'était pas pointé. Grâce aux données à disposition cette année, ce type d'image peut être pointé.

Il faut cependant noter que cette méthode n'est pas nécessairement plus rapide en temps que pour le pointage manuel des images. En effet, la vitesse varie selon l'ordinateur sur lequel notre programme qui génère la base de données est exécuté. Il semble qu'un des facteurs limitant de cette méthode soit la vitesse de lecture et d'écriture des images sur le disque de l'ordinateur.

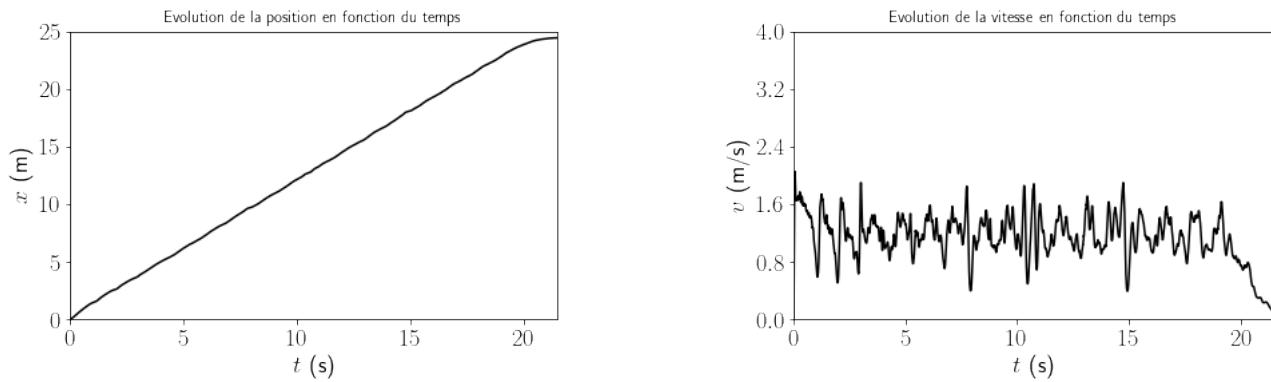


FIGURE 2 – Exemple des données obtenues grâce au fil tendu sur le nageur.

Les graphes ci-dessus montrent un exemple des données obtenues grâce au fil tendu sur le nageur qui viennent compléter les images vidéos. Sur le graphique de la vitesse instantanée du nageur, nous pouvons observer des grosses fluctuations de vitesse caractérisées par des pics abrupts. Ces fluctuations peuvent s'interpréter comme des coups de pieds donnés par le nageur sur le fil lors de sa course. Cependant, ces phénomènes sont très localisés et ne semblent pas grandement influencer le pointage semi-automatique que nous avons réalisé.

### 2.1.1 Calibration de la vidéo

La base de données générée est constituée d'images issues de vidéos de courses avec une vue "du dessus" (*top-down view* en anglais). Le principal avantage d'opter pour une telle perspective est que toutes les lignes d'eau ont une même largeur. En effet, ce n'est pas forcément le cas dans les images en perspective comme le montre la figure 3.

Dans l'image corrigée, les lignes d'eau sont parallèles entre elles selon la direction horizontale. Aussi, cette correction de perspective permet de standardiser les images sur lesquelles le réseau de neurones va s'entraîner et ainsi permettre de constituer une base de données construite grâce à des flux vidéos provenant de différentes caméras ainsi que de différents points de vues.

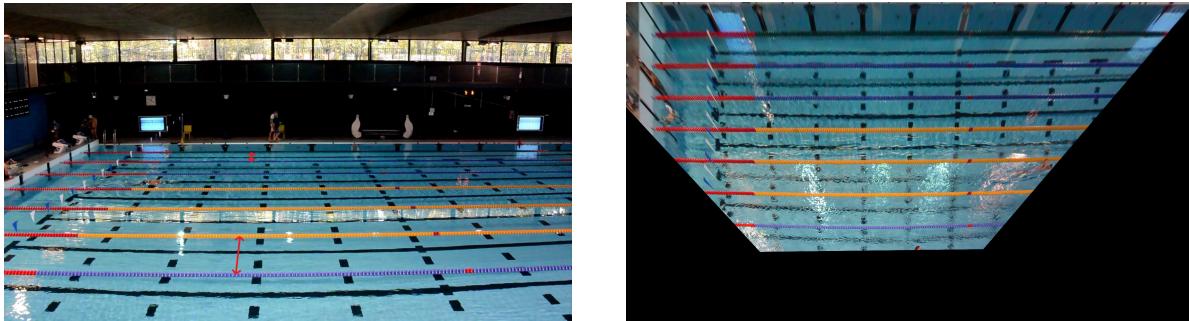


FIGURE 3 – Images avant correction (gauche) et après correction (droite). Après correction de la perspective, l'apparence de la piscine ne dépend plus du point de vue de la caméra, ce qui facilite la détection du nageur.

Pour corriger la perspective d'une image, nous devons sélectionner 4 points dans l'image dont on connaît les coordonnées réelles (en mètres). Les piscines respectant des normes internationales, il est facile de trouver de tels points, par exemple en prenant les marqueurs rouges sur les lignes d'eau. Ensuite, nous calculons la matrice d'homographie entre les coordonnées dans l'image (en pixels) de ces points et leurs coordonnées réelles. En multipliant l'image entière (vue comme un vecteur de taille *hauteur* × *largeur*) par cette matrice d'homographie, nous obtenons une image vue du dessus. Enfin, nous agrandissons cette image tout en respectant les proportions afin d'obtenir la plus grande image possible. L'image corrigée est finalement rognée horizontalement par ligne d'eau. Les sous-images ainsi construites représentent chaque ligne d'eau.

### 2.1.2 Synchronisation

Pour chaque image ainsi obtenue, la position de la tête du nageur est enregistrée dans un fichier .csv. Pour cela, nous calculons la position réelle du nageur à partir de la vitesse mesurée par le fil tendu. Nous transformons ensuite cette position en pixels dans la vidéo filmée par la caméra grâce à la matrice d'homographie calculée précédemment. En appliquant la matrice d'homographie reliant l'image brute à l'image corrigée, nous obtenons les coordonnées du nageur dans l'image finale.

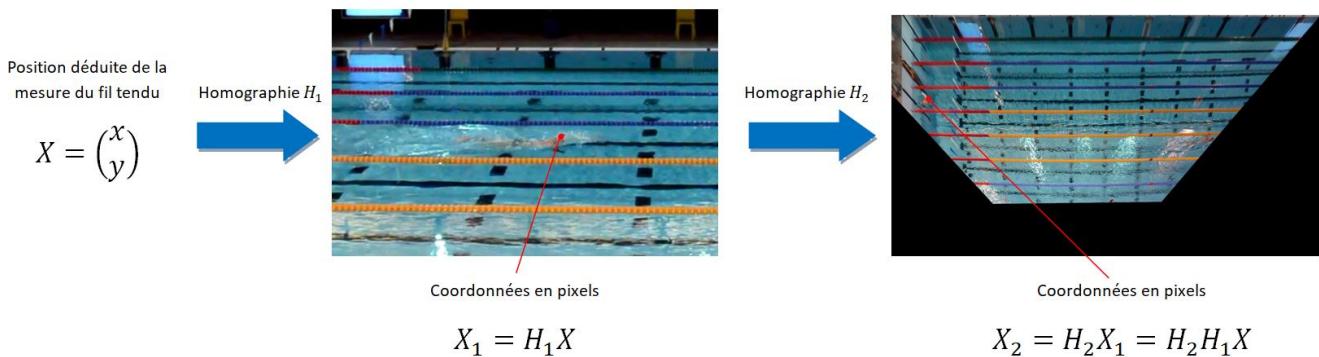


FIGURE 4 – Schéma des transformations successives des coordonnées de la tête du nageur

Une fois la perspective de la vidéo corrigée, il ne reste plus qu'à synchroniser les données mesurées par le fil

tendu et la vidéo du nageur : l'utilisateur doit rentrer à la main la frame de départ du nageur.

Afin de faciliter la détection de la frame de départ, nous avons codé un programme qui permet à l'utilisateur d'obtenir un intervalle avec les frames à tester. Le programme lit une vidéo et l'utilisateur double-clique quand le nageur part. Le programme lit une deuxième fois la vidéo de manière plus lente en reprenant juste avant l'instant du clic, ce qui permet à l'utilisateur de confirmer son choix. Cette deuxième lecture du fichier permet de corriger un éventuel décalage obtenu à cause du temps de réaction de l'utilisateur.

### 2.1.3 Rappel des conventions de la base de données

Nous avons repris les conventions et les standards fixés par le groupe de l'année dernière dans l'optique de pouvoir utiliser la base de données générée cette année grâce au pointage semi-automatique et la base de données à notre disposition. Pour chaque vidéo pointée, deux fichiers sont créés :

- un fichier `.txt` stockant la matrice d'homographie qui permet de corriger la perspective et d'avoir une *top down view*
- un fichier `.csv` qui stocke les informations pointées

Les conventions sont détaillées de manière plus précises dans l'annexe A.

### 2.1.4 Statistiques sur la base de données finale

La base de données initiale avant l'enrichissement effectué dans le cadre de ce projet de département était de 3153 images labelisées.

Notre enrichissement apporte 5868 images, portant le nombre total d'images à 9021 images, soit une augmentation de la taille de la base de données d'environ 186 %.

Par ailleurs, le modèle de Deep Learning que nous allons entraîner par la suite ne prend pas en argument des lignes d'eau entière mais plutôt des sous-images des lignes d'eau labelisées. Ainsi, à partir d'une base de données de 9021 images, nous pouvons agrandir de manière artificielle la taille de la base de données en échantillonant à partir d'une ligne d'eau des sous-images de taille fixe avec leurs labels associés.

## 2.2 Présentation du modèle

### 2.2.1 Idée générale

Pour détecter la tête du nageur dans une image lors d'une course, nous avons décidé d'implémenter un algorithme reposant sur le Deep Learning.

D'un point de vue théorique, la tâche que nous souhaitons modéliser est un problème de régression. En effet, dans l'idéal, à partir d'une image de couloir d'eau, l'algorithme doit savoir trouver un point dans l'image où se trouve la tête du nageur et renvoyer ses coordonnées.

Aussi, afin de simplifier la tâche de détection de la tête d'un nageur dans une image, nous avons fait l'hypothèse qu'un nageur nage droit, donc que la position de sa tête évolue uniquement dans la direction horizontale. En effet, nous estimons que durant une course, un nageur ne bouge pas trop sur l'axe vertical et qu'en première approximation, sa position peut se ramener à sa composante horizontale.

La régression étant une tâche complexe, nous avons décidé de nous ramener à une classe de sous-problèmes plus simples. En effet, plutôt que de chercher les coordonnées de la tête du nageur dans toute la ligne d'eau, nous avons divisé la ligne d'eau en sous-parties puis cherché à détecter la tête dans chacune de ces sous-images. La modélisation adoptée ici se ramène donc à deux sous-problèmes : un problème de classification binaire couplé à un problème de régression. Le problème de classification binaire permet de détecter si une tête est présente dans l'image donnée en entrée du modèle. Le problème de régression quant à lui renvoie la prédiction de la composante horizontale de la tête du nageur.

Enfin, afin de visualiser les prédictions de notre modèle, l'idée principale est d'utiliser deux réseaux de neurones indépendants qui traitent tous les deux le même problème. La seule différence entre ces deux modèles est la largeur des images prises en entrée par les deux réseaux. Le premier réseau est entraîné sur des images plus larges et permet d'avoir une première approximation de la position de la tête du nageur tandis que le deuxième réseau permet d'affiner le résultat obtenu avec le premier réseau. Cette idée permet notamment de ne pas avoir à faire tourner l'algorithme le plus fin sur toute la ligne d'eau subdivisée dans les dimensions des images prises en entrée par ce dernier, ce qui serait lourd en termes de calculs. En effet, la subdivision plus grande prise par le premier algorithme permet de donner un ordre d'idée de la zone où se trouve la tête dans l'image entière et permet d'avoir à faire tourner l'algorithme plus fin uniquement dans cette zone localisée.

### 2.2.2 Description détaillée du modèle

Le réseau implémenté ici est un réseau de neurones qui comporte 3 blocs de convolutions suivis d'un bloc final permettant de faire la prédiction voulue.

Dans notre cas, les entrées du modèles sont des images de dimensions `window_size × 108`. `window_size` est un hyperparamètre du modèle qui désigne la largeur des images d'entraînement de notre modèle et 108 correspond à la hauteur d'une ligne d'eau. Ces sous-images sont générées à partir de l'image de la ligne d'eau entière comme le montre la figure 5.



FIGURE 5 – Exemple d'image à partir de laquelle les sous-images sont générées

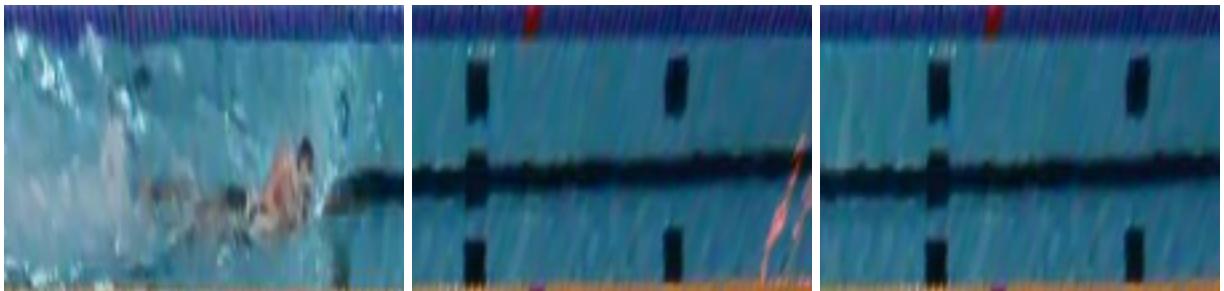


FIGURE 6 – Exemples de sous-images générées à partir d'une image entière de ligne d'eau. Ici trois sous-images de largeur 150 pixels sont générées.

La prédiction faite par le modèle est un triplet dont les deux premières composantes sont la prédiction de la classification binaire effectuée par le modèle tandis que la dernière composante est la prédiction de la coordonnée de la tête dans l'image.

Il faut tout de même noter que la prédiction est faite dans les coordonnées de la sous-image en question. Pour avoir la prédiction réelle sur toute la ligne d'eau, il faut reconstruire la ligne d'eau bout à bout.

Les trois blocs de convolutions successifs de notre modèle sont composés des trois couches suivantes :

- Conv2D ;
- BatchNormalization ;
- MaxPooling2D.

La non-linéarité ReLU est utilisée entre les couches convolutionnelles.

Les couches de BatchNormalization sont ajoutées à notre modèle à couches convolutionnelles afin de limiter au maximum le phénomène d'overfitting.

La couche finale est elle composée d'une couche Flatten suivies de deux couches linéaires séparées par une non-linéarité ReLU.

La figure 7 est une image représentant l'architecture du modèle implémenté dans le cadre du problème du pointage automatique de la tête en natation.

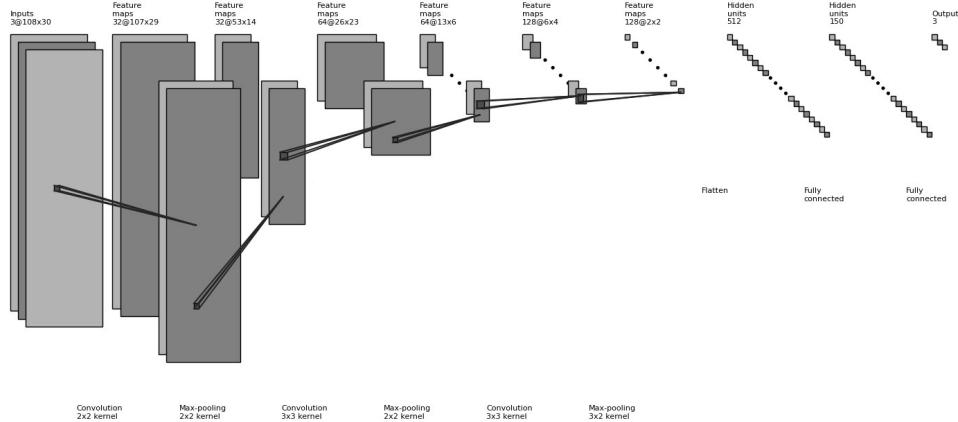


FIGURE 7 – Schéma représentant l'architecture du modèle implémenté pour des images en entrée de taille (108 × 30)

### 2.2.3 Loss implémentée

Afin que le réseau puisse apprendre et renvoyer une prédiction à la fois pour le problème de classification binaire et de régression, la loss sur laquelle nous avons entraîné notre modèle est une combinaison linéaire entre :

- l'entropie binaire croisée empirique pour le problème de classification binaire (*binary cross entropy*) ;
- l'erreur *l2* empirique (*mean square error*) pour le problème de régression.

Introduisons la fonction  $f$  qui représente le modèle décrit ci-dessus et notons  $(\mathbf{X}, \mathbf{Y})$  les images données en entrée au modèle munies de leurs labels. Pour une image  $i$ , la prédiction du modèle  $f(X_i)$  est donc un triplet où  $(f(X_i)_1, f(X_i)_2)$  est la prédiction liée au problème de classification binaire et  $f(X_i)_3$  la prédiction liée au problème de régression. Comme la classification effectuée ici est binaire, il vient dans les labels que  $Y_{i,1} = 1 - Y_{i,2}$ . La loss est alors :

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}) = \underbrace{-\frac{1}{n} \sum_{i=1}^n Y_{i,1} \log f(X_i)_1 + Y_{i,2} \log f(X_i)_2}_{\text{loss pour la classification}} + \alpha \underbrace{\frac{1}{n} \sum_{i=1}^n (f(X_i)_3 - Y_{i,3})^2}_{\text{loss pour la régression}}$$

Ici, le terme  $\alpha$  est un hyperparamètre de notre réseau de neurones appelé `trade_off` qui permet de donner plus d'importance au problème de classification ou au problème de régression.

