

Master of Science HES-SO in Engineering  
Av. de Provence 6  
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation : Data Science (DS)

# Machine learning for analyzing egocentric videos of patients during rehabilitation

Done by

**Sam Corpataux**

Under the direction of

Prof. A. Perez Uribe and PhD Student Y. Izadmehr

Mandated by HES-SO Master and CHUV

Lausanne, HES-SO//Master, 2023

## Table of versions

Version	Description
1.0	Initial version
1.1	Adapting the document structure to match the required format
1.2	Correction after first proofreading

## Executive summary

This in-depth project (PA) report was produced by Sam Corpataux, a 1st year Master's student at the HES-SO in the Data Science stream.

The project was commissioned by the University Hospital of Vaud (CHUV) and the University of Applied Sciences and Arts of Western Switzerland (HES-SO).

The brief is to explore the use of computer vision and/or machine learning algorithms to track the gestures of a user and the objects present in the scene to identify what the user is doing and gain insights into his/her intentions and functional capabilities.

In more concrete terms, the goal is to develop deep learning models that can automatically recognize and segment different sub-movements of a hand movement. These algorithms could be then used in various applications such as gesture recognition, human-robot interaction, and rehabilitation.

The types of sub-movement are as follows: reaching, grasping, transporting, manipulating, and placing objects.

MediaPipe and YOLOv7 models were used to detect and track the hand and objects in the scene of an egocentric video. The outputs of these models were then used to create a dataset used to train classification models.

The results of the classification models are not very satisfactory. The causes of that not-so-good results are discussed at the end of this report. That being said, those results could probably be improved by spending more time on the model's creation and training.

The project was carried out in collaboration with Ph.D. student Yasaman Izadmehr, who is working on a larger project on the same topic and was supervised by Prof. Andres Perez Uribe.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Objectives of the project . . . . .	1
1.2.1	State of the art in motion detection and existing algorithms . . . . .	1
1.2.2	Video annotation . . . . .	2
1.2.2.1	Video specifications . . . . .	2
1.2.2.2	Labelisation . . . . .	2
1.2.3	Creating datasets . . . . .	2
1.2.4	Model creation . . . . .	3
<b>2</b>	<b>Analysis</b>	<b>4</b>
2.1	Analysis of user needs . . . . .	4
2.2	Technological analysis . . . . .	4
2.2.1	YOLO . . . . .	4
2.2.2	MediaPipe . . . . .	5
2.2.3	Frames resolution . . . . .	5
2.2.3.1	Resizing techniques . . . . .	6
2.2.3.2	Results combination . . . . .	6
<b>3</b>	<b>Video labelling</b>	<b>7</b>
3.1	Distinctive classes . . . . .	7
3.2	Video annotation . . . . .	8
3.2.1	ROI labels . . . . .	8
3.2.2	Scene labels . . . . .	8
3.3	Label storage . . . . .	9
3.4	Frame extraction . . . . .	10
<b>4</b>	<b>Model design</b>	<b>11</b>
4.1	Video frame model . . . . .	11
4.2	Model without video frames . . . . .	11
<b>5</b>	<b>Datasets creation</b>	<b>12</b>
5.1	First dataset creation process . . . . .	12
5.2	Dataset splitting . . . . .	12
5.3	Dataset balancing . . . . .	13
5.4	Second dataset creation process . . . . .	15
5.4.1	MediaPipe output . . . . .	15
5.4.1.1	MediaPipe output - landmarks . . . . .	15
5.4.1.2	MediaPipe output - hands center . . . . .	16
5.4.2	YOLO output . . . . .	16
5.4.3	Phantom object . . . . .	18
5.4.4	Distances between the hands landmarks and the object . . . . .	19
5.4.5	Final data structure . . . . .	20

<b>6</b>	<b>Implementation</b>	<b>22</b>
6.1	MediaPipe model . . . . .	22
6.1.1	Model input . . . . .	22
6.1.2	Hands center detection . . . . .	22
6.1.3	Hands center results merging . . . . .	23
6.1.4	Hands landmarks detection . . . . .	24
6.1.5	Hands landmarks results merging . . . . .	24
6.2	YOLO model . . . . .	24
6.2.1	Model input . . . . .	24
6.2.2	Running the model . . . . .	24
6.2.3	Cleaning model results . . . . .	26
6.3	Submovements classification model . . . . .	26
6.3.1	Model architecture . . . . .	26
6.3.2	Model results . . . . .	27
6.4	Binary classification model . . . . .	29
6.4.1	Dataset modification . . . . .	29
6.4.2	Model architecture . . . . .	30
6.4.3	Model results . . . . .	30
<b>7</b>	<b>Problems encountered</b>	<b>33</b>
7.1	Missing frames . . . . .	33
7.2	Missing computing power . . . . .	33
7.3	Unbalanced datasets . . . . .	33
<b>8</b>	<b>Conclusion</b>	<b>35</b>
8.1	Time management . . . . .	35
8.2	Futur work . . . . .	36
8.3	Personal conclusion . . . . .	36
8.4	Acknowledgement . . . . .	37
8.5	Declaration of honor . . . . .	37
	<b>Bibliography</b>	<b>38</b>

# List of Figures

3.1	MathLab Video Labeler tool with scene labels . . . . .	9
3.2	MathLab workspace view with the "gTruth" variable . . . . .	10
5.1	Initial classes distribution . . . . .	13
5.2	Final classes distribution . . . . .	14
5.3	Initial vs final classes distribution . . . . .	14
5.4	MediaPipe landmarks before cleaning . . . . .	15
5.5	MediaPipe hands center before cleaning . . . . .	16
5.6	YOLOv7 objects before cleaning . . . . .	17
5.7	The 4 quadrants of the 300x300 image . . . . .	18
5.8	Example of the structure of the distances between the hands and the object . . .	20
5.9	Final data structure . . . . .	21
6.1	MediaPipe hands detection results . . . . .	23
6.2	Accuracy and loss curves of the submovements classification model . . . . .	28
6.3	Confusion matrix of the submovements classification model . . . . .	28
6.4	Accuracy and loss curves of the submovements classification model with more layers	29
6.5	Accuracy and loss curves of the binary classification model . . . . .	30
6.6	Confusion matrix of the binary classification model . . . . .	31
6.7	Accuracy and loss curves of the binary classification model with zeros values in- stead of the phantom . . . . .	31
8.1	Simplified time repartition . . . . .	35

# 1 Introduction

## 1.1 Context

Vision-based human activity recognition can provide rich contextual information, but it has typically been computationally costly and raises certain privacy issues. Yet, the employment of egocentric cameras in the context of patient rehabilitation opens the door to numerous applications while protecting people's privacy. In addition, convolutional neural networks (CNNs), a recent development in machine and deep learning algorithms, have shown promising results in achieving high accuracy in hand segmentation and tracking. These algorithms train the model to identify and categorize various hand movements into sub-movements using sizable annotated datasets.

In this project, we will investigate how computer vision and/or machine learning algorithms can be used to track a user's gestures and the objects in the scene to identify what the user is doing.

## 1.2 Objectives of the project

The main objective of this project is to develop algorithms that can automatically recognize and segment different sub-movements of a hand movement. Those algorithms could be then used in various applications such as gesture recognition, human-robot interaction, and rehabilitation. The types of sub-movement are as follows: reaching, grasping, transporting, manipulating, and placing objects.

This work is divided into sub-objectives :

- Learn about the state of the art in motion detection and discover the existing algorithms
- Video annotation
- Creating datasets
- Model creation

These sub-objectives are explained in detail below.

### 1.2.1 State of the art in motion detection and existing algorithms

This work is part of a larger project, on which the PhD student Izadmehr Yasaman is working. Before diving into the development of new algorithms, I need to know what already exists and whether I can use these tools in this project. To guide me and as a prerequisite, I have been provided with examples of implementations of the YOLO (see chap. 2.2.1) and MediaPipe (see chap. 2.2.2) algorithms.

At the moment, it seems that the state of the art to achieve video-based segmentation of hand movements into sub-movements is by using a combination of CNNs and recurrent neural networks (RNNs). CNNs are used for hand detection, segmentation, and feature extraction, while RNNs are used to model the temporal dependencies between the sub-movements.

### 1.2.2 Video annotation

To develop an algorithm, you need data to train it. Since the domain of this project is very specific (using egocentric cameras to estimate the rehabilitation rate of patients according to their movements), I do not have access to prefabricated datasets, so I have to build it myself.

To do this, I was provided with a video, filmed in the laboratory, in which a fake patient is seen performing various tasks to represent different types of sub-movements.

#### 1.2.2.1 Video specifications

The video is about 8 minutes long and is filmed in 60 frames per second, giving us a total number of precisely 28'854 frames. During the label extraction process, 28 frames disappeared for some unknown reason, so the total number of frames used to create the datasets is 28,826. A whole chapter is concerned with the detailed description of this problem, at the end of the document (see chapt. ??).