Intuitivement, plus la valeur prise par  $\alpha$  est grande et plus l'accent sera mis sur le problème de régression et inversement. Cet hyperparamètre est particulièrement important pour entraîner les deux réseaux de neurones qui vont permettre de visualiser les résultats. En effet, pour le premier réseau entraîné, il est souhaitable que la composante la plus importante soit la précision des résultats de classification tandis qu'avec le deuxième réseau plus fin, il faut mettre l'accent sur le problème de régression.

#### 2.2.4 Métriques d'évaluation

Au-delà de la courbe de loss que nous cherchions à minimiser, nous nous sommes aussi intéressés à deux métriques d'évaluation lors des différentes phases d'entraînements effectuées ainsi que pendant les phases de cross-validation de certains hyperparamètres. La précision (*accuracy*) pour la classification binaire et l'erreur absolue moyenne (*mean absolute error* ou MAE) pour la partie régression.

En gardant les mêmes notations que dans la partie précédente sur la loss utilisée dans le modèle, les deux métriques sont définies de la manière suivante :

$$\left\{ \begin{array}{l} \text{Accuracy}(\mathbf{X}, \mathbf{Y}) = \frac{1}{n} \sum_{i=1}^n (\mathbb{1}_{f(X_i)_1=Y_{i,1}} + \mathbb{1}_{f(X_i)_2=Y_{i,2}}) \\ \text{MAE}(\mathbf{X}, \mathbf{Y}) = \frac{1}{n} \sum_{i=1}^n |f(X_i)_3 - Y_{i,3}| \end{array} \right.$$

L'erreur absolue moyenne permet d'évaluer la distance moyenne d'erreur commise par le réseau dans la phase de régression. Par exemple si la MAE est de 20 pixels, cela veut dire qu'en moyenne la prédiction se trouve sur le segment centré en la prédiction et de longueur totale de 40 pixels. Cette interprétation s'illustre sur la figure qui suit : la prédiction moyenne se trouve dans le rectangle noir.

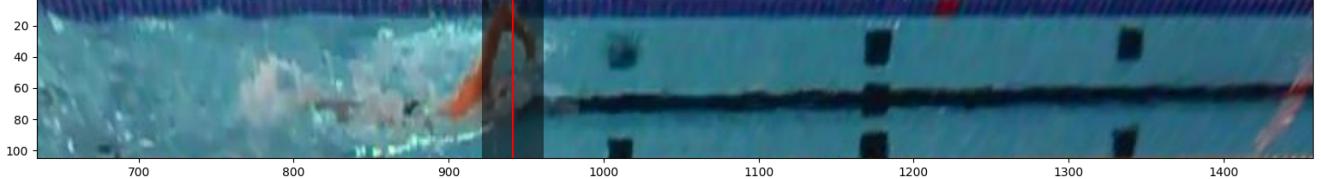


FIGURE 8 – Interprétation de la MAE dans le cas où la MAE vaut 20 pixels

#### 2.2.5 Résultats

Lors de la cross validation sur le paramètre `windows_size` régissant la largeur des images d'entraînement, nous avons réalisé que ce paramètre n'influe que très peu sur la précision du modèle pour la tâche de classification.

En revanche, plus la taille de la fenêtre est grande et plus la *mean absolute error* (MAE) est élevée. Cela s'explique par le fait qu'une grande fenêtre permet à l'algorithme de faire de plus grandes erreurs de prédiction (ie. une prédiction éloignée de la tête du nageur), ce qui influence directement la MAE. Nous avons écarté l'idée de normaliser la MAE par la largeur de l'image prise en entrée par notre modèle. En effet, bien que cela puisse permettre de comparer les performances de deux modèles avec des largeurs d'entrée différentes, comme l'objectif final est d'être le plus précis possible, il nous semblait plus pertinent de garder la MAE sous sa forme interprétable dans une image comme une distance moyenne.

Cette différence de comportement entre l'erreur de régression et l'erreur de classification justifie notre approche pour visualiser les prédictions de notre modèle avec deux réseaux de neurones indépendants. En effet, l'erreur de classification étant indépendante de la largeur de l'image considérée, nous pouvons nous permettre de découper grossièrement la ligne d'eau en plusieurs classes sans perte de précision pour la classification. Le premier réseau de neurones prédit alors dans quelle classe est la tête. Enfin, le deuxième réseau de neurones effectue la régression dans cette classe en la découplant en des images de taille plus petite afin de minimiser la *mean squared error* pour la régression dans la *loss*.

L'enrichissement de la base de données est effectué dans le but de limiter les phénomènes de sur-apprentissage d'un modèle de Deep Learning. Aussi, dans une optique comparative, nous avons fait tourner notre modèle sur la base générée cette année ainsi que sur la base totale obtenue. Les courbes ci-dessous mettent en lumière un phénomène auquel nous nous attendions pas : les courbes de *loss* et les courbes des métriques d'évaluation sont en dents de scies pour la base de données complète. Cela nous amène à penser qu'il y a éventuellement un problème dans la base de données initiale. Nous avons donc entraîné le réseau de neurones uniquement sur l'ancienne base de données, puis uniquement sur la nouvelle base de données afin de confirmer notre hypothèse. Nous pensons que ce phénomène

peut en partie s'expliquer par la trop forte similarité entre les images de validation et d'entraînement dans la base de données initiale. En effet, la base de données initiale est générée à partir d'images de courses internationales où les nageurs ont tous la même cadence. Sur les six vidéos de l'année dernière, cinq sont des compétitions de 100 mètres nage-libre. A contrario, notre base de données regroupe des images de plusieurs nageurs à plusieurs allures différentes.

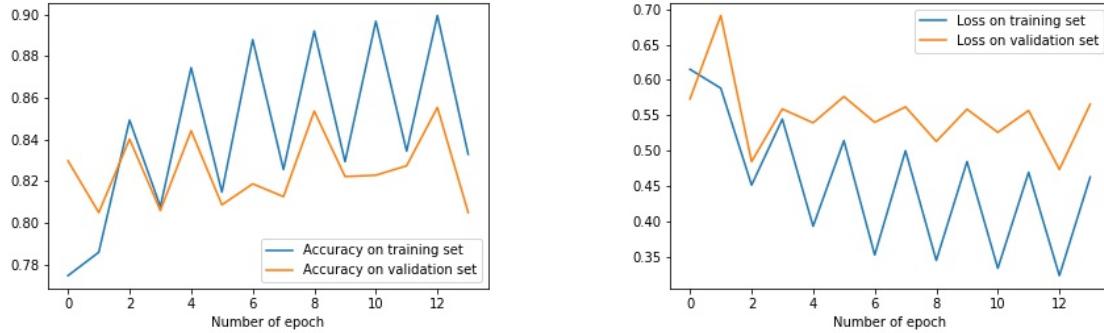


FIGURE 9 – Courbes d'accuracy et de loss sur l'ancienne base de données. Les courbes sont en dent de scies, peut-être à cause d'une certaine inconsistance de la base de données.

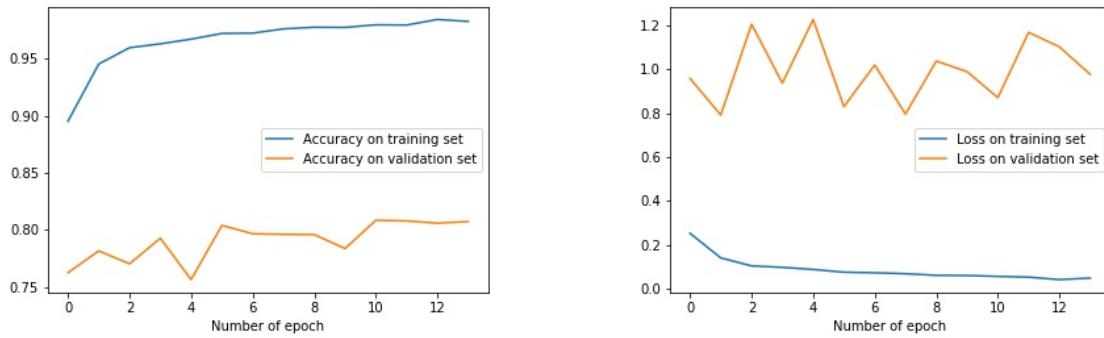


FIGURE 10 – Courbes d'accuracy et de loss sur la nouvelle base de données. Les courbes sont plus lisses et les résultats obtenus sont similaires à la figure précédente.

En effet, les courbes du modèle entraîné sur la nouvelle base de données sont beaucoup moins erratiques et présentent les mêmes performances (l'accuracy sur l'ensemble de validation est de l'ordre de 80%). Néanmoins, le phénomène de surapprentissage est plus marqué puisque l'écart entre les courbes d'entraînement et de validation est plus élevé (17% d'écart contre 5% avec l'ancienne base de données).

Dans l'optique d'utiliser les deux bases de données conjointement, nous avons essayé de résoudre les problèmes de fluctuation liés à la base de données initiale. Une première approche est de réduire le *learning rate*. Les résultats que nous avons obtenus avec cette approche semblent prometteurs. En effet, en réduisant le *learning rate* (initialement à  $1 \times 10^{-3}$ ), les courbes obtenues sont beaucoup moins erratiques et le sur-apprentissage semble limité. Il faut tout de même noter que l'entraînement doit être réalisé sur plus d'époques pour pouvoir obtenir des résultats similaires.

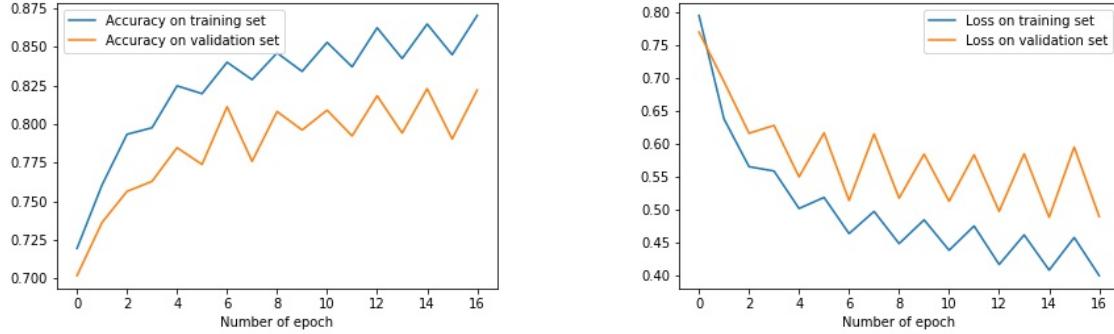


FIGURE 11 – Courbes d’accuracy et de loss sur la base de données complète avec un *learning rate* initial de  $2 \times 10^{-5}$  pour l’optimizer *Adam*

Concernant la métrique du MAE, les résultats obtenus sont satisfaisants. En effet, cette dernière est d’environ de 5 pixels après entraînement sur le modèle prenant en entrée des images de tailles 30 pixels de largeurs. Ce résultat est d’autant plus satisfaisant que la tête d’un nageur fait environ 20 pixels dans une image sur les vidéos constituant la base de données d’entraînement (voir figure 8).

Ainsi, le facteur limitant la marge de progression du modèle dans le problème de régression global semble être le problème de classification pour lequel les modèles peinent à obtenir une précision très satisfaisante au-delà de 90%.

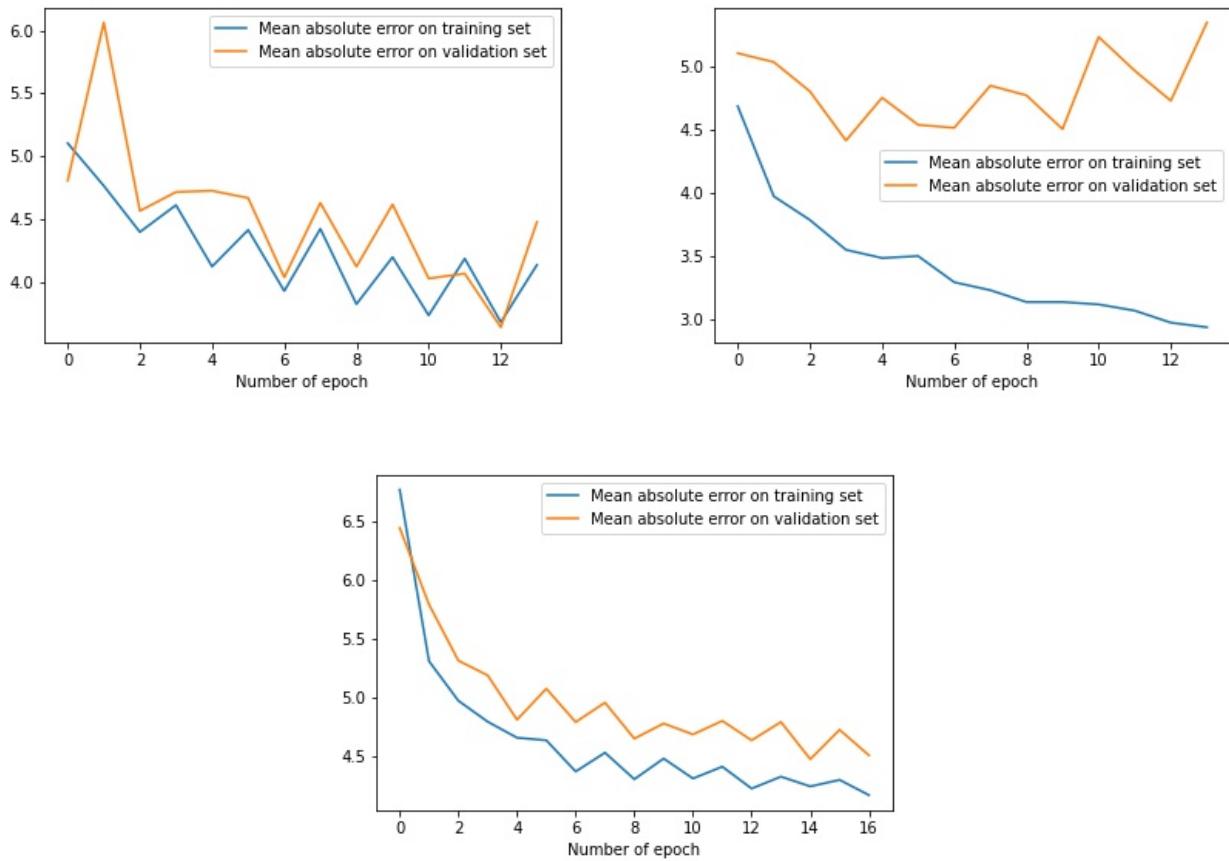


FIGURE 12 – Courbes des MAE respectivement sur l’ancienne base de données (à gauche), la nouvelle base (à droite) et la base mixte (en bas)

## 2.3 Pistes d'amélioration

Au regard des différentes courbes de loss obtenues à travers tous les entraînements, il apparaît que le phénomène de sur-apprentissage est très présent dans le problème de classification. En effet, notre modèle semble bien apprendre car l'accuracy sur la base d'entraînement frôle les 95% après quelques époques d'entraînements, ce qui n'est clairement pas le cas pour la base de validation.

Il faut noter que pour le problème de régression, ce même phénomène de sur-apprentissage, bien que présent ne semble pas être aussi important.

Notre modèle semble donc plus ou moins adapté pour répondre à la tâche souhaité mais il faut encore travailler pour chercher à le rendre plus généralisable en limitant l'overfitting.

Une idée serait par exemple d'utiliser les premières couches de réseaux pré-entraînés (par exemple les réseaux de neurones VGG ou ResNet) en y modifiant les dernières couches pour l'adapter au problème de détection de la tête en natation.

D'un tout autre point de vue, il peut être envisagé à la place de la classification binaire de séparer les couloirs d'eau en un nombre de classe fixe et de voir le problème de classification comme un problème multi-classes et non binaire. Plutôt que de prédire si la tête du nageur est présente ou non dans chacune des classes, le réseau va prédire directement dans quelle classe se trouve la tête. Dans ce cas, la loss décrite ci-dessous permet de combiner régression et classification.

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}) = \sum_j \sum_i \underbrace{|f(X_{ji})(i \times w + g(X_{ji})) - Y_j|}_{\in \{0,1\}}$$

Dans la loss ci-dessus, l'image  $X_j$  est divisée en plusieurs sous-images, et  $X_{ji}$  représente la  $i$ -ème sous-image de l'image  $X_j$ . Ainsi, si  $f(X_{ji}) = 0$ , cela signifie que la tête du nageur n'est pas dans la classe  $i$  et inversement si  $f(X_{ji}) = 1$ . Similairement,  $g(X_{ji})$  renvoie la coordonnées (en pixels dans la sous-image) de la tête dans l'image  $X_{ji}$ .

Une dernière idée serait de changer les images prises en entrées par le modèle. Plutôt que de considérer les images extraites à partir de la vidéo une par une, le modèle prendrait en entrée la différence de deux images consécutives. Cette approche permet notamment de mettre en valeur les éléments en mouvements sans réduire la taille de la base de données. Aussi, cette méthode ne serait pas sensible à la variation d'éclairage entre les différentes vidéos.

### 3 Calibration automatique sur vidéo

#### 3.1 Objectifs

La deuxième partie de notre travail a été consacrée à automatiser la phase de calibration des vidéos. Cette phase indispensable à l'exploitation des vidéos (voir section 2.1.1, page 11) a pour objectif d'obtenir une vue du dessus du bassin (*top-down view* Figure 13). Effectivement, après correction de la perspective, l'apparence de la piscine ne dépend plus du point de vue de la caméra, ce qui facilite la généralisation des méthodes permettant de détecter les nageurs.