The video was shot with an egocentric camera, placed on the patient's shoulder. In other words, the video stream represents what the person sees herself (P.O.V). This way, both the patient's hands and arms are almost constantly in the field of vision and it is easier to determine what kind of movement she is making.

The video dimension is 1920x1080 pixels, which corresponds to an HD resolution. However, as explained later in this document (see chapt. 2.2.3), the frames exported from the video have a dimension of 640x360 pixels, which is a lower resolution.

There are multiple reasons for this. First, the higher the resolution, the more data you have to process, which can slow down the algorithm. And secondly, it has been shown that higher resolution does not necessarily lead to better results (see chapt. 2.2.3).

#### 1.2.2.2 Labelisation

There are many different ways to label a video. In this project, I was encouraged to use the MathLab Video Labeler tool [1]. This tool allows you to use pre-created algorithms to label videos. This is very useful since it makes you save a lot of time.

Since the objective of this project is to segment hand movements into sub-movements, the easiest and the smartest way to label the video is to use label entire frames and not just a part of it.

Our goal is to detect those five different sub-movements :

- Reaching
- Grasping
- Transporting
- Manipulate
- Placing

### 1.2.3 Creating datasets

As mentioned in the following chapter (see chapt. 5), we need to create datasets to train our model. To do this, we need to extract the frames from the video and their corresponding labels.

As perfectly well explained in this article [2], in Machine Learning projects, it is important to split the data into three different sets :

- Training set
- Validation set
- Test set

The training set is used to train the model. The validation set is used to evaluate the model during training and to tune the hyperparameters. The test set is used to evaluate the final model after the training is completed.

The distribution of the data between those three sets must be done carefully. If the training set is too small, the model will not be able to learn properly. If the validation set is too small, the model will not be able to generalize well. And if the test set is too small, the evaluation of the model will not be accurate. There is not a single rule to follow to split the data, it depends a lot on the size of the dataset. More information about how the data have been split in this project can be found in chapter 5.

#### 1.2.4 Model creation

Once the datasets are created, we can start the model creation. As mentioned in chapter 1.2.1, the state of the art to achieve video-based segmentation of hand movements into sub-movements is by using a combination of CNNs and RNNs. In this project, we will use the combination of YOLO and MediaPipe to achieve this goal.

The two algorithms' results will be used to train a third model that will be able to segment the hand movements into sub-movements. The architecture of this model is explained in detail in chapter 6.3.

An example of YOLO and MediaPipe algorithm implementation has been provided to me by Ph.D. student Izadmehr Yasaman. This is only an example, it does not suit our needs perfectly, but it is a good starting point.



## 2 Analysis

In this chapter, we will address two axes of analysis, namely: the analysis of user needs and the technical analysis.

The fruit of this analysis work allowed me to choose the best algorithms already existing in the framework of hand and object detection to improve the results of my submovement classification model.

### 2.1 Analysis of user needs

The user needs analysis phase having already been more or less done before this work, I did not spend much time on this task.

The initial objective of the large project in which this work is included is to produce a model capable of assisting doctors in the motor rehabilitation of patients. At present, patients who have lost the ability to use one of their limbs have to undergo physiotherapy and other medical treatment to gradually regain the use of their injured limb.

The evaluation of the patient's progress during these rehabilitation sessions is, until now, mainly done subjectively by the doctors. By "subjective", I mean that there is no precise way of measuring the evolution of this rehabilitation with figures.

The objective of this project would therefore be to create a tool capable of automatically detecting the extent to which a patient is progressing in his rehabilitation.

To create such a model, capable of assessing the extent to which a patient regains normal use of one of these upper limbs, we first need to create a model that can distinguish between the different ways in which a human can move their arm.

This is where my work comes in, as described in the chapter (see chapt. 1.2.2.2) , I aim to be able to classify the 10 types of submovement that our arms can perform.

I have not had the opportunity to meet patients or to go into more detail about the concrete needs of users, as the current context is more than enough for me to do my work.

### 2.2 Technological analysis

Now that the user requirements have been clearly defined, we turn to the analysis of existing technologies. The choice of these technologies is essential to maximize the performance of the final submovement classification model.

#### 2.2.1 YOLO

YOLO (You Only Look Once) [3] is a highly accurate and efficient object detection algorithm that is based on a Convolutional Neural Network (CNN). It is particularly well-suited for hand detection tasks, as CNNs are excellent at learning complex patterns and spatial relationships in images, which is essential for detecting and localizing hands in different poses and lighting conditions.

YOLO uses a single CNN to simultaneously predict object classes and bounding boxes for multiple objects in an image or video frame, making it extremely fast and efficient. It achieves this by dividing the input image into a grid and predicting the object and its bounding box for each grid cell, as well as the class probabilities for each object.

With its high accuracy, real-time performance, and suitability for hands or objects detection, YOLO is an excellent choice for a wide range of applications, such as sign language recognition, human-computer interaction or hand movements detection.

### 2.2.2 MediaPipe

MediaPipe [4] is a very powerful computer vision framework that was developed by Google to create real-time multimedia applications that work seamlessly across different platforms. This open-source platform offers pre-built components, including face detection, hand tracking, object detection, and pose estimation, that can be easily integrated into applications.

One of the key advantages of MediaPipe is its use of both traditional computer vision techniques and state-of-the-art machine learning algorithms, such as CNNs and RNNs. These algorithms enable MediaPipe to perform complex tasks, such as object detection and hand tracking, with high accuracy and in real time.

For example, the hand tracking component in MediaPipe combines a CNN-based hand detector with an RNN-based hand pose estimator to detect and track the position and orientation of hands in real-time video streams. Similarly, the object detection component in MediaPipe uses a CNN-based object detector to detect and localize objects in images or video frames.

Overall, MediaPipe is an essential tool for anyone looking to build a real-time computer vision applications with a high degree of accuracy and flexibility.

### 2.2.3 Frames resolution

The frames resolution is a very important parameter in the context of this work. Indeed, the higher the resolution of the frames, the more information the models will have to work with and therefore the longer the training time will be. It is not just a question of training time, but also a question of computing power and storage space.

Taking my machine specifications in consideration, I first decided to work with frames of resolution 640x360. I tried to maximise the resolution of the frames as much as possible while taking into account the not very powerful computing power of my machine. But even with this "quite low" resolution, I quickly reached my pc limits. This problem is explained in more detail in the chapter (see chapt. 7.2). To summup, Mr. Perez gave me access to HEIG-VD's servers, which allowed me to work more efficiently.

However, despite the fact that I had access to a more powerful machine, I learned that higher frames resolutions were not necessarily better. Indeed, Ms. Izadmehr explained to me that she had conducted studies [5] on the impact of frames resolution on the performance of YOLO and MediaPipe models. She explained me that there is some specific resolutions that seems to increase the performance of the models. Those resolutions are not the same for YOLO and MediaPipe.

She told me that, according to her studies, in order to maximise the results of MediaPipe, we have to use a combination of 300x300 resolutions frames, while for YOLO, the best results are obtained with 640x640 resolutions frames.

### 2.2.3.1 Resizing techniques

There are several ways to resize images to get the desired resolution. However, if not done properly, resizing can harm the quality of the images. That is why it is important to choose the right resizing technique.

In my case, I have to resize the original frames of resolution 1920x1080 to 300x300 and 640x640.

To resize the frames, I used the open-source software library FFmpeg [6]. FFmpeg is an open-source software library renowned for its extensive multimedia capabilities. While its primary focus is handling audio and video files, it also offers two useful options for image manipulation: padding and cropping. With FFmpeg, we can leverage its command-line interface and powerful video filtering options to perform these operations on images. Padding allows us to add borders to an image, adjusting its size or aspect ratio, while cropping enables us to selectively extract a specific region of interest from an image.

In our case, we will use these two options to resize the frames. Since the original shape of the frames are not square (1920x1080), we will first add borders to the frames with the padding option. Then, we will crop the frames to the desired resolution.

Here is the command I used to add borders to the frames :

```
for j in *.png; do ffmpeg -i "$j" -qscale 0 -vf pad="max(iw\,ih):  
ow:(ow-iw)/2:(oh-ih)/2" -q:v 1 "path_to_save/${j%.png}.png" ;  
done
```

Basically, this command will loop through all the frames in the current directory and add borders to them. The options -qscale 0 and -q:v 1 are used to ensure that the quality of the frames is not degraded. The option -vf pad="max(iw,ih):ow:(ow-iw)/2:(oh-ih)/2" is used to specify the padding options. The first parameter max(iw,ih) is used to specify the width and height of the output frames. The second parameter ow is used to specify the width of the output frames. The third parameter (ow-iw)/2 is used to specify the left padding. The fourth parameter (oh-ih)/2 is used to specify the top padding.

And here is the command I used to crop the frames :

```
for j in *.png; do ffmpeg -i "$j" -vf scale=300:300 "path_to_save  
/${j%.png}.png" ; done
```

Similarly, as in the previous one, this command will loop through all the frames in the current directory and crop them to the desired resolution. The option -vf scale=300:300 is used to specify the cropping options. In this particular case, we want to crop the frames to the resolution 300x300.