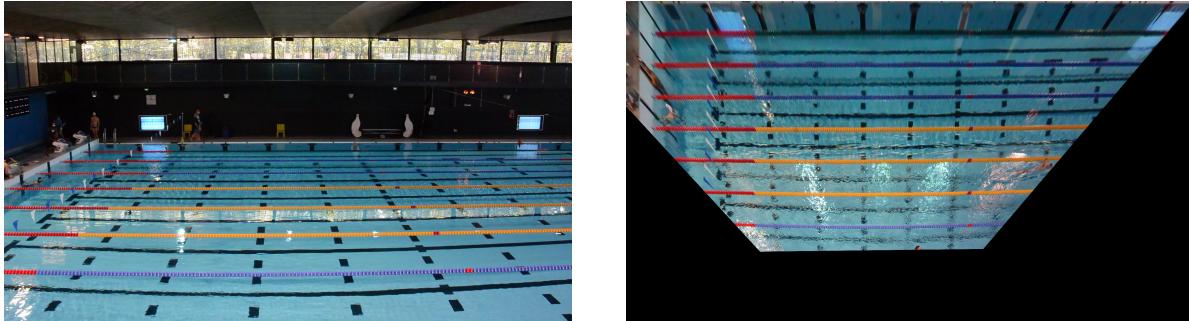


FIGURE 13 – Images avant correction (gauche) et après correction (droite)

Les difficultés sont nombreuses : les distances entre les lignes d'eau mais aussi entre les marqueurs de couleur doivent être conservées. Un étirement dans le sens de la largeur ou de la longueur aurait une influence non négligeable sur l'étude des images étant donné que les méthodes que l'on utilise pour repérer les nageurs ont des sensibilités de l'ordre du pixel.

Jusqu'à maintenant cette correction de la perspective était effectuée manuellement pour chaque vidéo et consistait à pointer 4 pixels correspondant à des marqueurs de ligne d'eau et à indiquer leurs coordonnées réelles dans le plan du bassin [3]. Nos prédecesseurs ont créé une interface graphique (voir Figure 14) permettant d'effectuer ce pointage. Cela permet de déterminer l'homographie à appliquer pour transformer l'angle de vue original en une *top-down view* (voir section 3.3 pour plus de précisions mathématiques).

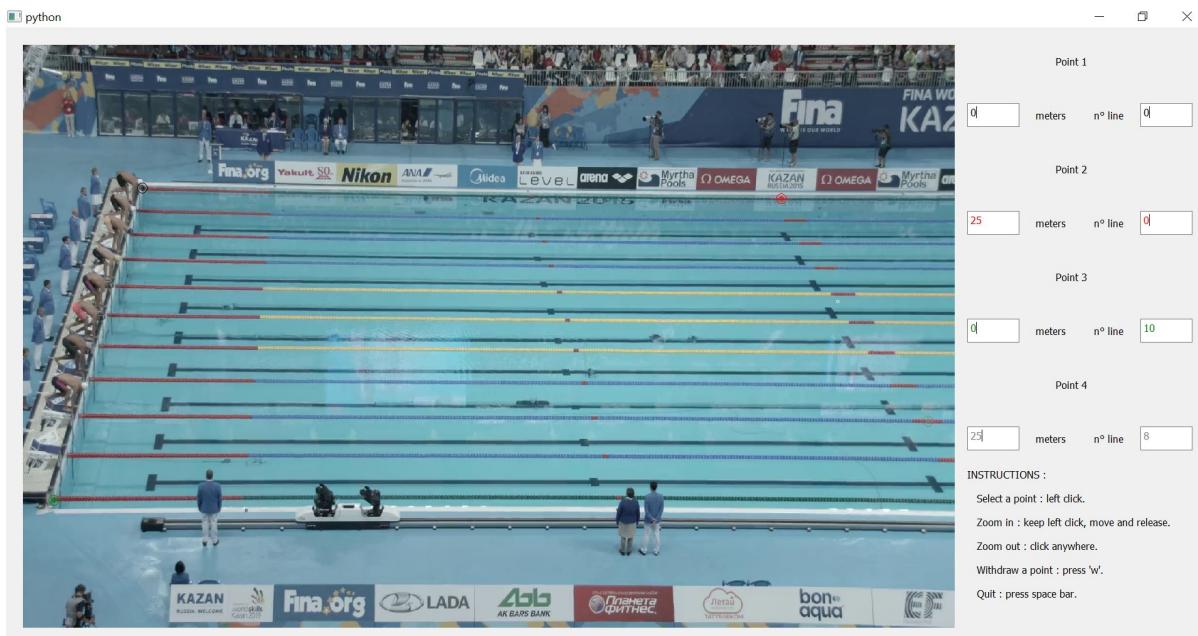


FIGURE 14 – Logiciel de pointage manuel implémenté et utilisé dans le cadre du projet précédent

La calibration manuelle, nécessitant la sélection de 4 points dans l'image d'origine et leurs coordonnées correspondantes dans l'image corrigée, en plus d'être longue, ne permet de travailler que sur des vidéos en plan fixe. En effet, pour une vidéo en travelling, la perspective change à chaque image ce qui rend la correction infaisable à la main.

Notre objectif est donc d'implémenter une méthode qui prendrait en argument une image de bassin de natation vu d'un angle quelconque et corrigerait la perspective automatiquement.

### 3.2 État de l'art

L'autocalibration de caméra consiste à déterminer la relation mathématique liant des points en trois dimensions, dans le plan de l'objet photographié, et des points en deux dimensions, dans le plan-image (voir section 3.3). Les articles de Jianhui Chen et James J. Little [4] et de Ankur Gupta, James J. Little et Robert J. Woodha [5] présentent des méthodes récentes (2018) permettant de réaliser cette autocalibration.

L'idée principale présente dans ces deux articles est l'utilisation des marqueurs de couleurs qui caractérisent les différents terrains sportifs étudiés (piscine olympique, terrain de football, terrain de hockey sur glace,...). Ils constituent en effet des points particuliers qu'on peut chercher à repérer. Les réseaux de neurones (dont les GAN pour Generative Adversarial Network en sont un exemple) sont souvent utilisés pour ce type de repérage.

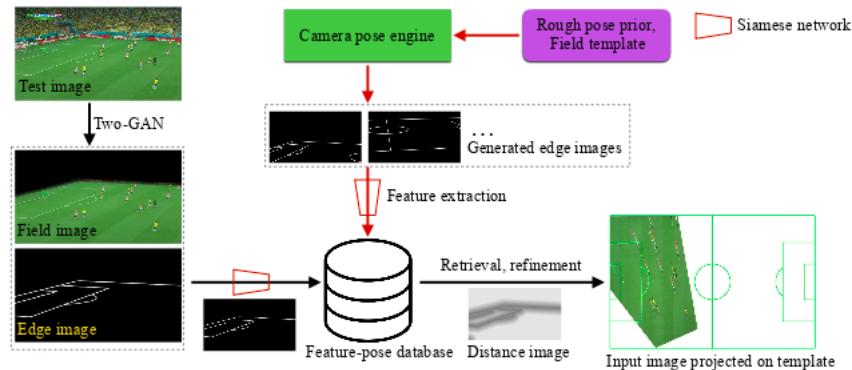


FIGURE 15 – Schéma d'autocalibration de caméra appliquée au football

Comme on peut le voir sur le schéma ci-dessus, en phase d'entraînement on récupère les contours du terrain, à savoir les lignes blanches soumises aux normes nationales et internationales (voir [6]). On construit ainsi une base de données de contours, que l'on cherche en phase de test à repérer sur une image de terrain à l'aide de deux GAN.

Nous exploitons également des points particuliers (marqueurs de couleur indiquant la distance parcourue le long des lignes d'eau) dans la conception de notre modèle d'autocalibration. Par ailleurs, le chercheur Renaud Jester [7] a implémenté un code permettant de repérer les lignes d'eau sur une image de piscine. Il se sert des normes internationales auxquelles sont soumises les piscines olympiques (voir [1]), y compris pour la couleur des lignes d'eau.

### 3.3 Aspect Mathématique

L'opération de correction de perspective est en fait une transformation 3D du plan de l'eau. Cette transformation, appelée homographie, peut être représentée par une matrice  $3 \times 3 H$  définie à un coefficient d'homogénéisation près. Sous cette représentation, la matrice d'homographie  $H$  vérifie en tout pixel de l'image l'équation suivante :

$$\begin{bmatrix} X_{top-down-view} \\ Y_{top-down-view} \\ 1 \end{bmatrix} = H \begin{bmatrix} X_{image} \\ Y_{image} \\ 1 \end{bmatrix}$$

Avec :

- $(X_{image}, Y_{image})$  les coordonnées du pixel dans l'image initial (point M de la Figure 16)

-  $(X_{top-down-view}, Y_{top-down-view})$  les coordonnées du pixel dans l'image en vue du dessus après correction (point m de la Figure 16).

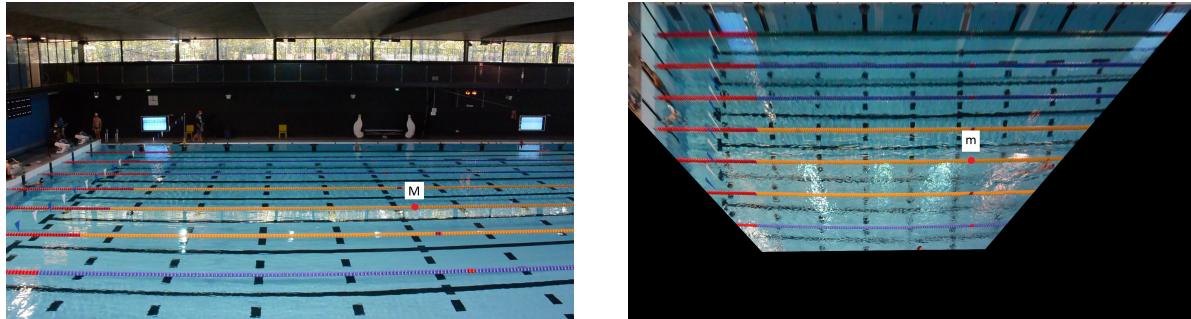


FIGURE 16

Il suffit donc de déterminer la matrice d'homographie  $H$  pour obtenir l'image en vue du dessus. Seuls 8 coefficients de la matrice  $3 \times 3$  sont en fait à déterminer car elle est définie à un coefficient d'homogénéisation près. Il suffit donc de connaître les coordonnées de 4 points dans les deux images pour résoudre l'équation qui détermine les 8 coefficients de la matrice d'homographie.

La première idée instinctive a donc été de repérer les 4 coins de la piscine mais nous nous sommes très vite heurtés à un problème pratique : les angles de prises de vidéo ne montraient en général qu'un ou deux coins. Il fallait donc s'intéresser à d'autres points à l'intérieur du bassin. Les autres points dont on connaît les coordonnées dans l'image corrigée sont ceux correspondant aux marqueurs sur les lignes d'eau que l'on peut observer sur la Figure 17.

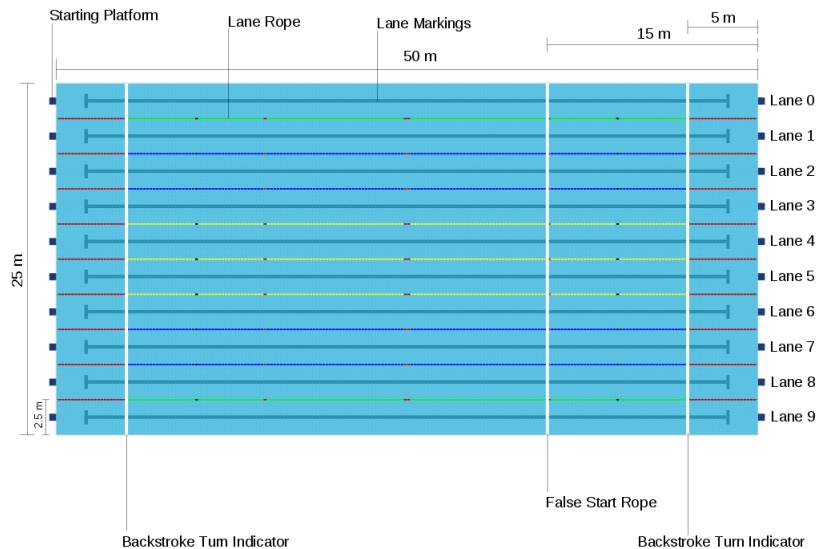


FIGURE 17 – Normes internationales sur les dimensionnements des bassins Olympiques

On s'efforcera donc par la suite de détecter automatiquement ces marqueurs pour ainsi estimer la matrice  $H$  et obtenir l'image en vue du dessus. Pour ce faire, nous avons réfléchi à deux stratégies : une méthode de traitement d'image traditionnelle déterministe et une approche par Deep Learning.

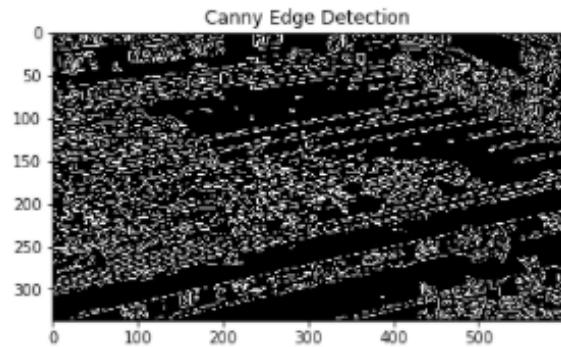
### 3.4 Première approche déterministe

La méthode traditionnelle consiste à repérer dans une image de bassin les lignes d'eau. Plus précisément, il s'agit d'exhiber quatre lignes dont trois non parallèles (il n'est pas nécessaire qu'elles soient visibles entièrement) afin de déterminer la transformation à appliquer pour récupérer l'image corrigée avec le bon angle de vue. Une fois ces droites repérées, la connaissance des angles qu'elles forment entre elles permet de construire l'homographie voulue. Cependant, sous les conseils de nos professeurs, nous n'avons pas poursuivi dans cette voie. En effet, si cette méthode est efficace pour certaines images, elle manque de robustesse et peut s'avérer inadéquate dans certains cas. C'est la raison pour laquelle nous développons une méthode utilisant le Deep Learning (voir section 3.5.1).

**Etat de l'art** En l'état actuel des choses, un certain nombre de méthodes permettant de repérer les contours d'un objet existent et peuvent être utilisées dans le but de repérer les lignes d'eau d'un bassin. Parmi les plus célèbres figurent la transformée de Hough et le filtre de Canny. La première est notamment utilisé dans la détection de lignes présentes dans une image (voir [8]). La seconde sert à la détection de contours (méthode abordée durant le cours de Traitement de l'Information et Vision Artificielle de Monsieur Vincent Lepetit). On peut observer ci-dessous les résultats obtenus avec le filtre de Canny sur une photographie d'un bassin Olympique avec un angle particulier.



(a) Photographie d'un bassin de natation Olympique



(b) Obtention des contours à l'aide du filtre de Canny

FIGURE 18 – Détection de contours par méthode de Canny

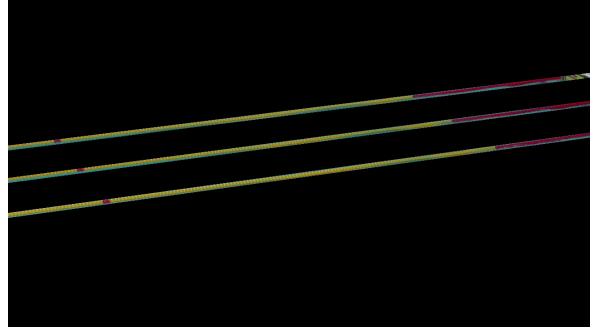
On remarque que le rendu est peu satisfaisant et sous-entend qu'un travail supplémentaire est nécessaire pour détecter au mieux les lignes d'eau (et non pas simplement les contours ou les droites présentent dans l'image).

**Principe** Nous nous sommes inspirés du travail réalisé par Renaud Jester (voir [7]) pour détecter les lignes d'eau. On se sert ici du fait que les bassins réglementaires comportent nombreux de marqueurs de couleur et sont normalisés. Ainsi on peut garder, par exemple, uniquement les lignes jaunes et rouges en appliquant un filtre qui ne conserve que les teintes rouges et jaunes d'une image. Ensuite, on réalise un cluster afin d'exhiber les lignes significatives de la transformée de Hough de l'image originale. Les trois lignes (jaunes) au centre de la piscine sont identifiables. Renaud Jester repère des points particuliers (repères des 5m, 15m, 25m, 35m), là où nous souhaitions repérer les angles formés par les lignes, afin de déterminer l'homographie.

**Résultats** Une des principales limites à cette méthode est le fait que les droites repérées sortent parfois du cadre du bassin comme on peut le voir ci-dessous. Aussi on ne repère plus seulement les lignes d'eau mais leur prolongement également comme on peut le voir ci-dessous sur les rendus obtenus( [7]).



(a) Photographie d'un bassin de natation Olympique

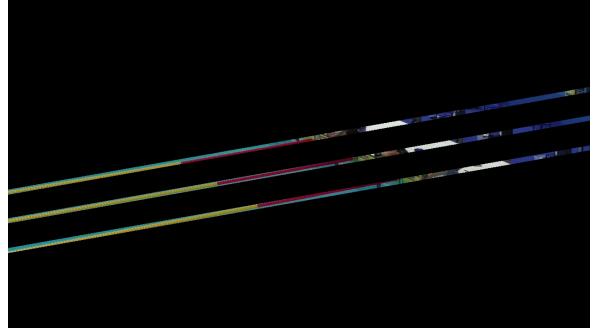


(b) Obtention des lignes d'eau

FIGURE 19 – Repérage des lignes d'eau



(a) Photographie d'un bassin de natation Olympique



(b) Obtention des lignes d'eau

FIGURE 20 – Repérage de ligne d'eau

### 3.5 Seconde approche par le Deep Learning

#### 3.5.1 Présentation

Afin de construire une méthode plus générale qui s'adapte mieux aux différentes images ou vidéos qu'on lui soumet, nous nous sommes tournés vers le Deep Learning. À titre de rappel, pour obtenir l'homographie, il suffit de détecter 4 marqueurs de couleur et de connaître leurs coordonnées réelles dans le bassin (cf section 3.3), ce sera donc l'objectif de cette méthode.

Parmi les travaux disponibles sur ce sujet, nous n'avons à ce jour trouvé aucun réseau entrainé pour ce type de tâche sur des images de bassin de natation et surtout aucune base de données d'images de bassin labellisées avec les marqueurs des lignes. Nous sommes donc parti de zéro, en définissant les classes, le format des labels, l'architecture du réseau, le type d'entraînement et en créant la base de données.

Définissons dans un premier temps les classes. Le cas idéal serait une classe par marqueur, ce qui permettrait de les identifier de manière unique et ainsi d'obtenir directement leurs coordonnées dans le bassin. Malheureusement ce cas idéal est irréalisable en pratique car beaucoup de marqueurs se ressemblent (bouées rouges sur ligne jaune, bouées rouges sur ligne bleu, etc), les marqueurs sont donc finalement regroupé par ressemblance afin de créer des classes différentiables par le réseau. On considère 8 classes, où chaque classe est un ensemble de marqueurs de couleur présents sur les lignes d'eau, définie comme ci-dessous (Figure 21).

L'utilisation d'un réseau de neurones nécessite l'uniformisation de la résolution des images. Nous avons donc essayé de trouver un bon compromis entre place mémoire et précision en choisissant de redimensionner les images au format  $512 \times 512$ . Pour les labels, nous avons opté pour 8 cartes de probabilité (une par classe) de la même taille que l'image en entrée, attribuant à chaque pixel d'une carte la probabilité d'appartenir à la classe correspondante à la dite carte. La Figure 22 montre par exemple pour une image de bassin la carte de probabilité correspondant à la classe n°7.

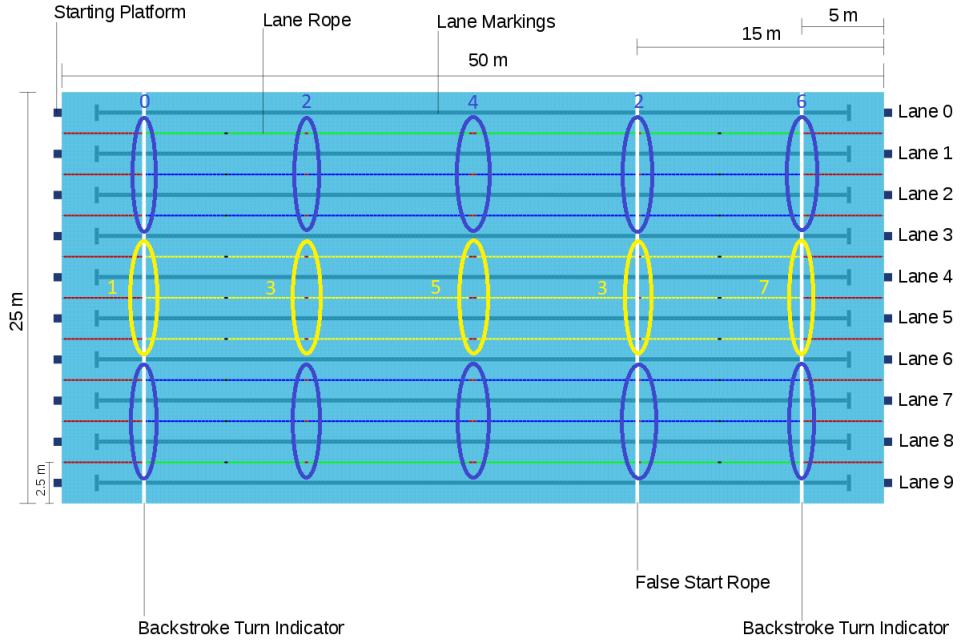


FIGURE 21 – Ensemble des classes des marqueurs de couleurs sur un bassin Olympique

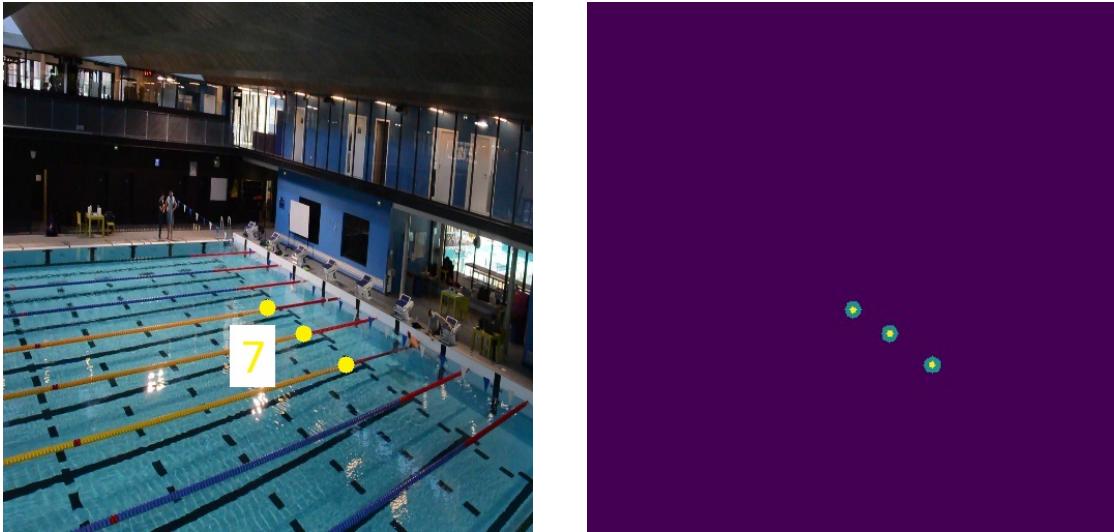


FIGURE 22 – Exemple de carte de probabilité pour la classe n°7

Le réseau va donc prendre en entrée une image de dimension  $3 \times 512 \times 512$ , et renvoyer une carte de probabilité de dimension  $8 \times 512 \times 512$  où pour la classe  $i \in [1, 8]$  on obtient pour chaque pixel la probabilité qu'il appartienne à la classe  $i$ .

Vient alors le choix du réseau de neurones. De nombreux algorithmes de classification existent (K-nearest neighbours, Random Forest, Support Vector Machine,...). La difficulté résulte dans le fait que, là où ces algorithmes classiques renvoient en sortie un vecteur de taille (input size, number of classes), nous souhaitons renvoyer un vecteur de taille (number of classes, size of an image). Nous avons donc choisi d'utiliser l'architecture U-Net, adaptée à ce genre de classification (et à la segmentation d'images de manière plus générale).

**Principe du réseau** Le réseau U-Net se construit comme suit ([2]) : il est formé d'un ensemble de convolutions (associées à des Max-Pooling) le tout formant une phase contractante (qui descend) et une phase expansive d'augmentation des tailles des sorties. Il tire sa forme de "U" de ces deux phases qui s'opposent symétriquement. Plus particulièrement, il est formé de quatre couches convolutionnelles qui permettent d'une part de diminuer l'importance des données spatiales pour mettre en valeur les caractéristiques en phase contractante et d'autre part d'insister sur les données spatiales en phase expansive. La Figure 23, présente l'architecture type d'un réseau U-Net à laquelle nous avons ajouté quelques modifications simples (dimensions d'entrée, nombre de classe et zero padding).

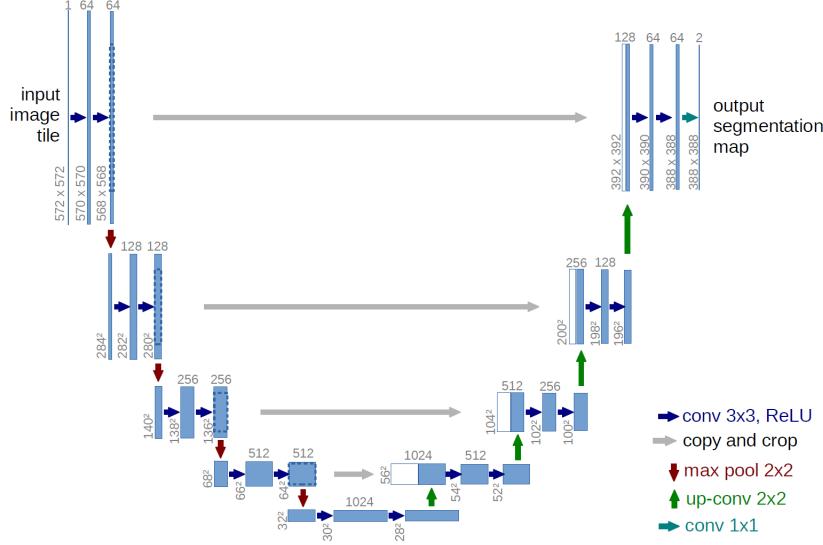


FIGURE 23 – Architecture du réseau U-Net

Maintenant que l'on a défini les classes, le format des labels et l'architecture du réseau, il ne nous reste plus qu'à créer la base de données.

### 3.5.2 Construction manuelle de la base de données

Ne disposant pas de base de données d'image de bassin de natation annoté (avec les positions des marqueurs), nous avons dû la construire entièrement, à partir des vidéos fournies par la Fédération Française de Natation, en les annotant à la main.

Pour économiser un temps précieux, nous avons implémenté un logiciel de pointage en modifiant celui implanté dans le cadre du projet de l'année précédente (voir Figure 24). Il permet d'obtenir simplement les coordonnées des pixels pointés et leur classe respective en l'indiquant dans les cases à droite en respectant l'ordre de pointage.

Devant l'aspect fastidieux et chronophage de cette tâche nous nous sommes limités à 229 images que l'on a tirées des vidéos de la base de données fournie par la FFN. Elles regroupent plusieurs angles de prises de vue sur des bassins Olympiques, différentes luminosité, ainsi que quelques nageurs à divers instants d'une course.

Grace à l'interface graphique de pointage, on obtient les coordonnées des pixels correspondant à chacune des 8 classes et on peut ainsi construire les labels sous forme de 8 cartes de probabilité.

Ces 229 paires images/labels ne sont bien évidemment pas suffisantes pour entraîner un modèle, c'est pourquoi nous avons effectué de la "*Data Augmentation*".

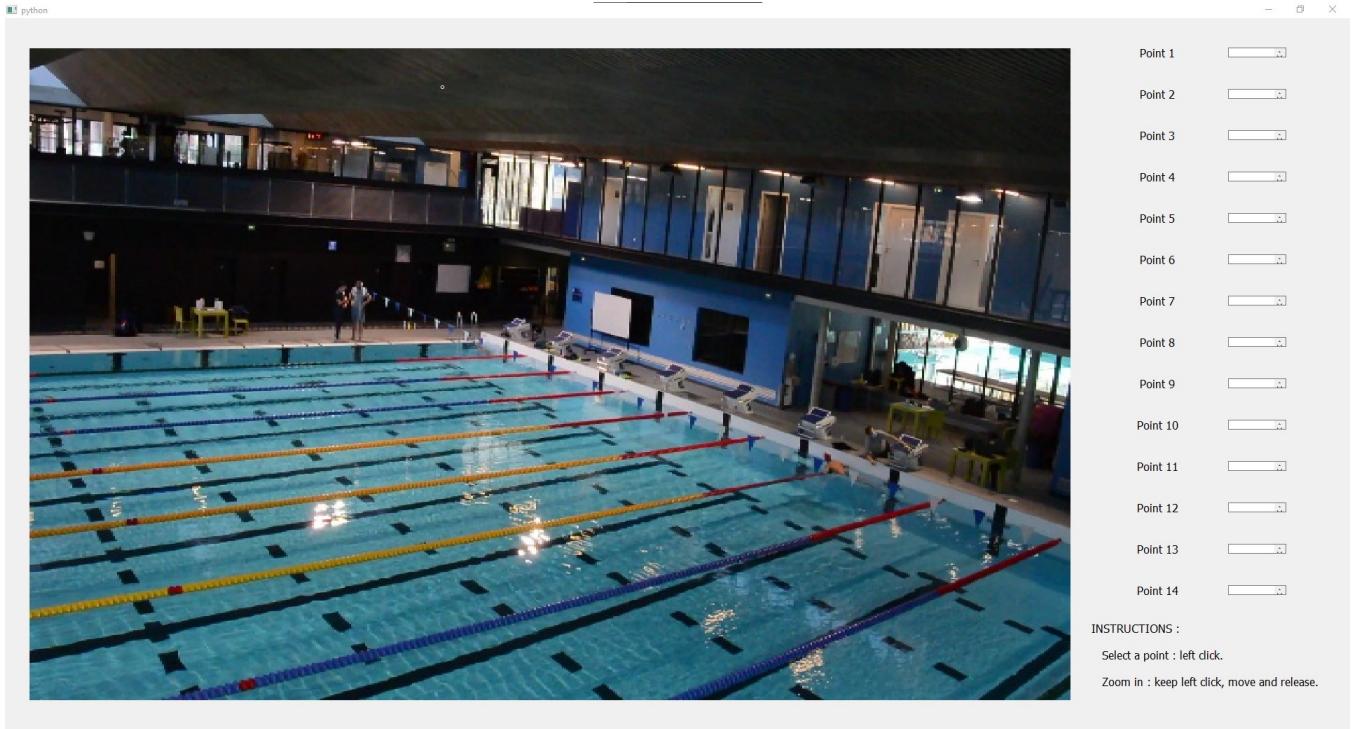


FIGURE 24 – Interface graphique de pointage manuel

### 3.5.3 Data augmentation

La "*Data Augmentation*" est une technique utilisée pour augmenter le nombre de données en ajoutant des copies légèrement modifiées de données déjà existantes. Cette technique permet au réseau entraîné d'avoir une meilleure robustesse.

Notre base de donnée étant composée d'images, 4 légères modifications parmi les plus classiques ont été retenues (voir Figure 25) :

- une rotation par rapport au centre de l'image
- une symétrie par rapport à l'axe vertical central
- un décalage de quelques dizaines de pixels
- l'ajout d'un léger bruit gaussien sur les trois canaux

On effectue ces opérations sur les images d'une part et sur les labels d'autre part, en s'assurant de garder les mêmes dimensions (512x512). Au total, le nombre d'éléments a été multiplié par cinq, on obtient donc une base de 1145 (5x229) données. On peut donc maintenant commencer à entraîner le réseau.

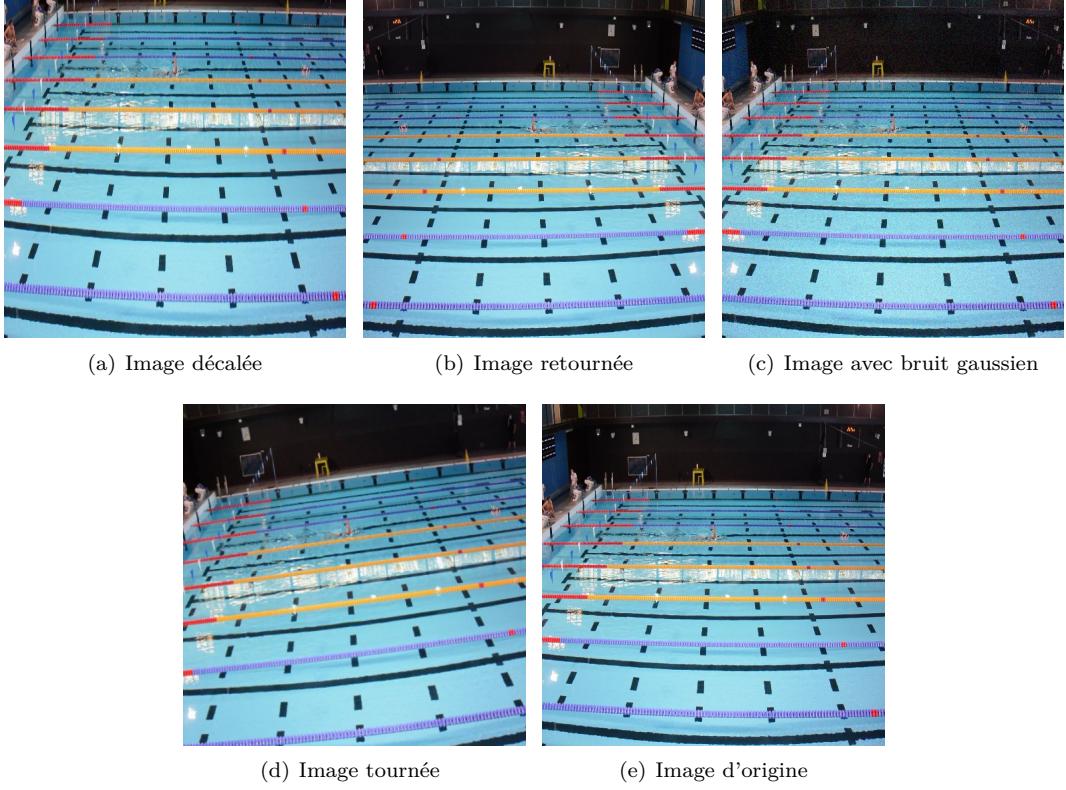


FIGURE 25 – Data augmentation

### 3.5.4 Résultats

Notre objectif étant de repérer des classes de marqueurs de couleur sur une image donnée en entrée, il est primordial pour assurer des résultats pertinents et utilisables par la suite dans la détermination des homographies de disposer d'un large pannel de données et de labels. Cela permet d'éviter un sur-apprentissage en entraînant sur des données similaires et une trop faible *accuracy* (pourcentage de prédictions justes) en sortie de réseau. Or nous ne disposons d'aucune donnée applicable à notre modèle dans la base de données créée par nos prédecesseurs. Aussi, la plus grande difficulté à laquelle nous avons été confrontée est ce manque de données. La construction de la base étant manuelle, nous n'avons pu entraîner le réseau UNet que sur 800 données (800 données d'entraînements et 345 de validation), ce qui s'avère être insuffisant pour obtenir de résultats convainquants.

La data augmentation (voir la figure 25) a néanmoins permis d'atteindre le millier de données et d'observer un apprentissage via les courbes de loss. On remarque ainsi la diminution de la loss au fur et à mesure de l'entraînement. On observe ci-dessous les courbes de loss sur les ensembles d'entraînement et de validation. On récupère les valeurs de loss toutes les 10 itérations (donc 60 itérations pour l'entraînement, contre 30 pour la validation).

```

Epoch 0
Iteration 0, loss = 1.0974
mean loss on the loader 1.2944035530090332

Iteration 10, loss = 0.1055
mean loss on the loader 0.288703978061676

Epoch 1
Iteration 0, loss = 0.0763
mean loss on the loader 0.22236531972885132

Iteration 10, loss = 0.0563
mean loss on the loader 0.15574806928634644

Epoch 2
Iteration 0, loss = 0.0458
mean loss on the loader 0.13615649938583374

Iteration 10, loss = 0.0392
mean loss on the loader 0.10857298970222473

```

FIGURE 26 – Valeurs de loss obtenues avec 3 epochs pour 1145 données avec des batch de tailles 64

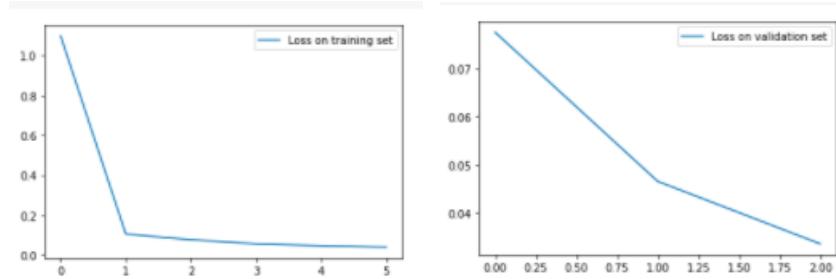


FIGURE 27 – Courbes de loss pour 60 itérations sur l’ ensemble d’entraînement (a) et 30 itérations sur l’ensemble de validation (b)

En sortie du réseau on récupère 8 cartes de probabilité de taille  $512 \times 512$  (soit la résolution des images d’entrée, qui peut varier pour des raisons de mémoire saturée) où la carte  $i$  est telle que chaque pixel a pour intensité la probabilité d’appartenir à la classe  $i$ . On peut voir ci-dessous un exemple de prédiction comparé au label réel correspondant.