### 2.2.3.2 Results combination

Again, according to Ph.D. student Yasaman Izadmehr, using frames of a specific resolution for each model to maximize their performance is good. But if we want to get the best results, we have to run models multiple times with different frame resolutions and then combine the results.

This is a crucial point that cannot be overlooked. An example of the importance of this point is demonstrated later in the document (see chapt. 6.1).

## 3 Video labelling

This chapter describes the video labelling process. It explains how the videos were labelled and how the labels were stored. It also describes the different tools used to label the videos.

### 3.1 Distinctive classes

As explained in the introduction (see chapt. 1.2.2.2), the goal of this project is to detect sub-movements in videos of hand movements.

It is not trivial to precisely define what a sub-movement is. In this project, we define a sub-movement as a movement that is part of a bigger movement. For example, when a person is cutting a vegetable, we can distinguish five types of sub-movements.

**Reaching :** The person is reaching for the knife. The sub-movement starts when the person is not holding the knife and ends when the person is about to grab the knife.

**Grasping :** The person is grabbing the knife. The sub-movement starts when the person is about to grab the knife and ends when the person is holding the knife.

**Transporting :** The person is transporting the knife to the vegetable (let's assume the knife was on another table and the person had to move it to the table where the vegetable is). The sub-movement starts when the person is holding the knife and ends when the person is about to cut the vegetable.

**Manipulate :** The person is cutting the vegetable. The sub-movement starts when the person is about to cut the vegetable and ends when the person is done cutting the vegetable.

**Placing :** The person is placing the knife back on the table. The sub-movement starts when the person is about to place the knife on the table and ends when the person is not holding the knife anymore.

Since humans have two hands, we need to label 10 different classes. The classes are as follows :

- Reaching Left
- Grasping Left
- Transporting Left
- Manipulate Left
- Placing Left
- Reaching Right
- Grasping Right
- Transporting Right
- Manipulate Right
- Placing Right

It is important to note that a single frame can contain zero, one, or two labels. For example, when the patient is cutting a vegetable, she is using the left hand to hold the vegetable and the right hand to hold the knife and cut the vegetable. In this case, the frame will be labeled with the "Grasping Left" and "Manipulation Right" labels.

## 3.2 Video annotation

As mentioned before, the video annotation was done using the MathLab Video Labeler tool [1].

To launch the tool, we simply need to run the following command in the MathLab console : `videoLabeler`. Once launched, the tool is very easy to use. We simply need to load the video and start labelling the frames. To do this, we start by importing the video and creating the labels.

### 3.2.1 ROI labels

Once the video has been imported and the labels created, we can start labelling the frames. There are two main ways to label the frames. The first one is to use "ROI labels". ROI stands for Region Of Interest. This is useful when we want to label a very specific part of the frame. For example if you want to detect all cars in a video, you can use ROI labels to specifically add a bounding box around each car.

Labelling frames is very time consuming, especially when you use ROI labels, because you need to add a bounding box for each label in each frame. Hopefully, the MathLab Video Labeler tool has a many algorithms that can speed up the process, by automatically adding some bounding boxes for you.

For example, the "AFC people detector" algorithm can automatically add a bounding box around each person in the frame. Or the "Temporal Interpolator" algorithm that can estimate the ROIs in intermediate frames using interpolation of rectangle or projected cuboids ROIs in key frames. The tool proposes many other algorithms that can help you label the frames faster. You can even create your own algorithm if you want. However, in this project, we didn't use any of these algorithms because ROI labels are not very appropriate in our case.

### 3.2.2 Scene labels

The second way to label the frames is to use "scene labels". This is useful when you don't have a specific part of the frame to label. Since it is quite hard to delimitate a specific part of the frame when labelling hand and arm movements, scene labels are more appropriate for this project.

For example, if you want to label a video of a person cutting a vegetable, you can use scene labels to label the frames with the labels described in the previous section.

Even though we don't have to add a bounding box for each label with scene labels, it is still very time consuming to label an entire video. In contrast with ROI labels, we don't need to have fancy algorithms to help us label the frames faster. With scene labels, we can simply choose between adding (or removing) a label to the current frame, or to a time range (multiple frames at once).

When put it like that, we might think that it is a job that is quickly done. However, to label a video of 8 minutes long, it took me around 12 hours. This is partly due to the fact that the video is filmed in 60 fps, which means that there are more than 28'800 frames to label. And even if we can use the "Time Range" option to label multiple frames at once, it still takes a lot of time and concentration to be sure not to miss any label.

Here is a screenshot of the MathLab Video Labeler tool with scene labels :

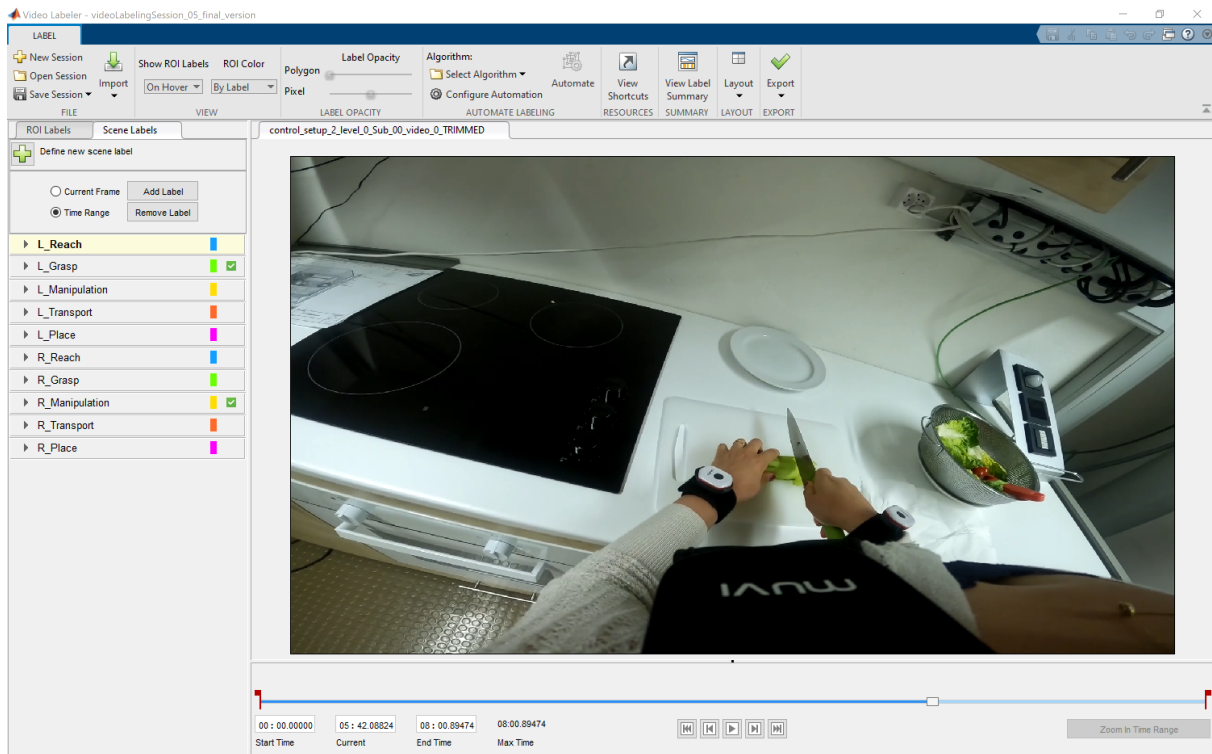


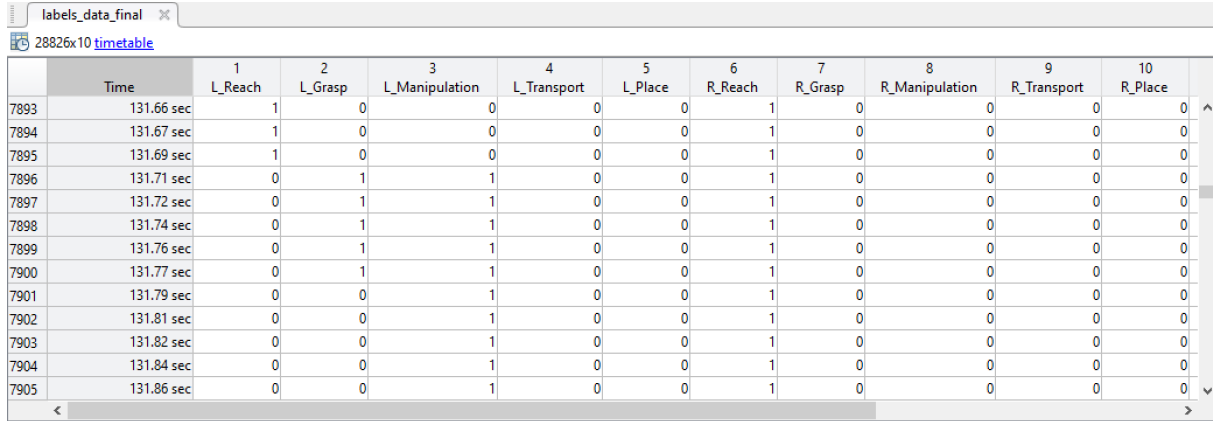
Figure 3.1: MathLab Video Labeler tool with scene labels

As we can see in the image, the patient is cutting a vegetable. She is holding/grasping a cucumber in her left hand and a knife in her right hand. As she is cutting the vegetable, we can consider that the right hand is not only grasping the knife, but also manipulating it. Therefore, the labels added for this frame are Left Grasping and Right Manipulating (this is shown in the image by the two little green validation rectangle, on the left of the screenshot).

### 3.3 Label storage

Once the video has been labelled, we need to store the labels in a file. To do this, we can simply export the labels in a .mat file, by clicking on the "Export" and then "Labels to file" buttons. Another way to store the labels is to export them in a .csv file. This is what we did in this project. To do this, we need to click on the "Export" and then "Labels to workspace" buttons. We then need to choose a name for the variable, we usually call it "gTruth" (ground truth). Once done, we can access the variable in the MathLab workspace and save it in a .csv file. This way to do is also very useful to have a quick look at the labels and to make sure that everything is correct.

Here is an example of the MathLab workspace view with the "gTruth" variable :



	Time	1 L_Reach	2 L_Grasp	3 L_Manipulation	4 L_Transport	5 L_Place	6 R_Reach	7 R_Grasp	8 R_Manipulation	9 R_Transport	10 R_Place
7893	131.66 sec	1	0	0	0	0	1	0	0	0	0
7894	131.67 sec	1	0	0	0	0	1	0	0	0	0
7895	131.69 sec	1	0	0	0	0	1	0	0	0	0
7896	131.71 sec	0	1	1	0	0	1	0	0	0	0
7897	131.72 sec	0	1	1	0	0	1	0	0	0	0
7898	131.74 sec	0	1	1	0	0	1	0	0	0	0
7899	131.76 sec	0	1	1	0	0	1	0	0	0	0
7900	131.77 sec	0	1	1	0	0	1	0	0	0	0
7901	131.79 sec	0	0	1	0	0	1	0	0	0	0
7902	131.81 sec	0	0	1	0	0	1	0	0	0	0
7903	131.82 sec	0	0	1	0	0	1	0	0	0	0
7904	131.84 sec	0	0	1	0	0	1	0	0	0	0
7905	131.86 sec	0	0	1	0	0	1	0	0	0	0

Figure 3.2: MathLab workspace view with the "gTruth" variable

As we can see in the image, the "gTruth" variable is a table with 11 columns. The first column contains the time of the frame in seconds (we can know the exact frame number by looking at the ID on the left). The other columns contain the 10 labels described in the previous section. The value of each cell is either 0 or 1. 0 means that the label is not present in the frame, and 1 means that the label is present in the frame.

### 3.4 Frame extraction

Store the labels in a file is essential to train a model, since we need to know which label is present in each frame. However, we also need to extract the frames from the video.

To do this, I used the open-source software FFmpeg [6]. More details and examples of FFmpeg commands can be found in the "Resizing techniques" section of the "Analysis" chapter (see 2.2.3.1).

The FFmpeg command used to extract the frames from the video is the following :

```
ffmpeg -i [video path] -r 60 -vf scale=640:-1 'frames/%06d.png'
```

The command is quite simple. We first need to specify the path to the video. We then need to specify the frame rate of the video. In our case, the video is filmed in 60 fps. We then need to specify the path to the folder where we want to store the frames. In this example, we also use the "-vf scale=640:-1" option. This option is used to resize the frames to 640x360 pixels, the only purpose of this is to reduce the size of the dataset, in order to be able to store it on my personal computer. But this is not necessary, we can also store the frames in their original size. And finally, we need to specify the name of the frames. The name of the frames is composed of 6 digits, starting from 000001.png.

The frame name is composed of 6 digits because the video is 8 minutes long, which means that there are around 28'800 frames. Actually, there are exactly 28'854 frames in the video but as explain in the "problem encountered" chapter (see 7.1), the ground truth labels are only available only contains 28'826 frames labels. This is why all the created dataset have "only" 28'826 frames, to be sure that the labels and the frames match.

## 4 Model design

Throughout the project, multiple model designs were imagined. In this section, the two main ideas will be presented. The first one is a model that uses the actual video frames and results of other algorithms as input. The second one is a model that uses only the output of the MediaPipe and YOLO models as input.

### 4.1 Video frame model

The first model design is a model that uses the actual video frames and the output of the MediaPipe and YOLO models as input. This is probably the most intuitive model design. The idea is to create a model that will learn to predict the labels based on the actual video frames and will have the assistance of the MediaPipe and YOLO models to help it learn. This model design was supposed to be the only one used in this project. However, since time was limited, we decided to simplify this model.

### 4.2 Model without video frames

The second model design is a model that uses only the output of the MediaPipe and YOLO models as input. The decision to get rid of the video frames was made because of the time constraint. The idea is to create a model that will learn to predict the labels based only on the output of the MediaPipe and YOLO models. This model design is the one that was used in this project.

When I say that the model does not use the video frames, I mean that the model does not use the actual video frames as input. However, the model still uses the video frames in a way. Indeed, the MediaPipe and YOLO models use the same video frames as input.

More information about the model types I used are specified in the Model types section.



## 5 Datasets creation

The creation of a dataset is of paramount importance when it comes to training a machine learning model. A dataset serves as the foundation on which the model learns and generalizes patterns, enabling it to make accurate predictions or, in our case, classifications. By curating a well-structured dataset that accurately represents the problem domain, we provide the model with diverse and relevant examples to learn from. This allows the model to learn the underlying patterns and relationships between the input data and the target labels.

Additionally, a sufficiently large dataset allows for better model performance by capturing a broader range of patterns and reducing overfitting. The 28'000 images dataset used in this project can seem like a lot of data, but after filtering out the images that are not relevant to the problem, and after splitting the dataset into training, validation and test sets, the dataset is actually quite small.

### 5.1 First dataset creation process

The first dataset I created was at the beginning of the project, just after I had finished to labelize the video. At that time, I thought that I was going to use the video frames as input for the model, as explained in section 4.1.

So, after extracting the frames from the video with FFmpeg (see section 3.4), I used them to create a first array. Then, I used the csv file containing the ground truth labels to create a second array.

The shapes of the two arrays are the following :

```
Video frame array shape: (28826, 300, 300, 3)
Ground truth array shape: (28826, 10)
```

The first array contains 28'826 images, each of them having a shape of 300x300 pixels and 3 channels (RGB). The second array contains 28'826 labels, each of them having a shape of 10 values. The 10 values represent the 10 sub-movements classes, described in section 3.1.

Finally, I merged the two arrays to create the final dataframe, containing the video frames and the ground truth labels. The shape of the final dataframe is the following :

```
Global set : (28826, 270010)
```

As we can see, the merging operation has flattened the images, so that each image is represented by a single row. The dataframe contains 28'826 rows and 270'010 columns. We can easily calculate the number of columns by multiplying the number of pixels of an image and by adding the number of labels. In our case, the number of pixels is  $300 \times 300 \times 3 = 270'000$  and the number of labels is 10. So, the number of columns is  $270'000 + 10 = 270'010$ .

### 5.2 Dataset splitting

In machine learning, the training set, validation set, and test set play crucial roles in developing and evaluating models. Each of them has a specific purpose and is essential to the machine learning workflow.

The training set is used to train the model by exposing it to labeled examples. It helps the model learn patterns and relationships within the data.

The validation set, on the other hand, is employed to fine-tune the model's hyperparameters and assess its performance during training. By evaluating the model on the validation set, we can make adjustments to improve its generalization capabilities and prevent overfitting.

Finally, the test set serves as an unbiased measure of the model's performance. It consists of unseen data that the model has not encountered during training or validation. Evaluating the model on the test set provides an estimation of its real-world performance.

In our case, I have arbitrarily decided to split the dataset as follows :

- Training set : 70%
- Validation set : 15%
- Test set : 15%

This may not be the optimal way to split the dataset, I simply decided to split it this way because I wanted to have a rough idea of the model's classes distribution. It is possible that this distribution may change when the final dataset is created.

### 5.3 Dataset balancing

After splitting the dataset, I noticed that the classes distribution was not balanced. Indeed, some classes were over-represented while others were under-represented.