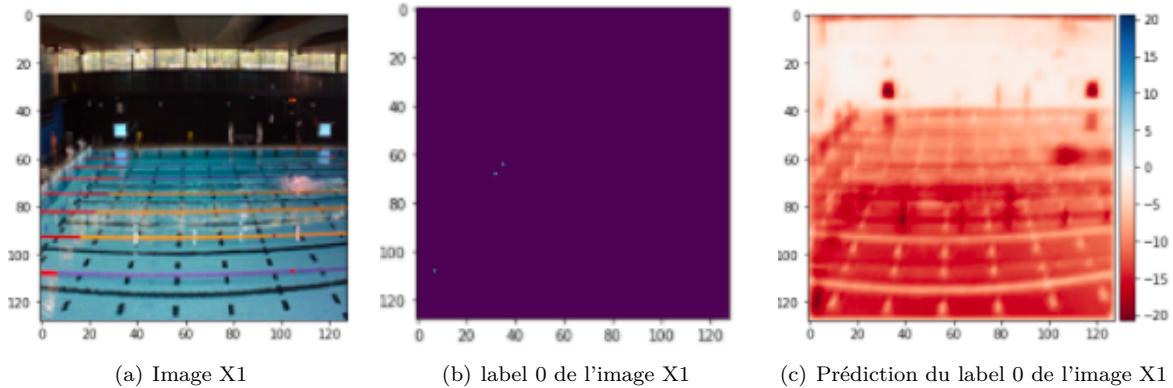


FIGURE 28 – Visualisation de l’image X1, du label 0 et de la prediction associée

Par ailleurs, il a été compliqué de se servir de la base de données avec une résolution  $512 \times 512$  car la mémoire vive (Random Acces memory) se saturait systématiquement (voir 29). Aussi nous avons utilisé une résolution de  $128 \times 128$  et c’est ce que l’on observe ci-dessous figure 28.

```

-----  

RuntimeError                               Traceback (most recent call last)  

<ipython-input-13-47797271f1> in <module>()  

      4     optimizer = optim.SGD(model.parameters(), lr=learning_rate)  

      5  

----> 6 loss_train, loss_val = train.module(model, optimizer, epochs=1)  

      7  

      8 frames  

/usr/local/lib/python3.7/dist-packages/torch/mnn/modules/conv.py in _conv_forward(self, input, weight, bias)  

  394         pair(0), self.dilation, self.groups)  

  395         return F.conv2d(input, weight, bias, self.stride,  

--> 396             self.padding, self.dilation, self.groups)  

  397  

  398     def forward(self, input: Tensor) -> Tensor:  

RuntimeError: CUDA out of memory. Tried to allocate 4.00 GiB (GPU 0; 11.17 GiB total capacity; 8.43 GiB already allocated; 2.24 GiB free; 8.51 GiB reserved in total by PyTorch)

```

FIGURE 29 – Saturation de la RAM après chargement de la base de données

### 3.6 Pistes d'améliorations

Les résultats obtenus sont encourageants car on observe sur la prédiction les motifs de l'image initiale (lignes d'eau, nageurs, contours de la piscine s'ils sont présents). Cela indique un bon apprentissage de la part du réseau ; cependant, les pixels particuliers qui correspondent à une classe donnée ne sont pas identifiables. Si sur un label donné seuls trois points sont de couleur différente comparée à celle du fond (blanc sur fond noir ou vert sur fond violet), ce n'est pas le cas en sortie du réseau.

Une des critiques à apporter à notre modèle est le manque de diversité de la base de données créée. Un travail conséquent à apporter pour améliorer la prédiction est donc de l'étoffer. De surcroît, on peut songer à non plus considérer trois pixels par classe mais plutôt trois zones (4 pixels par zone par exemple) pour chaque classe ce qui permettrait une meilleure détection de ces dernières. On pourrait même adapter le nombre de pixels par zone au marqueur de couleur repéré sur l'image. La labellisation n'en serait que plus précise et pourrait permettre d'améliorer le modèle.

Si ces obstacles sont surmontés, le fait que le réseau apprenne (observation des courbes de loss) laisse présager une amélioration du modèle. Par la suite, il s'agira de repérer les points les plus pertinents qui permettront de déterminer la transformation de perspective à appliquer. L'analyse par composantes principales (PCA) est une méthode permettant de repérer de tels points. A terme, il est souhaitable d'appliquer ce processus sur une vidéo afin de corriger la perspective sur toutes ces frames.

## 4 Détection des phases de nage

Lors de cette dernière partie, nous nous sommes focalisées sur l'obtention d'une des statistiques du nageur. Notre travail s'est donc basé sur ce besoin pour générer deux choses :

- un réseau de neurones pour détecter la nage de la vidéo : le nageur est-il en train de nager du crawl, de la brasse, ... ? Cet algorithme, pas spécifique au besoin énoncé plus haut, a une application plus générale.
- un algorithme permettant de détecter les cycles de bras du nageur, permettant à ce dernier de connaître la durée de chacun de ses cycles et agir en conséquence.

Ces deux points seront traités dans les sous deux parties qui suivent.

### 4.1 Détection de la nage

#### 4.1.1 Préparation de la base de données

##### Données utilisées

Pour détecter la nage, nous avions besoin d'images croppées autour du nageur pour éviter que notre réseau de neurones ne s'entraîne sur des données trop précises.

Nous possédions trois vidéos de crawl de l'équipe de l'année dernière, découpées par image et avec le `csv` contenant toutes les positions de la tête dans chacune des frames. En plus de cela, nous avons utilisé 16 nouvelles vidéos (4 dans chacune des nages, avec des nageurs féminins et masculins) dans l'algorithme de l'année dernière. Ce dernier permet entre autre de lire une vidéo frame par frame, de pointer la tête sur chacune d'elle, et d'enregistrer à la fin le `csv` avec ces positions ainsi que les images associées dans le dossier `data/2_intermediate_top_down_lanes/lanes` (cf partie précédente, figure 14).

Nous avons donc pu générer pour chacune des vidéos des images vues de haut, ligne d'eau par ligne d'eau, avec une position de tête associée, sur lesquelles nous nous basées pour créer la base de données.

*Remarque* : Chaque vidéo filme une piscine avec un nageur par ligne d'eau, d'où la nécessité de couper frame par frame pour se concentrer uniquement sur un nageur.



FIGURE 30 – Vidéo `backstroke_m_left` - Ligne 1 - Frame n°2140

##### Première difficulté

Nous nous sommes donc dans un premier temps penchées sur la récupération de la position de la tête. Cette dernière était soit donnée par le `csv`, soit à déduire par interpolation car les nouvelles vidéos avaient été pointées toutes les 10 frames pour éviter de perdre du temps. C'est là que nous avons rencontré notre premier problème : les noms de frames générées par l'application de l'équipe de l'année lors du pointage rencontraient des sauts. Les noms des frames présentaient tous la même structure : "`l`" + *numéro de la ligne* + "`f`" + *numéro de frame*. Par exemple, l'image `l1_f2140` correspond à la ligne d'eau numéro 1 et l'on regarde l'image 2140 de la vidéo. Et nous pouvions passer de la frame n° 2140 à la frame n°2220 alors que les images en elles-même se suivaient dans la vidéo. Il y a eu une erreur dans l'algorithme de l'équipe de l'année dernière, mais nous n'avons pas su la corriger directement dans leur code, qui était trop dense. Et cette erreur devait être résolue car nous nous basions sur le nom des frames pour pouvoir faire notre interpolation.

Nous avons donc procédé de cette manière : nous avons gardé les frames générées par l'algorithme de l'année dernière, et nous avons créé une fonction qui détecte et corrige les erreurs dans la succession des numéros des images directement dans le dossier. Nous pouvions ainsi générer les positions de la tête manquantes par interpolation et avoir ainsi une toute nouvelle liste de positions correspondant à chacune des images.

##### Croppage

Les données ainsi triées et ayant chacune leur label, nous avons donc pu créer une fonction croppant et enregistrant automatiquement les images. Nous prenions 200 pixels avant et après la position de la tête pour obtenir l'image du

nageur (nous n'avons pas pris en compte le sens de nage pour réduire le nombre de pixels devant la tête du nageur, car sur certaines images le pointage est approximatif et pouvait engendrer un coupage de bras lors du crop).

Toute notre base de données a été enregistrée dans le dossier `2021/detection_nage/data/cropped_images` et dans le dossier de la nage associée.



FIGURE 31 – Image croppée de `l1_f2140` de la vidéo `backstroke_m_left`

#### 4.1.2 Le réseau de neurones

##### Motivation

Au départ, nous hésitions entre deux méthodes. Nous avions pour idée d'utiliser les courbes de positions du nageur et d'en déduire la vitesse. Nous voulions ensuite implémenter un réseau de neurones qui, à partir de ces vitesses, serait capable de déterminer le type de nage. La deuxième idée était de créer un classifieur à partir d'un réseau de neurones, qui prendrait en entrée une image et qui donnerait sa nage en sortie.

Nous avons alors décidé de nous pencher sur la méthode du classifieur pour plusieurs raisons. D'une part, étant donné que l'on doit prédire une nage parmi quatre, il semblait naturel de faire une classification. D'autre part, les images brutes contenaient beaucoup plus d'informations que des courbes de positions, d'autant plus que celles-ci étaient déjà obtenues à partir d'un réseau de neurones, et constituaient donc des données trop bruitées pour entraîner un nouveau réseau de neurones. Enfin, l'équipe précédente qui travaillait sur le projet avait implémenté un réseau de neurone pour classifier la position de la tête du nageur dans le bassin. Notre problème se rapprochant de celui-ci, et ayant une base de données proches de l'équipe précédente, nous avons décidé d'implémenter également un réseau de neurones pour classifier les nages.

Néanmoins l'architecture de notre réseau diffère de celle de l'équipe précédente. En effet, du fait de la précision que demande la détection de la tête du nageur, l'équipe qui travaillait avant nous sur le projet a décidé de couper chaque ligne d'eau en dix zones, et de faire un classifieur pour déterminer dans laquelle de ces dix zones se trouvait la tête. Or notre projet s'avère à priori plus simple, puisque nous n'avons besoin que d'un classifieur de quatre nages, et que nous disposons de plus de données. Il nous est alors possible de créer un réseau de neurone assez simple, et dans le même temps assez efficace, et c'est pour cela que nous avons décidé de réaliser un réseau de neurones convolutionnel à deux couches. L'avantage d'un réseau de neurones convolutionnel par rapport à un réseau de neurones linéaire est que celui-ci est beaucoup plus adapté à un problème avec des images en entrée. En effet, une couche convolutionnelle correspond à l'application d'un ou plusieurs filtres sur une image, les filtres étant généralement de petite taille. Cela permet de créer des réseaux de neurones avec des couches de petites dimensions, et étant capables d'entraîner des données de grande dimensions, comme des images.

##### Architecture du réseau

Notre réseau de neurones convolutionnel est alors constitué de deux couches, avec en entrée des batchs de 64 images de taille  $27 \times 50 \times 3$ . Le réseau étant simple, il s'avérait alors nécessaire de rajouter des outils pour éviter le sur-apprentissage. Nous avons donc normalisé les données avec la fonction 'Batch Normalisation' juste avant les fonctions ReLu. De plus, nous avons rajouté une couche de Max Pooling afin de diminuer la taille de l'image tout en conservant ses caractéristiques. Cela se passe via une fenêtre, qui va glisser pas à pas sur l'ensemble de l'image, et ne récupérer que certaines valeurs de l'image, soit en prenant la valeur maximale des pixels de la région analysée par cette fenêtre. L'architecture de ce réseau est illustré figure 32.

Enfin, la loss que nous avons choisi d'utiliser est la *Cross Entropy Loss*, qui est une loss classique lorsqu'on entraîne un classifieur :

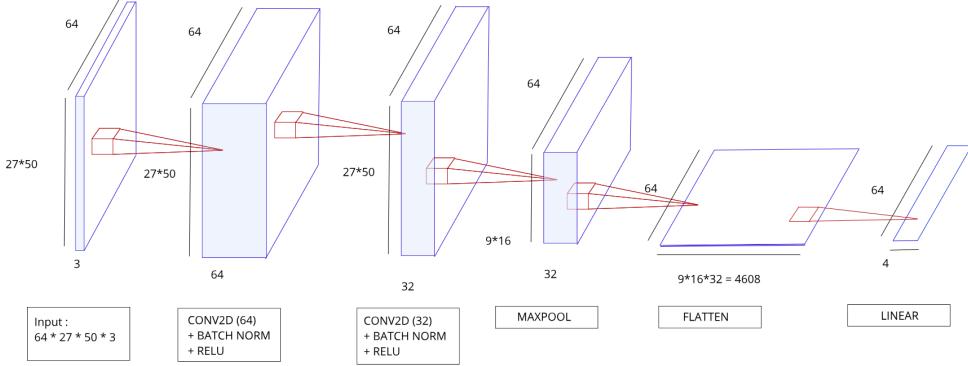


FIGURE 32 – Architecture de notre réseau de neurones

$$\text{loss}(x, \text{class}) = -\log \left( \frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])} \right)$$

#### 4.1.3 Résultats

##### Résultats sur l'ensemble de validation

Concernant l'ensemble de validation, les résultats sont très bons : la loss est de 0.15 tandis qu'il y a 98% d'*accuracy* (pourcentage de bonnes classifications). Les résultats sont répertoriés sur le graphique de la figure 33. La courbe bleu correspond à la loss sur l'ensemble de train, tandis que la courbe jaune correspond à la loss de l'ensemble de validation. On constate qu'au fur à mesure des itérations sur le train, la courbe de loss de l'ensemble de validation diminue. Bien qu'il y ait un écart entre la loss de train et la loss de validation, cet écart est assez faible, et il n'y a pas d'augmentation significative de la loss de validation au fur et à mesure des itérations. On peut donc en conclure que l'over-fitting est assez limité.

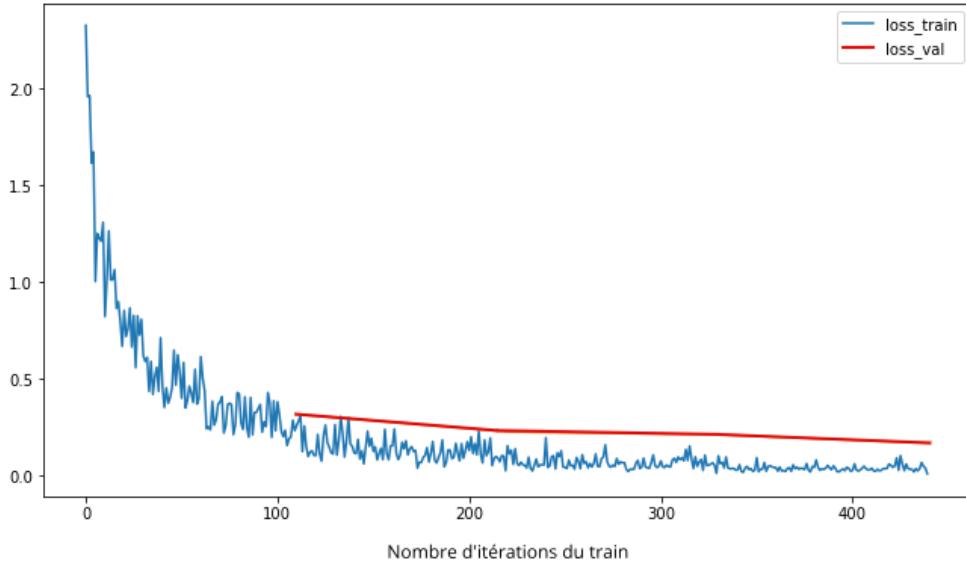


FIGURE 33 – loss en fonction du nombre d'itérations

##### Résultats sur différents tests

Nous avons ensuite fait un test sur une vidéo qui n'avait jamais été vue par l'algorithme. Le tableau ci dessous présente l'*accuracy*, c'est à dire le pourcentage de bonnes classifications, en fonction de la ligne d'eau :

Ligne d'eau	1	2	3	4	5	6	7
Accuracy (%)	28,7	61.74	76.09	91.30	73.48	62.61	98.26

TABLE 1 – Accuracy en fonction de la ligne d'eau

On constate que le résultat d'*accuracy* du test varie beaucoup en fonction de la qualité des photos de ce test. En effet, plus la ligne d'eau est loin de la caméra, plus la qualité de l'image est floue, et déformée du fait de la calibration. De plus, plus un nageur est sur une ligne d'eau éloignée de la caméra, plus il est filmé de coté, et non de haut. Il y a donc d'avantage d'éclaboussures qui rajoutent du bruit à l'image.



FIGURE 34 – Echantillon d'images de la ligne d'eau 1



FIGURE 35 – Echantillon d'images de la ligne d'eau 5



FIGURE 36 – Echantillon d'images de la ligne d'eau 7

Enfin, nous avons réalisé un test sur une vidéo de 4 nages, le but étant que l'algorithme réussisse à repérer les instants de changement de nage. Nous avons décidé d'effectuer ce test sur un nageur de la ligne 1, dont les images sont de moins bonne qualité que les autres, afin de tester la robustesse de notre réseau à des données de mauvaise qualité.

Pour détecter les phases de nage au cours des quatre traversées, nous nous sommes basées sur la remarque qu'un nageur ne changeait pas de nage au milieu d'une traversée. Pour repérer les phases de nages, il fallait alors repérer les plages de frames où chaque nage était la plus dominante. Pour cela, nous avons découpé nos frames de test en batchs de la même taille que les batchs de train, et nous avons appliqué notre réseau de neurones dessus. La nage prédite sur chaque batch était alors la nage prédite la plus représentée parmi les frames de chacun de ces batchs.

Les résultats sont alors les suivants :

- l'accuracy par frame est de **73%** (2845 / 4020 frames bien classifiées)
- l'accuracy par batchs est de **79.4%** (50/63 batchs bien classifiés)

## Discussion sur les autres inputs possibles

Pour rendre l'algorithme plus efficace et plus robuste à la luminosité et au changement de piscine, nous avons testé le modèle avec trois inputs différents :

- Nous avons utilisé comme input la différence de deux images, pour rendre l'algorithme plus robuste aux changements de luminosité. Néanmoins, le modèle dépassait difficilement les 50% d'accuracy sur l'ensemble

de validation, peut être car faire la différence de deux images enlève de l'information, ou que le réseau de neurones n'était pas adapté à ce genre de données.

- Nous avons défini comme input non pas une image mais une suite de quatre images successives. Les résultats sont similaires aux résultats avec en input une seule image, mais les itérations du modèle sur le train sont beaucoup plus longues. Cela peut s'expliquer par le fait que sur ces quatre images, il y a quasiment la même information. Par conséquent, on obtient les mêmes résultats en donnant au modèle des données de plus grande dimension, ce qui est inutile.
- Enfin, puisque l'idée de quatre images consécutives n'offrait pas beaucoup d'amélioration par rapport à des inputs d'une seule image, nous avons décidé de prendre en input quatre images aléatoires de la même nage. En effet, on peut supposer qu'alors le modèle aura beaucoup plus d'informations en entrée pour déterminer la nage de ces images. Il s'est en fait avéré que l'entraînement du réseau de neurones était très long, et que, bien que les résultats étaient de bonne qualité, ceux-ci étaient moins bons que sur le modèle à une seule image, et c'est pour cela que nous avons retenu le modèle initial.

## 4.2 La détection des cycles de bras en crawl

### 4.2.1 Synchronisation des données gyrométriques avec les images

#### Données gyrométriques

Se baser uniquement sur des images pour repérer les cycles de bras était insuffisant pour essayer de faire apprendre un réseau de neurones. Heureusement, comme expliqué dans le paragraphe 1.2, des essais en bassin olympique à l'INSEP de nageurs équipés de capteurs ont été réalisés dans le cadre de l'ARN NePTUNE. Les nageurs étaient équipés de fils tendus permettant d'obtenir la position et la vitesse instantanée, ainsi que de capteurs sur les bras et les poignets permettant d'obtenir la vitesse angulaire. Tous ces essais étaient filmés, et les nageurs nageaient uniquement du crawl.

En récupérant ces données, voici par exemple ce que nous pouvons obtenir :

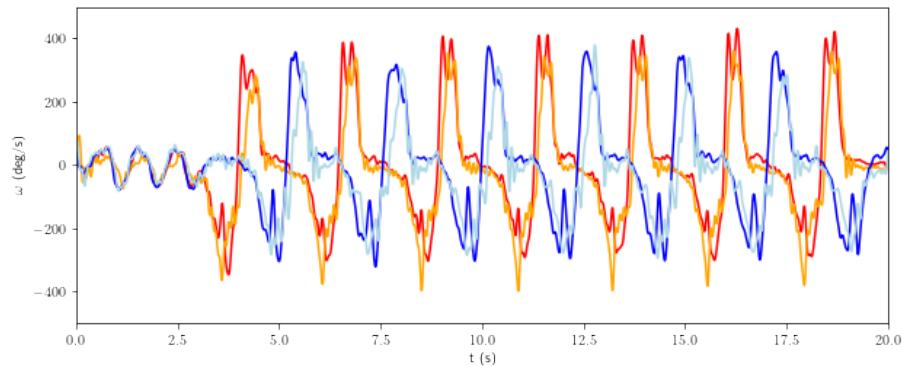


FIGURE 37 – Visualisation des données gyrométriques lors de l'aller d'un nageur

Les courbes bleues correspondent aux données du bras et du poignet droits, tandis que les données rouges correspondent à celles du bras et du poignet gauche. On observe ainsi au début que le nageur, après avoir poussé le mur, réalise trois ondulations puis commence à nager. Les parties négatives correspondent aux phases sous-marines, et les pics positifs aux moments où le nageur a le bras levé.

Notre but était donc d'associer à chaque frame de la vidéo une de ces données gyrométriques, qui sont presque continues. Pour ce faire, il fallait déterminer le temps de chaque frame et l'associer au temps le plus proche dans le csv des données gyrométriques, lui attribuant ainsi une donnée. Nous voulions initialement reprendre l'algorithme de l'année dernière de génération des frames à partir d'une vidéo. Cependant, comme mentionné précédemment, cet algorithme rencontrait des problèmes de nommage dans le numéro des frames et dans le timing associé à chacune d'elles. Nous avons donc tout repris de zéro en nous basant sur le code de synchronisation des données de fil tendu avec les images, créé par le groupe de Manh-Dan et Charles détaillé dans le paragraphe 2.1.2.

## La synchronisation en pratique

Nous possédions ainsi un lot de 40 vidéos environ, avec pour chacune des données gyrométriques et de fil tendu associées (utile pour plus tard).

De la même manière que dans la partie 2.1.1, nous avons calibré chaque vidéo en pointant le repère des 5 et des 15 mètres sur la piscine, ce qui nous a permis d'obtenir la matrice de perspective pour pouvoir passer des coordonnées réelles aux coordonnées dans l'image et vice-versa, et d'obtenir la *top-down view*. Nous avons enregistré la matrice de calibration dans un fichier `.txt` dans le dossier *calibration* de chacun des nageurs (*N0001*, *N0003*, *N0004* et *N0005*).

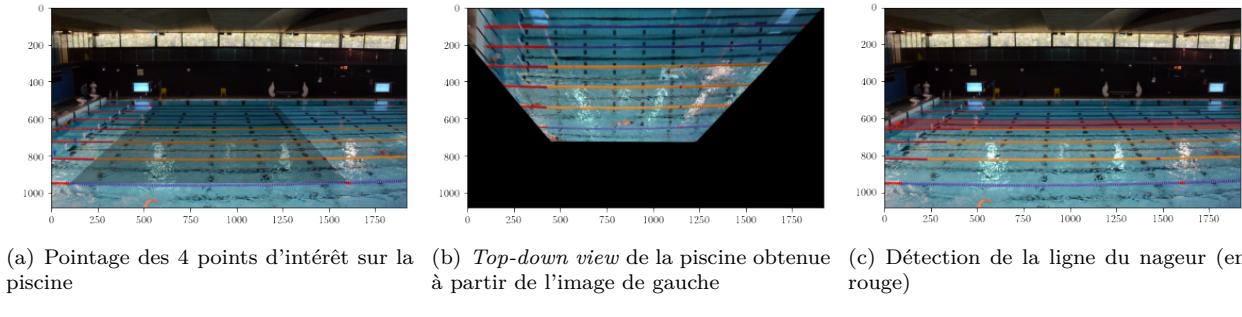


FIGURE 38 – Les différentes étapes de la calibration d'une vidéo

Nous avons donc appliqué la *top-down view* sur chacune des images générées, en ayant pris soin de ne garder que la ligne où le nageur nageait (il n'y a qu'un seul nageur par vidéo dans la partie détection de cycles) et de cropper autour du nageur, en utilisant la position de la tête grâce aux données du Fil Tendu.

Mais pour générer ces images, il fallait donc leur associer en même temps un temps et une donnée gyrométrique. Le fait de générer les images une par une à partir de la vidéo nous permettait d'obtenir facilement le timing de la frame, mais le temps de début de la vidéo n'était pas le même que celui du gyromètre. Il a donc fallu trouver à la main, pour chacune des vidéos, la frame qui correspondait au début des données gyroscopiques (et qui diffère de celle trouvée à la main pour le fil Tendu). Pour essayer de les synchroniser au mieux, nous avons détecté sur la courbe des données gyrométriques du bras gauche les pics positifs, et regardé l'image associée dans un premier temps en prenant un numéro de frame de départ de manière intuitive. Puis nous ajustions en fonction de l'écart au numéro de frame que nous avions repéré en enregistrant une première fois ces images dans un dossier *test*.

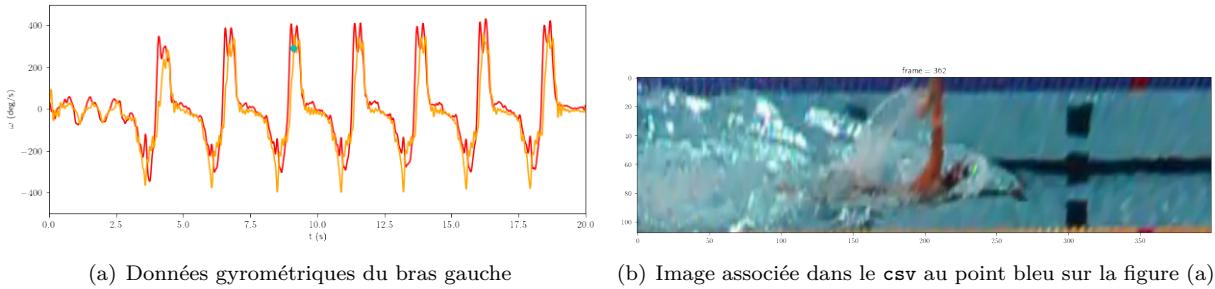


FIGURE 39

FIGURE 40 – Synchronisation des données gyrométriques avec les images

Nous avons ainsi réussi à créer un csv de colonnes :

temps	frame	gyro Poignet Gauche	gyro Bras Gauche	gyro Poignet Droit	gyro Bras Droit
-------	-------	---------------------	------------------	--------------------	-----------------

### 4.2.2 Préparation des données synchronisées

Nous avons récupéré les fichiers `.csv` générés lors de la sous-partie précédente pour attribuer un label à chacune de ces images.

Nous connaissons ainsi la donnée gyrométrique de chaque image et pouvons déterminer si le bras est en haut, sous l'eau, etc. Mais nous ne pouvons pas mettre directement ces données gyrométriques en labels car chaque amplitude de bras est différente d'un lever de bras à l'autre, et diffère beaucoup entre les nageurs. Nous avons donc eu l'idée d'essayer de les répartir selon une gaussienne, mettant les images à 1 quand le bras était en haut, et les autres se répartissant tout le long de la courbe.

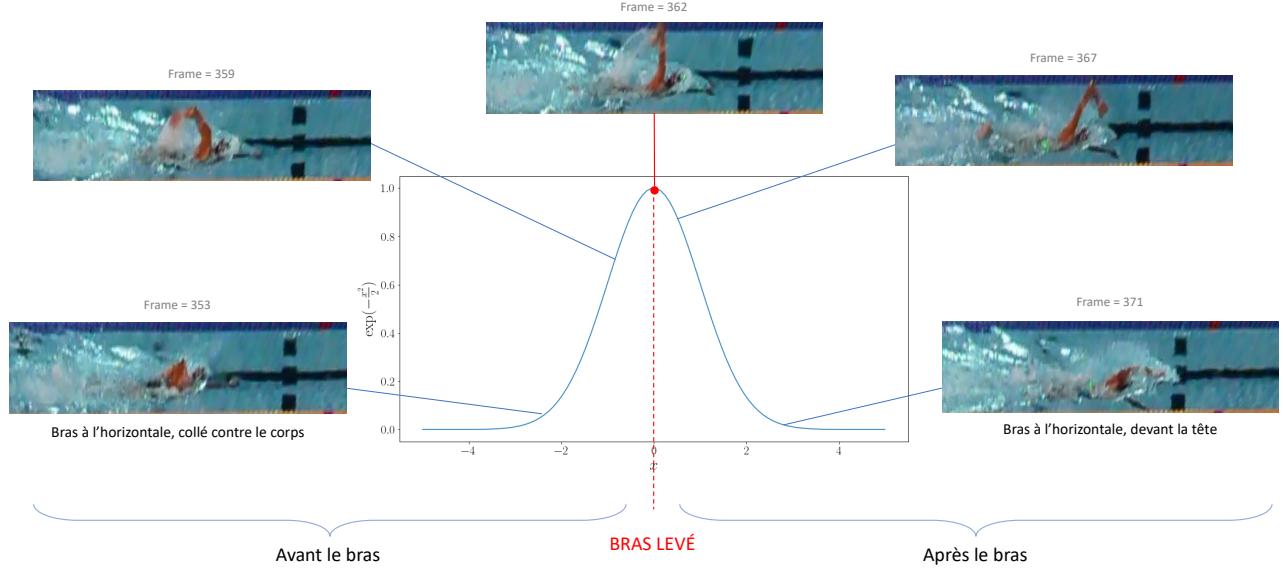


FIGURE 41 – Explication de la labellisation souhaitée par une gaussienne

Pour trouver la gaussienne, nous avons d'abord détecté le pic le plus haut lors de la nage et nous l'avons pris en pic de référence (i.e. c'est celui qui vaudra 1). Puis nous avons testé différentes fonctions :

$$g_1(x) = \exp\left(-\frac{(x - a)^2}{2}\right)$$

$$g_2(x) = \exp(-|x - a|)$$

où  $a$  est la valeur du pic de référence.

En prenant la vidéo *DSC\_7392* du dossier *N0001* comme test, dont les données gyrométriques sont visibles dans la figure 37, nous obtenons la figure 42 pour la fonction  $g_1$ , et la figure 43 pour la fonction  $g_2$ .

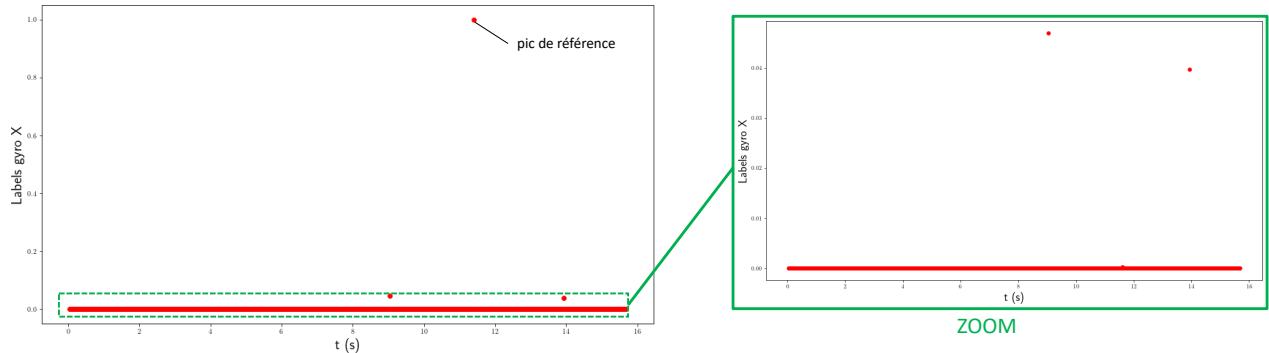


FIGURE 42 – Labels avec  $g_1$

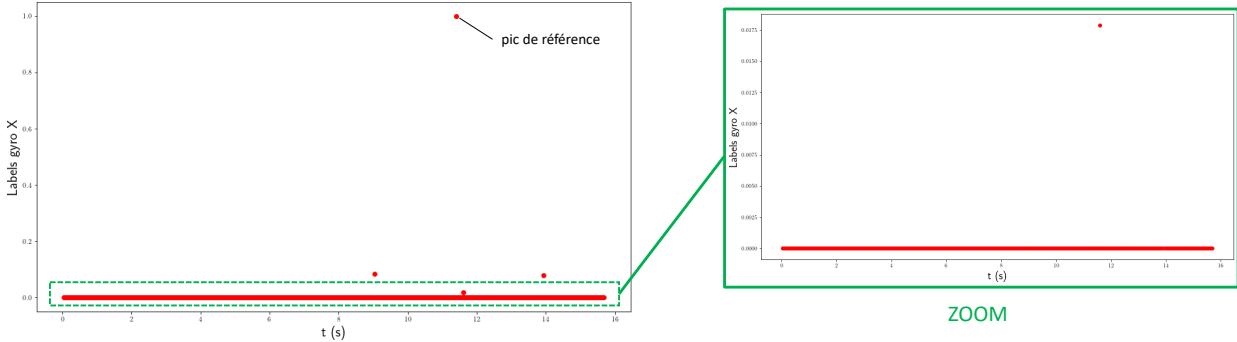


FIGURE 43 – Labels avec  $g_2$

On observe que le pic de référence est bien mis à 1, mais ces deux fonctions peinent énormément à détecter les autres pics, et les seuls points qui arrivent à se détacher, dont la donnée gyroscopique est proche de  $a$ , sont quand même très éloignés du pic et proches de 0. On voulait pourtant que les autres pics et les points proches de ces pics soient aux alentours de 1, ce qui n'est donc absolument pas le cas ici.

Ces deux fonctions ne parviennent pas à différencier suffisamment les pics et à effectuer une bonne répartition. Ceci peut s'expliquer par le fait que les valeurs d'écart avec le pic de référence sont amplifiées par l'exponentielle, qui tend vers 0 vers  $-\infty$ . D'où la volonté d'enlever le carré à l'intérieur de l'exponentielle dans  $g_2$ , mais les résultats n'en sont pas plus convaincants.