Here is the initial classes distribution :

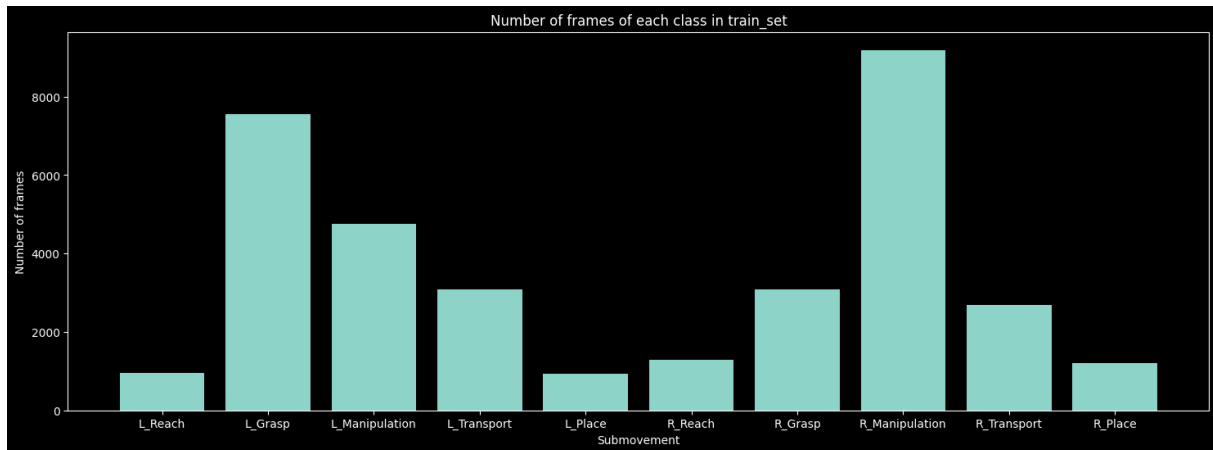


Figure 5.1: Initial classes distribution

This is no surprise, since there is some sub-movements that are more frequent than others in the video. For example, the sub-movements "Left Grasping" and "Right Manipulation" are more frequent than the others. It happens because in this particular video, the subject spend a lot of time washing or cutting vegetables and since she is right-handed, she uses her right hand to manipulate the knife and her left hand to grasps the vegetables.

However, this is not a good thing for the model, because it will tend to overfit the over-represented classes and underfit the under-represented classes. So, I decided to balance the dataset by selecting the same amount of images for each class. The most under-represented class is the "Left

"Placement" class, with only 924 frames. So, I decided to restrict the number of frames for each class to 920. Here is the final classes distribution :



Figure 5.2: Final classes distribution

As we can see, the classes distribution is more balanced now. Nevertheless, the dataset is still not perfectly balanced. We can explain this simply by the fact that a single frame can contain multiple sub-movements. Therefore, the over-represented classes are still over-represented, since they are more likely to be present in a single frame.

A solution to have a perfectly balanced dataset would be to create a dataset containing only one sub-movement per frame. But this would considerably reduce the size of the dataset, since we would have to remove a lot of frames. So, I decided to keep the dataset as it is, since it is already quite balanced.

To get a clear picture of the difference between the initial dataset and the final dataset, here is a comparison of the two datasets :

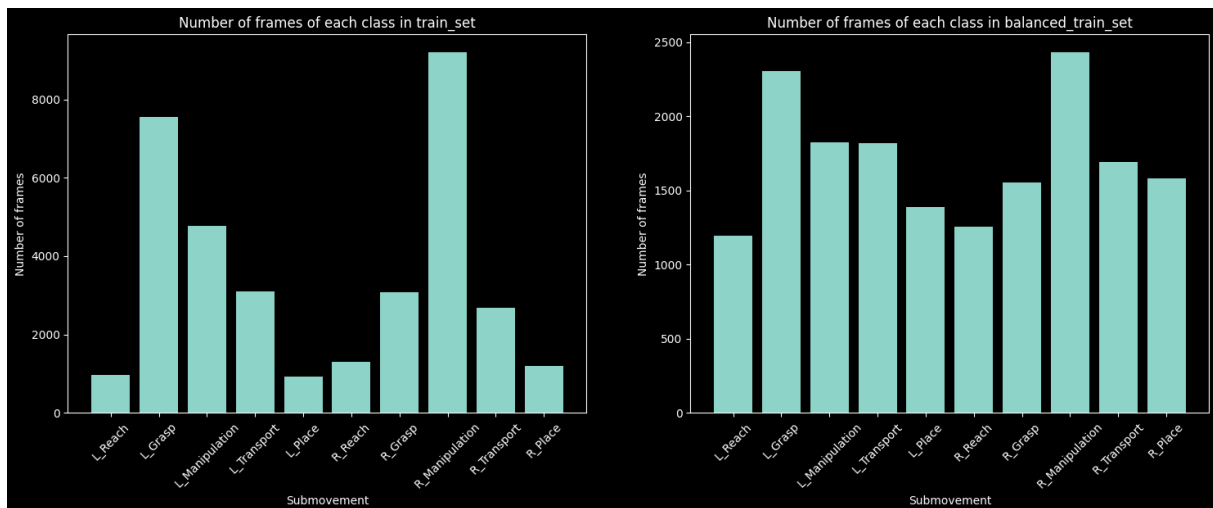


Figure 5.3: Initial vs final classes distribution

If we focus on the size ratio between the "Left Reach" class and the "Right Manipulation" class, we can see that it has been reduced from 1:10 to 1:2. This represents a huge improvement in terms of class distribution.

## 5.4 Second dataset creation process

As mentioned before (see 4.2), it has been decided, due to the lack of time, to use a different approach to train the future model. Instead of using the video frames as input for the model, we will use the output of the MediaPipe and YOLO models. The goal of this dataset is to rely on actual hands and objects detection instead of relying on the video frames. This should improve the model's accuracy, since the model will be able to focus on the hands and objects instead of trying to discover patterns in the video frames.

### 5.4.1 MediaPipe output

This section describes only the output of the MediaPipe model. No specific information about the model itself is given here. To get more details about how the model has been used, please refer to the following section 6.1.

#### 5.4.1.1 MediaPipe output - landmarks

The MediaPipe model has been used to detect the hands in the video frames. In fact, to classify the sub-movements, we need to know what the subject is doing with his hands. So, we need to know where the hands are located, what are their orientations and what are their hand gestures (fist closed, hand open, etc.). The MediaPipe model is able to detect all of these informations for us, since it can detect up to 21 landmarks per hand. These landmarks are located at specific points on the hand, such as the tip or the joints of the fingers. In concrete terms, each landmark is represented by a 3D point (x, y, z) in the video frame. The x and y coordinates represent the position of the landmark in the video frame, while the z coordinate represents the depth of the landmark. Here is an example of the landmarks detected by the MediaPipe model :

```
[0,
  [[203.678, 215.0048, 2.0850416149187367e-07],
   ...
   [183.4057, 175.4153, -0.03252025693655014]],
  None],
[1,
  [[210.2895, 194.1755, -2.3149007688516576e-07],
   ...
   [173.3287, 150.2569, 0.01571262814104557]],
  [[223.4509, 192.7871, 1.3033458756694927e-08],
   [224.9105, 184.4901, -0.018910404294729233],
   ...
   [219.5879, 158.5862, 0.0031116241589188576]]],
[2, None, None],
...
]
```

Figure 5.4: MediaPipe landmarks before cleaning

The code above shows the structure of the landmarks returned by the MediaPipe model. The id we can see (0, 1 and 2) represents the id of the frame. Each frame contains a list. Each list contains two nested lists, which in turn contain 21 nested lists containing the coordinates (x,

y and z) of each landmark. So for each frame we have a total of 42 possible landmarks, which corresponds to  $3 \times 42 = 126$  coordinates.

The problem is that the MediaPipe model is not perfect. Indeed, it can sometimes not detect all the hands. The non-detected hands are represented by a None value. In the example above, we can see that MediaPipe has detected only one hand for the frame 0. The frame 1 contains two hands, so everything is fine. But no hands at all have been detected for the frame 2.

To solve this problem, there is two solutions. The first one is to remove the frames where hands are missing. But this is not optimal, since we would lose a lot of data. The second solution is to replace the missing coordinates by a default value. This is the solution I chose to use. The default value I chose is (0, 0, 0). This way we can keep the frames where hands are missing and the model should be able to learn that the coordinates (0, 0, 0) represents a missing hand. Also, this technique allows us to keep the same structure for all the frames, which is important for the model.

#### 5.4.1.2 MediaPipe output - hands center

In addition to the landmarks, the MediaPipe model also provides us the center of the hand. This center is represented by a 2D point (x, y).

Here is an example of the hands center detected by the MediaPipe model :

```
[[0,
 [208.6511743068695, 179.4220232963562], None],
 [1,
 [168.38631391525269, 161.72376036643982],
 [217.94182062149048, 175.3351378440857]],
 [2, None, None],
 ...
 ]
```

Figure 5.5: MediaPipe hands center before cleaning

Similarly to the landmarks, the center of the hand is not always detected. Actually, the center of the hand is always detected as the same time as the landmarks, since it relies on the fact that the hand has been detected or not. To solve this problem, I used the same technique as for the landmarks. I replaced the missing coordinates by a default value, which is (0, 0).

#### 5.4.2 YOLO output

This section describes only the output of the YOLO model. No specific information about the model itself is given here. To get more details about how the model has been used, please refer to the following section 6.2.

The YOLOv7 model has been used to detect the objects in the video frames. In fact, to classify the sub-movements, it could be interesting to know what objects the subject is manipulating and how he is manipulating them. The YOLOv7 model is able to detect all of these informations for us, since it can detect up to 80 different objects. 80 objects is a lot but it does not contains present in the video frames. For example, the YOLOv7 model is able to detect a banana or

a broccoli, but it has no class for cucumber. Anyway, while testing the model I noticed that the model sometimes interprets the cucumber as a broccoli and once peeled, it interprets it as a banana. So, even if the YOLOv7 model does not contain the exact class name we are looking for, it can still be useful to detect the objects in the video frames.

Here is the list of the class object I decided to keep (the class number + the class name) :

- 39 = bottle
- 43 = knife
- 45 = bowl
- 46 = banana
- 50 = broccoli
- 72 = refrigerator

Each detected object is represented by a vector of 6 values. The first value is the class number, the second and third values are the coordinates (x, y) of the center of the object, the fourth and fifth values are the width and height of the object and the sixth value is the confidence score of the object. The confidence score represents the probability that the object is correctly classified. The higher the score, the higher the probability that the object is correctly classified. Here is an example of the objects detected by the YOLOv7 model :

```
[[0, [[]]],  
 [1, [['39', 25.5, 242.4998, 37.0, 55.0, '0.819336']]],  
 [2, [['39', 73.0003, 236.0, 28.0, 52.0, '0.85791'],  
      ['50', 34.0623, 238.0, 31.0, 34.0, '0.92751']]],  
 ...  
 ]
```

Figure 5.6: YOLOv7 objects before cleaning

As we can see, the YOLOv7 model is not perfect either. In fact, in this example, no objects have been detected for the frame 0. The frame 1 contains one object and the frame 2 contains two objects. There is no limit to the number of objects that can be detected by the model. In theory, a frame could contain 10 objects or more, which could be a problem for us. Indeed, our goal is to classify the sub-movements, not to detect all the object in the video frames. And the more objects there are in the video frames, the more difficult it will be for the model to actually understand what the subject is doing.

To solve this problem, I had to choose which object was the most important one to keep and eliminate the others. More details about how I did this are given in the section 6.2.3. For now, let's just say that I decided to keep only the most important object for each frame.

To solved the problem of missing values, instead of simply add a vector of zero values like I have done for the MediaPipe model, I decided to add a phantom object. More information about this phantom object are given in the section below (see 5.4.3).

Finally, since we only keep one object per frame, we can remove the confidence score from the vector. This way, the vector only contains the class number, the coordinates (x, y) of the center of the object and the width and height of the object.

### 5.4.3 Phantom object

As we have seen in the previous sections, the MediaPipe and YOLOv7 models are not perfect. In fact, they do not always detect the hands and the objects in the video frames. To solve this problem, my first solution was to add vectors of zero values to the dataset. This way, the model could learn by itself that zero values means that the hands or the objects are missing. However, I learned that this solution is not optimal. In fact, the model could understand that an object exists and that its coordinates are  $(0.0 ; 0.0)$ . This is not what we want. Moreover, if we only have zero values for all the missing objects and hands it would be hard to compute the euclidian distance which separates them.

A better solution is to use "phantom" values. The phantom value of an inexistant object is the coordinate which is the furthest away from the hands. In other words, if we don't detect any object, we will simulate that we detect one but very far away from the hands, this way the model should understand that there is no interaction between the hands and the object.

Naturally, the furthest point from a hand depends on the position of the hand. But in all cases, it should be in one of the corner. The image format use for the hands detection is  $300 \times 300$ , same for the object detection (it was  $640 \times 640$  but I have adapted the distances to match the hands pixels one). So we have a square of  $300 \times 300$  pixels, which can be divided in 4 quadrants :

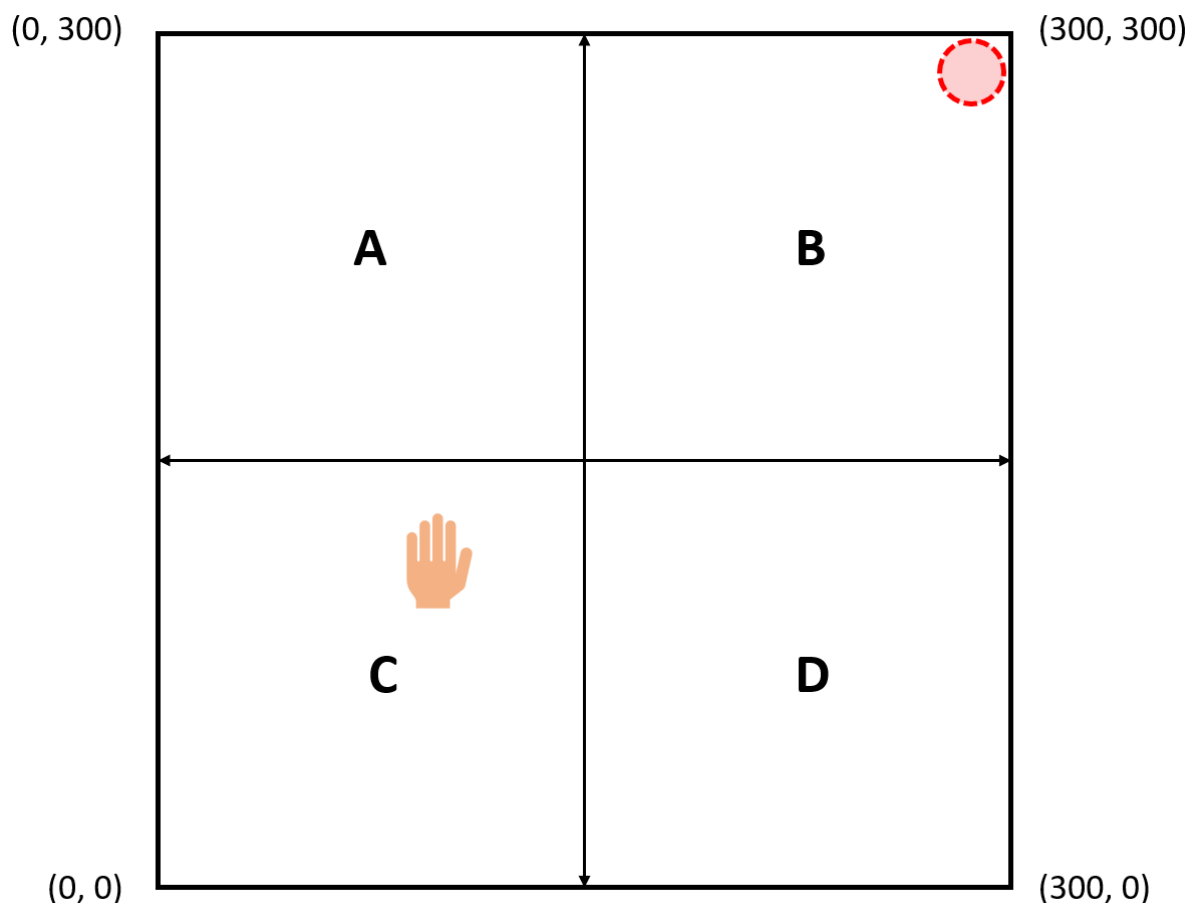


Figure 5.7: The 4 quadrants of the  $300 \times 300$  image

The furthest point from a hand is the corner in the opposite quadrant. For example, in the figure

above, we can see a hand in the bottom left quadrant (quadrant C). The furthest point away is then in the very top right corner. In other words, if the x and y coordinates of the hand center are respectively ( $X < 150$  ;  $Y < 150$ ), then the phantom object will be in the corner of quadrant B ( $X = 300$  ;  $Y = 300$ ).

We now have a very simple way to calculate the phantom value of an inexistant object. We just need to know in which quadrant the hand is and then we can calculate the coordinates of the phantom object. Those coordinates are the following :

- Quadrant A : ( $X = 0$  ;  $Y = 300$ )
- Quadrant B : ( $X = 300$  ;  $Y = 300$ )
- Quadrant C : ( $X = 0$  ;  $Y = 0$ )
- Quadrant D : ( $X = 300$  ;  $Y = 0$ )

Since we replace the None value of missing hands by zero values, it is like if we have a fake hand place in the bottom left corner (quadrant C). This way, we are sure that we will always have hands coordinates to know in which quadrant the phantom object should be placed.

This means that in the particular case where no object is detected and no hands are detected either, the phantom object will placed on the top right quadrant ( $X = 300$  ;  $Y = 300$ ). This way we guarantee that the euclidian distance will be maximal if no hands and no objects are detected.

#### 5.4.4 Distances between the hands landmarks and the object

In addition to the landmarks and the objects, we find relevant to calculate the distance between the hands and the object. Knowing how close the hands are from the object could be useful to classify the sub-movements. For example, if both hands are close to the object, it could mean that the subject is manipulating the object (like peeling a cucumber). On the other hand, if only one hand is close to the object, it could mean that the subject is holding the object (like holding a bottle).