En prenant finalement  $g(x) = \exp(-(\frac{x-a}{a})^2)$ , on obtient une répartition cohérente :

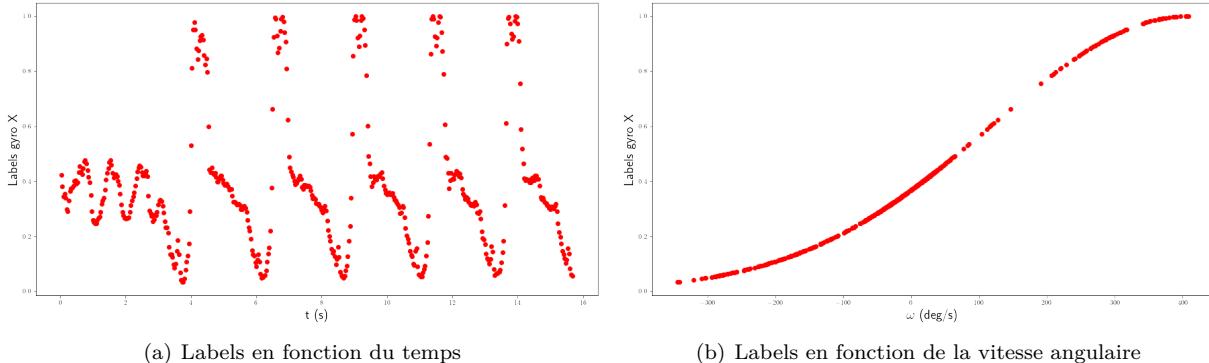


FIGURE 44 – Labels avec  $g$

La forme de la courbe obtenue en figure 44 ressemble bien à celle des données gyrométriques (figure 37), donc notre fonction détecte bien les différents pics. Et on observe bien sur la figure 44 que les labels sont bien répartis de manière uniforme entre 0 et 1.

Nous avons donc créé l'ensemble des labels pour le réseau de neurones grâce à cette fonction, que nous avons stocké dans de nouveaux fichiers .csv dans le dossier *labels*.

#### 4.2.3 Apprentissage

##### Stratégie pour le repérage de cycles

Après avoir affecté un label à chaque image, comme expliqué dans la partie précédente, nous voulions trouver un modèle qui, à partir des images et de leurs labels affectés, arrive à prédire les labels de nouvelles données. Il est à noter que la précision de l'output est dans cette partie là bien plus importante que pour le classifieur, puisque sans

cela le repérage de cycle serait approximatif. A partir des labels prédicts sur une vidéo, il serait alors possible de retrouver tous les labels maximums consécutifs. Le temps qui sépare deux labels maximums correspond au temps entre deux positions de bras levé, et donc correspond à un cycle de nage. Un exemple de repérage de cycle à partir des labels figure ci-dessous, les points rouges étant les labels maximums locaux. Chaque image correspond à un instant du cycle :

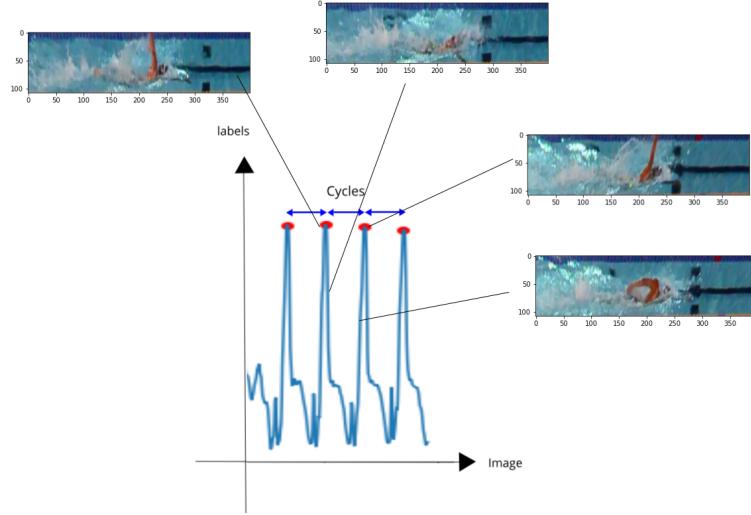


FIGURE 45 – Repérage de cycles à partir des labels

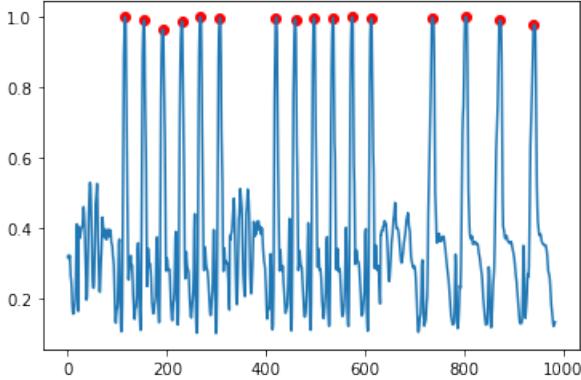
## Modèle

Notre première idée fut de s'inspirer de la première partie de notre travail, c'est à dire le classifieur de nage. Nous voulions créer un réseau de neurone pour prédire le label de chaque image. Nous avons alors tenté de créer un réseau de neurones convolutionnel à deux couches, mais avec cette fois ci une seule sortie, suivie d'un softmax pour obtenir un score entre 0 et 1. Malheureusement, nous n'avons pas réussi à obtenir de résultats.

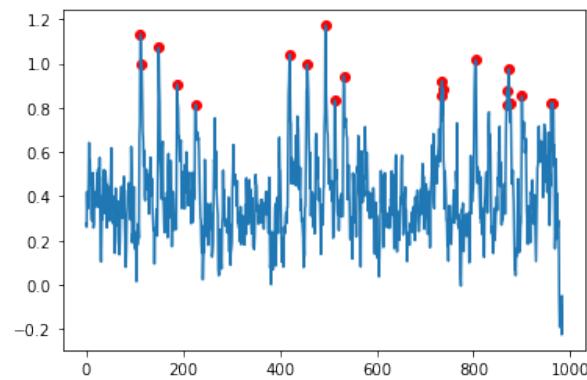
Nous nous sommes alors penchés non pas sur du deep learning, mais sur un modèle de regression linéaire. Nous pensons en effet que, bien qu'un réseau de neurones pourrait donner des résultats plus précis, une régression linéaire pourrait donner en un temps bien plus rapide des résultats satisfaisants. En effet, l'important n'est pas d'obtenir exactement les bons labels, mais d'avoir les même position des maxima locaux, puisque ce sont ces positions qui nous permettent de détecter les cycles.

## Résultats

Notre régression linéaire, entraînée sur 1000 images, est de bonne qualité puisque son coefficient de détermination  $r^2 = 0.99$ . Bien que la figure des labels prédicts (cf ci-dessous) ne ressemblent pas à la figures des vrais labels, nous pouvons tirer des résultats satisfaisant de ce modèle.



(a) Labels sur la vidéo test



(b) Labels prédits sur la vidéo test

FIGURE 46 – Comparaison entre labels attendus et prédits sur une vidéo test

En effet, au vu de la figure, il est possible de retrouver la plupart des maxima locaux, et donc la plupart des cycles. Par exemple, sur les deux premiers groupes de 6 oscillations sur la figure 46, on voit que sur la figure 46 le modèle arrive à détecter 4 cycles, puis 5 cycles.

De plus, comme on peut le voir sur la figure 47, on a cherché les images qui correspondaient aux maxima locaux trouvés par le modèle. Ce sont les images qui apparaissent autour de la figure. On constate que ces valeurs maximales prédites par la régression linéaire correspondent bien à des positions de bras levés. Ainsi, bien que tous les cycles ne soient pas détectés, nous obtenons des résultats très satisfaisant sur ceux qui le sont.

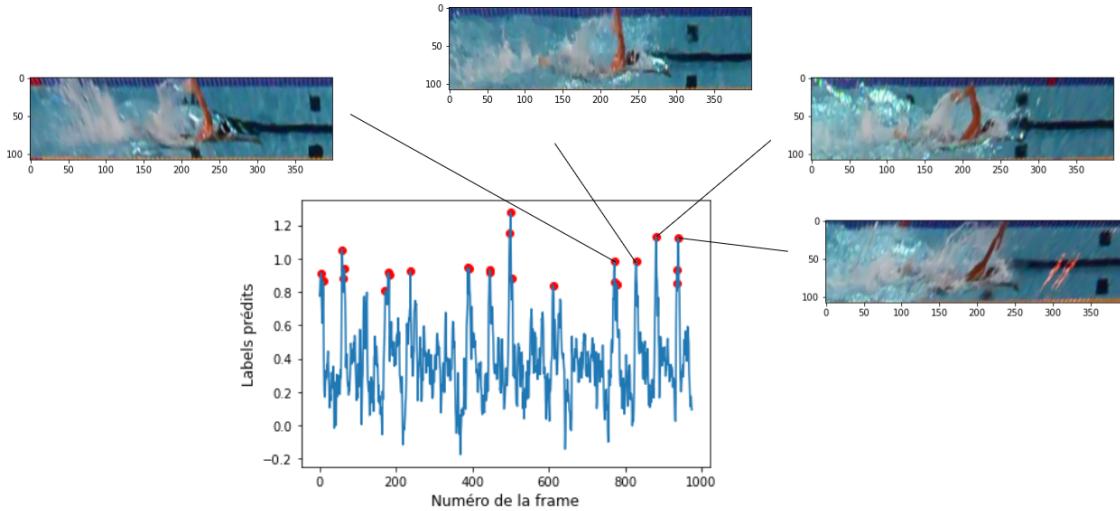


FIGURE 47 – Repérage de cycles à partir des labels

### 4.3 Pistes d'amélioration

Bien que nos modèles donnent des résultats satisfaisants sur les objectifs que nous voulions atteindre, quelques points demanderaient à être améliorés :

- Concernant la détection du type de nages, nous avons obtenu un résultat satisfaisant par image et par batch sur le classifieur (supérieur à 70%). Néanmoins, lorsqu'on lui donne une suite d'images sur lesquelles il y a plusieurs phases de nages qui s'enchaînent, notre modèle n'est pas capable de détecter la frame de changement de nage. Pour palier à ce problème, il faudrait créer un deuxième réseau de neurones, qui prendrait en entrée les suites de nage prédites par frames, et qui donnerait les frames correspondant aux changements de nage.

- Pour ce qui est de la détection de cycles de nage, comme nous venons de le voir, nous obtenons de bons résultats sur les cycles détectés. Or tous les cycles ne sont pas détectés, et certains maxima sont détectés alors qu'ils ne correspondent pas à des bras levés (par exemple sur le dernier pic figure 47). Il serait donc intéressant de raffiner le modèle en combinant à la fois régression linéaire et méthodes de Deep Learning, qui peuvent s'avérer très efficaces sur des images.
- Concernant l'algorithme de synchronisation des données gyrométriques avec les images de la vidéo, il faut au départ trouver soi-même un ordre de grandeur pour la frame de départ : repérer sur la vidéo la seconde où le nageur est en train de pousser sur le mur, et comme l'on connaît le *frame per second* (FPS) de la vidéo dans le notebook (cf Annexe C), on peut en déduire un premier numéro pour la frame de départ. Et lors de l'enregistrement frame par frame dans le dossier *test*, il est possible de visualiser les images en même temps qu'elles s'enregistrent, et ainsi mémoriser le numéro de la première frame où le bras gauche est à la verticale. Vous pourrez ainsi en déduire le décalage de frames, en regardant dans les cellules suivantes le numéro qui a été attribué dans le `csv` à cette position du bras gauche à partir de votre premier numéro intuitif de frame de départ.

Ce processus se réalise donc entièrement à la main et peut être un peu fastidieux au début. Une idée pourrait être de reprendre le code de Manh Dan et Charles associé à cette partie là pour le rendre plus facile et peut-être plus rapide (cf fin du 2.1.1).

## 5 Conclusion

### Détection de la tête du nageur

Dans le cadre de la **détection de la tête du nageur**, notre travail s'est divisé en deux tâches principalement. La première d'enrichissement de la base de données générées l'année dernière : nous avons apporté **5868 images** ce qui représente une **augmentation de 186% de la taille de la base de données**. Nous avons codés deux algorithmes indépendants permettant de générer la base de données à partir de vidéos brutes avec leurs fichiers "Fil-Tendu" correspondants. Le premier permet de synchroniser la vidéo avec les données de vitesse et le deuxième génère les images ainsi que les labels associés aux images pointées.

La seconde partie de notre travail s'est concentrée sur l'amélioration du réseau de neurones codé l'an dernier. En entraînant le réseau de neurones uniquement sur la première base de données, nous avons obtenu une précision pour la **classification** de l'ordre de 82% sur l'ensemble de validation. Cependant, ce résultat est à nuancer dans la mesure où cette base de données contient des images trop similaires, ce qui fait que les ensembles d'entraînement et de validation se ressemblent trop : le modèle entraîné risque donc de mal se généraliser à d'autres types de courses. Avec la deuxième base de données, les performances sont similaires (à **environ 80%**) **sur l'ensemble de validation**. Les résultats sur l'ensemble d'entraînement sont quant à eux meilleurs (97% contre 85%), laissant penser que le phénomène de surapprentissage est plus présent avec cette base de données. Cela nous amène à penser que la base de données générée cette année est plus générale car elle est plus difficile à généraliser. En combinant les deux bases de données, les résultats sont meilleurs : les défauts des deux bases de données sont moins importants. Le phénomène de surapprentissage est beaucoup plus faible : l'écart entre l'entraînement et la validation passe de 17% d'écart à 5% d'écart.

Concernant le **problème de régression** à savoir quelles sont les coordonnées de la tête du nageur, les résultats obtenus sont très précis. Nous arrivons à obtenir une erreur de **régression moyenne de 5 pixels** sachant que la tête d'un nageur est d'environ 20 pixels pour les vidéos à notre disposition. Ainsi, pour obtenir de meilleurs résultats sur la détection de la tête du nageur dans sa globalité, il faudrait plus se concentrer sur le problème de classification où il existe une marge de progression.

### Calibration automatique sur vidéo

Concernant l'**autocalibration des vidéos**, notre travail s'est séparé en deux phases. La première consistait à appliquer des méthodes de traitement d'images déterministes afin d'extraire les lignes d'eau puis des points particuliers reconnaissables sur les images mises à notre disposition. Nous avons pu observer que si ces méthodes s'implémentaient sans difficulté, elles souffraient d'**un manque d'adaptabilité**. En effet, en fonction de l'angle de la vidéo, de l'éclairage ou de la présence de nageurs, les lignes repérées pouvaient sortir du cadre du bassin. Cela nous a poussé, sous les conseils de Monsieur Vincent Lepetit, à entamer la seconde phase : l'approche par le Deep Learning. Nous avons donc choisi le **réseau convolutionnel U-Net** puis créer une base de données adaptée. Ce réseau renvoie pour chaque entrée, une image de bassin, **8 cartes de probabilité de même résolution que l'entrée** et où chaque pixel de la i-ème carte a pour intensité la probabilité d'appartenir à la classe i. Pour créer la base de données, il a fallu choisir le type puis le nombre de classes a utilisé ainsi que les données d'entrée, effectuer une 'data augmentation' pour étoffer la base de données et enfin labelliser cette dernière. Les résultats obtenus sont encourageants mais soumis au **manque de diversité de la base créée**, faute de ressources à disposition. En effet, si le nombre de vidéos disponibles est suffisant pour le réseau de neurones visant à repérer les nageurs, dans notre cadre d'étude puisque les données sont les images tirées des vidéos à caméra fixe, beaucoup sont similaires. Il en va de même pour les labels utilisés. Il serait intéressant d'une part d'**agrandir cette base de données** et d'autre part de tenter une labellisation plus large où les classes seraient des zones de plusieurs pixels plutôt que des pixels seuls (la taille des zones pourraient être adaptée au marqueur de couleur qu'elle représente). De surcroît, nous avons rencontré des problèmes de mémoire vive liée à notre base de donnée. Il serait utile de disposer de plus de capacité en mémoire sur les processeurs utilisés mais aussi et surtout de diminuer la place mémoire prise par la base de données. Celle-ci étant créée de toute pièce, elle n'est pas optimisée en ce qui concerne son stockage et sa manipulation.

## Détection de la nage et des cycles de nage

Pour la **détection de nage et de cycles**, nous avons segmenté notre travail en plusieurs sous-parties, chacune ayant permis un avancement différent du projet. De plus, nous avons fait en sorte que chaque notebook soit facilement compréhensible et utilisable par tous. Ainsi, pour la détection de la nage, nous avons réalisé un algorithme permettant, à partir d'images obtenues par l'algorithme de l'équipe précédente, de **créer une base de données de test et de train d'images croppées autour du nageur**, étant donné que toute la longueur de la frame n'était pas utile pour notre modèle. Nous avons ensuite créé un **classifieur de nage** qui, à partir des frames du dataset, prédit pour chacune d'entre elles un chiffre entre 0 et 3, correspondant à une nage.

En ce qui concerne la **détection des cycles de bras**, notre travail se découpe en trois parties. Tout d'abord, à partir d'une base de données de vidéos, accompagnées de leurs données gyrométriques, nous avons **synchronisé les données gyrométriques avec les frames de ces vidéos**. Nous avons ensuite **labelisé ces données** de façon à ce que les frames sur lesquelles le nageur a le bras gauche levé aient un label de 1. Enfin, nous avons entraîné une **régession linéaire** afin d'obtenir les cycles de bras sur de nouvelles vidéos.

Les résultats que nous obtenons sont prometteurs. Nous avons en effet **une accuracy de plus de 70% en test sur le classifieur de nage**, et les images qui sont détectées par la régression linéaire comme ayant le plus haut score sont bien des images de nageurs le bras levé. Il est alors possible d'en **déduire les cycles de nage**. Néanmoins, nos résultats demandent encore à être améliorés, et nous avons pour cela quelques perspectives d'amélioration. Par exemple, il serait intéressant d'**affiner les modèles de réseaux de neurones et de régression linéaire** dans la détection de nage et de cycles pour obtenir des résultats plus précis, ou encore d'automatiser la détection de la première frame dans la synchronisation des données gyrométriques.

## Bilan

Ce travail de 6 mois nous a permis de redécouvrir le fait de travailler en équipe pour un projet de grande ampleur. Il a pour nous suscité beaucoup d'intérêt et nos tuteurs Rémi Carmignani et Vincent Lepetit y ont une grande part à jouer. Messieurs Carmignani, membre de SCIENCES<sup>2024</sup> et Lepetit, chercheur et docteur en vision par ordinateur, ont su transmettre leur savoir avec pédagogie et sympathie. Notre rencontre avec Monsieur Renaud Jester s'est notamment faite par l'intermédiaire de Monsieur Carmignani. Forts de leurs expériences respectives sur le sujet et sur les techniques de Machine Learning et vision par ordinateur, ils ont su nous guider tout au long du projet.