To be precise enough about the distance between the hands and the object, we need to calculate the distance between each landmarks of the hands and the center of the object. This way, we can know exactly how close each hand is from the object. The distance between two points is calculated using the euclidian distance formula :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (5.1)$$

The structure of the distances between the hands and the object is quite similar as the one of the landmarks. The x and y coordinates of the landmarks have been replaced by the distances and the z coordinate has been removed. Here is an example of the distances between the hands and the object :



```
[
  [1, [251.55424524, 245.39239099, 249.03576008, 252.34618057,
    ...
    258.46003483]],
    [[424.26406871, 424.26406871, 424.26406871, 424.26406871,
    ...
    424.26406871]]],
  [2, [239.47975877, 241.96706373, 242.00371242, 241.49409469,
    ...
    229.39044061]],
    [[295.12229776, 290.89745617, 283.45751984, 277.31343042,
    ...
    278.1492925 ]]],
  ...
]
```

Figure 5.8: Example of the structure of the distances between the hands and the object

As we can see in the figure above, we don't have None values for the distances. This is because we use the phantom object method explained in the section 5.4.3. This ensures that we always have a value for the distances between the hands and the object.

The structure of the distances between the hands and the object is the following : for each frame, we have a list of 21 distances for each hand. So we have a total of 42 distances per frames. If we pay attention to the figure 5.8, we can see that the values of the second list of distances of the first frame is always the same : 424.26406871. This is not an error. In fact, this is the maximal euclidian distance we can have between two points in a 300x300 pixels. This means that the second hand of the first frame has not been detected and no object has been detected either.

#### 5.4.5 Final data structure

Now that we have gathered and cleaned all the informations we need, we can create the final data structure. This data structure will be used to train the model and to test it. It is a cluster of all the outputs of the models mentioned above. Here is a recap of the informations we have gathered, each value will be consider as an input column for the model :

- Number of columns for the hands landmarks : 126
- Number of columns for the object : 5
- Number of columns for the distances between the hands and the object : 42
- Number of columns for the hands center : 4

At this point, we have a total of 177 columns. To this, we need to add the output column, which is the sub-movement class. It corresponds to 10 different classes, so we need to add 10 columns to the data structure. This gave us a total of 187 columns. This means that the data structure is a matrix of 187 columns and N rows, where N is the number of frames in the video. Each row represents a frame and each column represents an input or output value.

The image below shows the shape and the first 5 rows of the data structure :

	center_0_X	center_0_Y	center_1_X	center_1_Y	obj_class	obj_X	obj_Y	obj_width	obj_height	landmark_0_0_X	...
0	195.091320	199.277472	0.000000	0.000000	4.0	0.0	0.0	1.0	1.0	203.6780	...
1	208.651174	179.422023	0.000000	0.000000	4.0	0.0	0.0	1.0	1.0	210.2895	...
2	90.084400	129.034737	0.000000	0.000000	4.0	300.0	300.0	1.0	1.0	95.7721	...
3	168.386314	161.723760	217.941821	175.335138	4.0	0.0	0.0	1.0	1.0	166.9457	...
4	242.268548	193.365412	0.000000	0.000000	4.0	0.0	0.0	1.0	1.0	236.6061	...

5 rows × 187 columns

Figure 5.9: Final data structure

By looking closely at the object columns, we can see that no objects have been detected in the first 5 frames. Phantoms objects have been placed instead. We can recognize the phantom object thanks to its class number : 4.

As mentioned in the section 5.4.2, YOLOv7 can recognize up to 80 different objects. Some of these objects are more likely to be detected within the framework of this project, than others. For example, it would not be surprising to see a patient holding a knife or a bottle.

To have a clear distinction between the objects that are detected and the phantom ones, I needed to give the phantom object a class number that is very unlikely to be present in the video we use in this project. This is why I gave the phantom object the class number 4, which is the class number of the "airplane" object. Since it is impossible for a patient to hold an airplane, we are sure that when we see the class number 4, it is a fake object.

Additionally, we can see that most of the phantom objects are placed in the bottom left corner ( $X = 0$  ;  $Y = 0$ ). This is because the hands have been detected in the top right corner ( $X > 150$  ;  $Y > 150$ ). Except for the second frame, where the hand is in the bottom left corner ( $X < 150$  ;  $Y < 150$ ), so this time this is the object that is placed in the top right corner ( $X = 300$  ;  $Y = 300$ ).

This final dataframe is the one that is used to train the classification models. The implementation of the classification models is explained in the section 6.3 and 6.4.

## 6 Implementation

This chapter contains explanations on the various algorithms and models implemented during this project, and their results. No code details are discussed here, since the full code is available on the GitLab repository via the following link: [Vision-based segmentation of hands movements code](#).

### 6.1 MediaPipe model

As explained in the previous chapter, the MediaPipe model is used to detect the hands in the frames of the videos. More precisely, the model detects the center of the hands, as well as the 21 landmarks of each hand. The output the model produces is an essential part of the final dataset, used to train the submovements classification model.

#### 6.1.1 Model input

The MediaPipe model takes as input a video frame. Those frames could, in theory, have any resolution. However, as explained in previous chapter (see [chapt. 2.2.3](#)), the frame resolution is an important parameter to take into account, in order to obtain the best possible results with the model. In this case, it seems that MediaPipe works best with frames of resolution 300x300 pixels.

To maximize the chances of obtaining good results, the original frames of the videos were resized to 300x300 pixels using two different methods : padding and cropping. Those two methods are presented in the [chapter 2.2.3.1](#).

Having two different versions of the frames allowed me to run the model twice on each frame, and merge the results afterwards, in order to obtain a more complete dataset.

#### 6.1.2 Hands center detection

The first output of the MediaPipe model is the center of the hands. This center is represented by a point, which coordinates  $x$  and  $y$  are expressed in pixels.

Running MediaPipe model is actually quite simple. The model is already implemented in the MediaPipe library, which is available in Python. The only thing to do is to import the library, and call the function that runs the model. The function takes as input the frame to analyze, and returns the coordinates of the hands center.

There are two variables that are important to take into account when running the model :

- **max\_num\_hands** : this variable represents the maximum number of hands that the model can detect in a frame. since both hands can be used to perform a gesture, this variable is set to 2.
- **min\_detection\_confidence** : this variable represents the minimum confidence value that the model must have in order to detect a hand. If the confidence value is lower than this threshold, the model will not track any hand. I try to keep this value as low as possible, in order to maximize the number of frames in which the model detects the hands. However, if the value is too low, the model will detect hands in frames where there are no

hands. This will result in a lot of noise in the dataset, and will make the training of the submovements classification model more difficult. After some testing, I decided to set this value to 0.5.

### 6.1.3 Hands center results merging

As explained in the previous section, I run the MediaPipe model twice on each frame, using two different resizing methods. This allowed me to detect more hands centers in the frames, and to obtain a more complete dataset.

The figure below shows the results of the MediaPipe hands center detection :

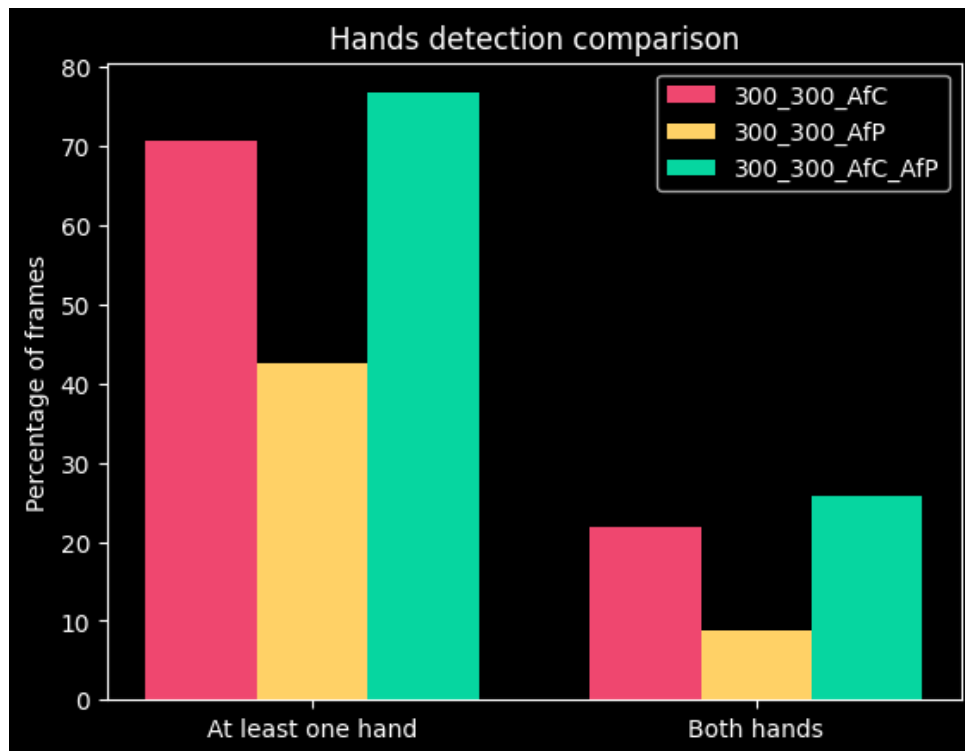


Figure 6.1: MediaPipe hands detection results

The Y axis represents the percentage of frames in which the model detected the hands center. On the X axis, we can see two "group" of bars. The first group shows the percentage of frame in which at least one hand has been detected. The second group shows the percentage of frames in which both hands have been detected. The red bars correspond to the results obtained with the cropping method, the yellow ones to the results obtained with the padding method, and the green ones to the results obtained by merging the two previous results.