## Références

- [1] Fédération Française de Natation. Réglementation des bassins en natation. [https://www.ffnatation.fr/sites/default/files/ckeditor\\_files/02-natation\\_course.pdf](https://www.ffnatation.fr/sites/default/files/ckeditor_files/02-natation_course.pdf), 2021.
- [2] Toward Data Science. Unet — line by line explanation. <https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5>, 2019.
- [3] Théo Vincent Victoria Brami, Maxime Brisinger. Optimized swimmer tracking system based on a novel multirelated targets approach. *Rapport de Projet, Département IMI, Mai*, 2020.
- [4] James J. Little Jianhui Chen. Sports camera calibration via synthetic data. 2018.
- [5] Robert J. Woodha Ankur Gupta, James J. Little. Using line and ellipse features for rectification of broadcast hockey vide. 2018.
- [6] Fédération Française de Football. Réglementation des terrains et installations sportives. [https://www.sports.gouv.fr/IMG/pdf/\\_a\\_reglement\\_des\\_terrains\\_et\\_installations\\_sportives\\_vdef.pdf](https://www.sports.gouv.fr/IMG/pdf/_a_reglement_des_terrains_et_installations_sportives_vdef.pdf), 2021.
- [7] Renaud Jester. Lane detection on swimming competition videos. [https://github.com/renaudjester/detect\\_ligne\\_eau.git](https://github.com/renaudjester/detect_ligne_eau.git), 2021.
- [8] La transformée de hough. [https://fr.wikipedia.org/wiki/Transformee\\_de\\_Hough](https://fr.wikipedia.org/wiki/Transformee_de_Hough), 2020.
- [9] A. Alfalou A. Verney P. Hellard D. Benarab, T. Napoléon. Optimized swimmer tracking system based on a novel multirelated targets approach. *ISEN Brest, Département Vision, Brest Cedex 2, France*, 2016.
- [10] James J. Little Jianhui Chen. Sports camera calibration via synthetic data. *Laboratory for Computational Intelligence*, 2018.
- [11] Robert J. Woodham Ankur Gupta, James J. Little. Using line and ellipse features for rectification of broadcast hockey video. 2018.

## A Annexe : Conventions sur les fichiers de la base de données

### Conventions sur le fichier .txt

Le fichier .txt créé lors de la création de la base de données est de la forme suivante :

	Type	Unité	Nombre d'éléments sur la ligne
Nom du fichier vidéo	string	-	1
src	int	pixel	8
dst	int	pixel	8
matrice d'homographie	float	-	9
valeurs extrêmes	float	m	4

Comme indiqué dans la partie calibration de la vidéo, afin de corriger la perspective des images, il faut calculer une matrice d'homographie. Cette matrice est une matrice  $3 \times 3$ , ici vue comme un vecteur. Pour calculer cette matrice d'homographie, il faut les coordonnées de 4 points dans l'image originale ainsi que les coordonnées de ces 4 points dans l'image calibrée.

La ligne **src** du fichier .txt représente les coordonnées des 4 points dans l'image originale pour calculer la matrice d'homographie tandis que la ligne **dst** du fichier .txt comporte les coordonnées de ces mêmes points dans l'image calibrée.

Ci-dessous est présenté un exemple de fichier .txt généré lors de la création de la base de données.

```
DSC_7401.MOV
574.194957, 498.419999,105.808688, 895.908818,1567.475963,912.503011,1217.123474,505.752317
427.921374, 0.0, 427.921374, 648.0, 1141.123665, 648.0, 1141.123665, 0.0
-1.944488, -4.762085, 2718.881776, 0.075164, -6.590737, 3241.796484, 6.520316e-05, -0.005697, 1.0
-1.0, 0, 25.920833, 25
```

### Exemple du fichier .csv

Le fichier .csv généré lors de la création de la base de données est de la forme suivante :

	lane	frame	x_head	y_head	swimming_way
Type	int	int	float	float	int

Dans la colonne **lane** doit se trouver un entier qui représente le numéro du couloir d'eau du nageur. Ici, le premier couloir par le haut est représenté par l'entier 0.

Dans la colonne **frame** doit y figurer le numéro de l'image dans la vidéo.

Dans les colonnes **x\_head** et **y\_head**, doit figurer les coordonnées du nageur dans l'image originale. Ces dernières sont exprimées en pixels, avec le coin supérieur gauche de l'image après correction de la perspective qui prend les coordonnées (0, 0). Bien que cette convention ne soit plus utile grâce à la méthode de génération semi-automatique de la base de données, la convention pour une tête non détectée lors du pointage sont les coordonnées (-1, -1).

Finalement, la colonne **swimming\_way** est une colonne ajoutée dans l'optique où il faut pointer des vidéos dans lesquelles les nageurs nagent de la droite vers la gauche. Ici, la convention fixée est +1 si le nageur nage de la gauche vers la droite et -1 dans le cas contraire.

On trouve ci-dessous les premières lignes d'un fichier .csv généré lors de l'enrichissement de la base de données.

lane	frame	x_head	y_head	swimming_way
4	75	268.3147075425432	592.7719816301874	1
4	76	276.1135829564271	592.8608250315438	1
4	77	280.0130147937425	592.9052466653573	1
4	78	284.69232783326055	592.9585525670929	1
4	79	289.3716352379588	593.0118584046387	1
4	80	295.6107030122987	593.0829327548493	1
4	81	301.8497607692234	593.1540069909452	1
4	82	308.0888085087571	593.2250811129267	1
4	83	314.3278462309241	593.2961551207941	1

On remarque que c'est principalement la coordonnée horizontale (**x\_head**) de la tête du nageur qui évolue avec les frames. En effet, la coordonnées verticale de la tête du nageur est supposée constante sur l'image *top-down view*, ce qui explique le déplacement non nul des coordonnées **y\_head** car ces dernières sont sur l'image avant la correction de la perspective.

## B Annexe : Arborescence de *TrackingSwimmingENPC-master*

```

2021/ ..... NOUVEAU
└── detection_cycles/
    ├── data/
    │   ├── N0001/
    │   │   ├── 10x25m/ ..... vidéos brutes
    │   │   ├── calibration/ ..... fichiers .txt de l'annexe A
    │   │   ├── FilTendu/ ..... fichiers .csv du fil tendu
    │   │   ├── Gyro/ ..... fichiers .csv des données gyro
    │   │   ├── labels/ ..... fichiers .csv des labels pour le réseau de neurones
    │   │   ├── synchronisation/ ..... fichiers .csv issus de la synchronisation et images associées
    │   │   │   └── frames/ ..... images associées aux csv
    │   │   ├── N0003/..... idem que N0001
    │   │   ├── N0004/..... idem que N0001
    │   │   └── N0005/..... idem que N0001
    ├── src/
    │   └── filtendu.py
    ├── CYCLE_DETECTION.ipynb ..... notebook pour détecter les cycles de bras
    ├── Labellisation données gyro.ipynb ..... notebook pour créer la base de données
    └── Synchronisation gyro video.ipynb .... notebook pour synchroniser des données gyro avec les vidéos

detection_nage/
├── data/
│   └── cropped_images/ on enregistre les images coupées dans la nage correspondante, (ou le dossier tests)
│       ├── backstroke/
│       ├── breastroke/
│       ├── butterfly/
│       ├── crawl/
│       └── tests/
├── CREATE_CROPPED_DATASET.ipynb ..... notebook pour créer la base de données
└── SWIM_CLASSIFIER.ipynb ..... notebook du réseau de neurones pour détecter le type de nages

head_tracking/
├── data/ ..... contient les sous-dossiers des images pointées ainsi que les fichiers .csv et .txt
├── reports/ ..... contient les courbes d'entraînements et de validation
│   └── figures_results/
│       └── deep_model/
├── weight/ ..... contient les poids des modèles entraînés
│   └── magnifier/
│       └── deep_model/
├── lane_creation.ipynb ...
├── model.ipynb ..... notebook qui permet de faire tourner le modèle sur Colab
├── creation_dataset.ipynb...
└── initial_frame_selection.ipynb .. notebook pour synchroniser une vidéo et le fil tendu correspondant

autocalibration/
├── auto_image_selection.py ..... script pour obtenir une image à partir d'une vidéo
├── Data_augmentation.ipynb ..... notebook pour effectuer la data augmentation
├── labelling/ ..... modification du code de l'interface graphique pour créer la nouvelle base de données
│   ├── auto_edit_point.py
│   ├── data_base_autocalibration.py
│   └── autocalibration_points.py
└── data/
    ├── image_db/ ..... dossier contenant les images de la base de données
    └── label_db/ ..... dossier contenant les labels de la base de données
    └── Train.ipynb ..... notebook définissant et entraînant le modèle

data/ ..... inchangé par rapport à l'année dernière, avec de nouvelles vidéos pointées
logos/..... inchangé par rapport à l'année dernière
reports/ ..... inchangé par rapport à l'année dernière

```

src/ .....	inchangé par rapport à l'année dernière
add_data_flip.py .....	inchangé par rapport à l'année dernière
create_data_set.py .....	inchangé par rapport à l'année dernière
neural_tracking_tune_trade_off.py .....	inchangé par rapport à l'année dernière
observe_on_lane.py .....	inchangé par rapport à l'année dernière
observe_on_original.py .....	inchangé par rapport à l'année dernière
README.md .....	inchangé par rapport à l'année dernière
requirements.py .....	inchangé par rapport à l'année dernière
rough_tracking.py .....	inchangé par rapport à l'année dernière

## C Annexe : Utilisation des notebooks de détection des phases de nage

### Détection de la nage

#### Création de la base de données

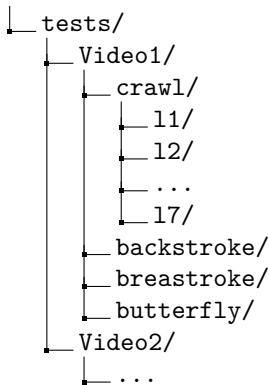
Se reporter à l'Annexe B pour l'arborescence des fichiers.

1. Dans le dossier `2021/detection_nage`, créer le dossier `data` et à l'intérieur le dossier `cropped_images`. Puis dans ce dernier dossier, créer quatre dossiers vides : `backstroke`, `breastroke`, `butterfly`, `crawl` et `tests`.
2. Dans le dossier `data/1_raw_videos`, enregistrer la vidéo à traiter. **Si celle-ci comporte plusieurs nages, découper celle-ci en plusieurs vidéos d'une nage unique.**
3. Exécuter le fichier `create_data_set.py`, en faisant attention à bien rentrer le nom de la vidéo et l'intervalle de pointage. Une application va alors se lancer. La calibration de la vidéo et l'enregistrement des images ligne d'eau par ligne d'eau va se faire. Puis il faudra pointer la tête du nageur sur chacune des frames (sauf si vous changez le code pour pointer toutes les 10 frames par exemple car une interpolation est ensuite possible dans le notebook). Les images sont enregistrées dans le dossier `data/2_intermediate_top_down_lanes/lanes`, et le csv du pointage dans le dossier `data/3_processed_positions`.
4. Ouvrir le Notebook `CREATE_CROPPED_DATASET.ipynb`, et rentrer dans la deuxième cellule les informations relatives à la nouvelle vidéo que vous souhaitez traiter.
5. Exécuter les cellules restantes.

### Réseaux de neurones

Pour l'entraînement du réseau de neurones, les images de `cropped_images` sont utilisés pour le train et le test.

1. Dans le dossier `tests`, la configuration est la suivante :



2. Ouvrir le notebook `SWIM_CLASSIFIER.ipynb` et exécuter toutes les cellules jusqu'à la partie "*Création des train, val et test sets (batchs)*".
3. Dans la première cellule de la partie "*Création des train, val et test sets (batchs)*", rentrer les différentes valeurs à modifier. D'abord, il faut choisir la valeur de `TAILLE_DATASET_PAR_NAGE`, c'est à dire le nombre de frames utilisées par dossier de nage. Par exemple, si on rentre 2000, il y aura 8000 images dans le dataset.
4. Ensuite, il faut choisir `PERCENT_DATASET`, qui est la part de données qui sera utilisée dans le train (exemple : 0.9 de train, 0.1 de validation).
5. Enfin, on choisit la vidéo que l'on veut tester, par exemple 'Video1', et la ligne d'eau de cette vidéo dont on veut classifier la nage. Si on veut regarder la nage du dossier 11, on met '`LANE = 1`'.
6. Pour finir, exécuter les cellules restantes.
7. Pour tester d'autres vidéos de test, il suffit de changer le nom de la vidéo et le numéro de ligne d'eau dans les paramètres à modifier, et exécuter les cellules '*creation du test et des loaders*' et '*resultats du test*'.

## Détection des cycles de bras

### Création de la base de données

1. Dans `2021/detection_cycles/data`, créer un dossier sous la forme suivante :

```
└── N0001/
    ├── 10x25m/
    ├── calibration/
    ├── FilTendu/
    ├── Gyro/
    ├── labels/
    └── synchronisation/
        └── frames/
```

`10x25m` contient l'ensemble des vidéos de passage du nageur `N0001`. `calibration`, `labels` et `frames` sont initialement des dossiers vides. Mettre dans `FilTendu` les fichiers `.csv` relatifs aux données de fil Tendu des vidéos, et dans `Gyro` ceux relatifs aux données gyrométriques.

N0001_corrige_1.csv
N0001_corrige_2.csv
N0001_corrige_3.csv
...

TABLE 2 – Exemples de fichiers contenus dans le dossier `Gyro`

2. Ouvrir le Notebook `Synchronisation gyro video.ipynb` et rentrer toutes les informations dans la deuxième cellule relatives au passage.
3. Exécuter les cellules qui suivent, en prenant garde à ne pas exécuter celle déclarant la variable `decalage`, qui n'est utile que dans certains cas et n'est à exécuter que si la figure générée quelques cellules plus loin des données gyroscopiques des deux bras (en bleu et orange), présente au début beaucoup plus que trois ondulations au départ. Visualiser la vidéo pour choisir et remplir le `tend`.
4. Passons à la calibration de la vidéo. Vous allez pouvoir sélectionner les quatres points pour calibrer la vidéo. Les données exemples à rentrer correspond à sélectionner le point en haut à gauche (ligne des 5 mètres la plus au haut fond), puis en haut à droite (ligne des 15 mètres la plus au haut fond), puis devant à droite (ligne des 15 mètres la plus proche) et enfin devant à gauche (ligne des 5 mètres la plus proche). Vous pouvez ensuite les remplir dans le notebook dans l'espace prévu à cet effet.
5. Remplir le numéro de la ligne où se situe le nageur ainsi que le nombre de lignes (par rapport au rectangle de calibration), et changer ainsi si besoin le tableau `swimmer_lane`. Exécuter les cellules suivantes jusqu'à la prochaine section.
6. Exécuter les cellules de la section "Récupération des données de position" (adapter le nom du fichier `datavel` au besoin)
7. Synchronisation manuelle : visualiser la vidéo et regarder le timing où le nageur pousse du mur. Comme vous connaissez le FPS, vous allez pouvoir en déduire un numéro de frame de départ à remplir dans la variable `frame_number_ref0`. Mettez la valeur de `test` à `True` et créer un dossier `test` (dossier temporaire, qui sera à supprimer entre chaque passage) dans le dossier `synchronisation`.
8. Sélectionner la position initiale de la tête.
9. Exécuter toutes les cellules jusqu'à la fin de la section. La dernière cellule va créer le `.csv` ainsi que les images associées (enregistrées dans le dossier `test`). Lors de la première exécution, décommenter tout ce qui touche à la visualisation `pyplot`, pour pouvoir afficher les images au fur et à mesure qu'elles s'enregistrent et ainsi mémoriser le numéro de frame de la première image du bras gauche levé.
10. Aller ensuite dans la section "Détection des cycles" et exécuter les deux premières cellules. Vérifier que la première détecte bien les pics et pour la deuxième, vous allez pouvoir vérifier (en prenant `idx0 = time2[peakind][0]`) si vous

avez vu juste sur votre frame de départ. Si le numéro de la frame correspond à celle que vous aviez mémorisée au point précédent, il ne vous reste plus à mettre la variable *test* sur *False* et qu'à réexécuter la cellule générant le .csv. Sinon, adapter votre frame de départ en fonction de la différence entre celle de l'algorithme et celle que vous aviez mémorisée, et réexécutez les cellules nécessaires puis revérifier (avec les autres pics en changeant l'indice de *idx* par exemple).

11. Une fois la valeur de *test* mise à *False* et les bonnes cellules réexécutées, le fichier .csv sera enregistré dans le dossier **synchronisation**, et les images dans le dossier **frames**. Veuillez bien à supprimer le dossier **test** (et à en recréer un vide), et à mettre la variable *test* sur *True*.
12. Ouvrir maintenant le Notebook **Labellisation données gyro.ipynb**, et remplir la deuxième cellule en fonction des caractéristiques de la vidéo. Bien faire attention à avoir créer le dossier **labels**.
13. Exécuter le reste du Notebook. Le dossier **labels** contient un fichier .csv pour chaque vidéo de passage avec pour colonnes *time*, *frame* et *label*.

time	frame	label
0.0333670000000001	40	0.40063018936603273
0.0667349999999995	41	0.35712768291846914
0.1001180000000001	42	0.35713017935268637
0.1334719999999993	43	0.3483037729773528
0.1668449999999985	44	0.3121452595835765

TABLE 3 – Exemple de fichier .csv dans le dossier **labels**

### Régression linéaire - prédition des cycles

1. Ouvrir le notebook **CYCLE\_DETECTION.ipynb**
2. Dans la partie *Création des train, val et test sets (batchs)*, modifier les deux paramètres **TAILLE\_DATASET** et **TAILLE\_TEST** en fonction des valeurs désirées. Ici, à la différence du classifieur de nage, la taille du dataset est la taille exacte du dataset, et non la taille par nage. La fonction **create\_dataset** va alors parcourir toutes les images de tous les fichiers de **detection\_cycles**. Il va choisir les  $n = \text{taille\_dataset}$  premiers pour le train/val (en les mélangeant), et les  $m = \text{taille\_test}$  suivants pour le test.
3. Exécuter toutes les cellules restantes.