As we can see, the results obtained with the cropping method are better than the ones obtained with the padding method. This is probably due to the fact that the padding method adds a lot of noise to the frames, which makes it more difficult for the model to detect the hands. However, the merging operation shows that even if the cropping method is better, the padding method still allows us to detect hands in frames where the cropping method failed. This is why I decided to use both methods, and merge the results afterwards.

We can also see that, even with the merged results, we don't reach 100% of frames in which at least one hand is detected. This is simply due to the fact that not all the frames of the videos

contain hands.

To have a better idea of the structure of the output of the MediaPipe model, please refer to the chapter 5.4.1.2.

#### 6.1.4 Hands landmarks detection

The second output of the MediaPipe model is the landmarks of the hands. This output is a list of 21 points, which coordinates x, y and z. The coordinates x and y are expressed in pixels, and the coordinate z is expressed in millimeters. The coordinate z is used to determine the depth of the hand.

To get the all the landmarks of the hands, I use a very similar method as the one used to get the hands center. The only difference is the number of landmarks I saved and the fact that I did not calculate the center of the hands. Otherwise, the method is the same, I keep the same values for the variables `max_num_hands` and `min_detection_confidence`.

#### 6.1.5 Hands landmarks results merging

Similarly to the hands center detection, I run the MediaPipe model twice on each frame, using two different resizing methods. This allowed me to detect more hands landmarks in the frames, and to obtain a more complete dataset.

The number of landmarks detected by the model is exactly the same as the number of hand center detected. In other words, if the model detects a hand, it will detect all its landmarks. And if we have the landmarks, we can calculate the center of the hand. So to have a graphical representation of the results, please refer to the figure 6.1.

To have a better idea of the structure of the output of the MediaPipe model regarding the hands landmarks, please refer to the chapter 5.4.1.1.

## 6.2 YOLO model

To detect the objects in the frames, I used the YOLOv7 model. It produces a list of vector containing the class name of the object, its coordinates (x and y), its height and width, and its confidence value. The confidence value represents the probability that the object is correctly classified. This list of vector is used in the final dataset.

### 6.2.1 Model input

Just like the MediaPipe model, the YOLO model takes as input a frame. However, the YOLO model requires the frame to be in a specific format, like mentioned in the chapter 2.2.3. The format which gives the best results is the one with a resolution of 640x640 pixels. To get this format from the original frame, I used the cropping method, similarly to the one used for the MediaPipe model.

### 6.2.2 Running the model

To run the YOLOv7 model, we first need to load the model. But before even thinking about loading the model, we need to be sure that we will be able to run it. To do so, we have to set up a miniconda environment with all the required dependencies.

Here are the steps to follow to set up the miniconda environment.

In a Linux terminal, run the following commands :

Make sure you have wget installed :

```
apt-get install wget
```

Then, download the miniconda installer :

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-  
x86_64.sh  
sha256sum Miniconda3-latest-Linux-x86_64.sh  
bash Miniconda3-latest-Linux-x86_64.sh
```

Create the miniconda environment :

```
conda create -n yolov7 python=3.8  
conda activate yolov7
```

Clone the YOLOv7 repository :

```
!git clone https://github.com/WongKinYiu/yolov7
```

Install the required dependencies :

```
cd yolov7  
pip install -r requirements.txt
```

Run the model :

```
python detect.py --weights yolov7-e6e.pt --conf 0.80 --img-size  
640 --source --save-txt --save-conf --nosave
```

The command above will run the "detect.py" script of the YOLOv7 repository. This script will run the model on the frames of the video, and save the results in a folder, using the following arguments :

- **--weights yolov7-e6e.pt** : the path to the weights of the model. In this case, the weights are the ones of the YOLOv7 model.
- **--weights yolov7-e6e.pt** : the path to the weights of the model. In this case, the weights are the ones of the YOLOv7 model.
- **--conf 0.80** : the confidence threshold. The model will only keep the objects with a confidence value higher than 80%.
- **--img-size 640** : the resolution of the frames. In this case, the resolution is 640x640 pixels.
- **--source** : the path to the images or videos to run the model on.
- **--save-txt** : save the results in a text file.
- **--save-conf** : save the confidence value of the objects.
- **--nosave** : specify that we do not want to save the images with the bounding boxes of the detected objects.

### 6.2.3 Cleaning model results

The commands above will save the results of the model in a folder. The results are saved in a text file. Each frame has its own text file. So the first thing to do is to create a numpy array with all the results of the model. To do so, I used the following code :

```
labels = ["0" if txt_path not in TXT_FILES else open(txt_path, "r")
          .read() for txt_path in tqdm([PATH + "frames" + str(i).zfill
          (6) + ".txt" for i in range(NB_IMG)])]
labels = np.array(labels)
```

Once in possession of an numpy array, I was able to clean the data. After the cleaning process, I obtained a list of vector containing the class name of the object, its coordinates (x and y), its height and width, and its confidence value.

Detecting multiple objects in a frame could be useful in some cases, but not in this one. Indeed, we are only interested in detecting the movement the patient is doing. So we only need to keep the object she is interacting with. In other words, we only want to keep the object which is the closest to the hand.

To know which object is the closest to the hand, I calculated the euclidian distance between the hand center and the object center. It is the same formula as the one used in the chapter 5.4.4, except that this time, we are calculating the distance between the hand center and the object center, and not the distance between the all the landmarks and the object center. The one with the smallest distance is the one we want to keep.

In total, there is 21'068 frames with (at least) an object detected. Since there is a total of 28'854 frames, it means that more than 73% of the frames have an object detected.

Since we know the object values to calculate the euclidian distances between the hand center and the object center, I use a trick to create "fake" objects for the frames without any object detected. Those "fake" objects are known as "phantom objects", their creation process is explained in the chapter 5.4.3.

## 6.3 Submovements classification model

After using the MediaPipe and YOLOv7 models to create a dataset and cleaning it, it was finally time to create the submovements classification model. This model objective is to classify the frame of the video into the different 10 submovements. To do so, I used a deep learning model, implemented with the Keras library.

### 6.3.1 Model architecture

The first thing to do was to choose the architecture of the model. Multiple architectures could have been used, but I decided to use a convolutional neural network (CNN). The reason behind this choice is that CNNs are often used for image classification. Even if the input of the model is not directly an image, it is a vector containing multiple coordinates values, representing the position of the hand and the object. There is a similarity in the data structure we have and the data structure of an image. So I thought that a CNN would be a good choice.

I was also considering trying to use a recurrent neural network (RNN). But to the best of my knowledge, RNNs are particularly useful when we have sequential data or when we want to do natural language processing (NLP). Since it does not match the nature of our data, I decided to not use a RNN.

I would have liked to have more time to explore other architectures, but I had to make a choice, since I only had a few days left to finish this project, as explained in the chapter 8.1.

The architecture of the model is the following :

```
model = Sequential()  
model.add(Dense(units=64, activation='relu', input_shape=(X_train  
    .shape[1:]))  
model.add(Dropout(0.4))  
model.add(Dense(10, activation='softmax'))
```

As we can see, the model is extremely simple. It is composed of only three layers. The first layer is a dense layer with 64 units, using the ReLU activation function. The second layer is a dropout layer with a dropout rate of 50% which is a good practice to avoid overfitting. The last layer is a dense layer with 10 units, using the softmax activation function. The softmax activation function is used because we want to do a multiclass classification.

For the compilation of the model, I used the following code :

```
model.compile(loss='categorical_crossentropy', optimizer='adam',  
    metrics=['accuracy'])
```

I used the categorical crossentropy loss function because we are doing a multiclass classification. I used the Adam optimizer because is one of the most used optimizer. I also used the accuracy metric to evaluate the model.

Finally, I trained the model with the following code :

```
history = model.fit(X_train, y_train, epochs=200, batch_size=32,  
    validation_data=(X_val, y_val))
```

I trained the model for 200 epochs with a batch size of 32. The number of epochs can seem low but it was enough to see that the model was not learning.

The hyperparameters have been chosen after running a grid search. The results of the grid search determine the best hyperparameters to use. However, I did not have time to run very complexe grid searches. So I can not guarantee that the hyperparameters I used are the best ones. But by looking at the model results, I don't think that changing the hyperparameters would have a big impact on the results of the model, since the model does not perform very well as shown in the next chapter 6.3.2.

### 6.3.2 Model results

We're not going to keep the suspense going, the model does not perform very well on this 10 classes classification task. We are going to try to understand why in this chapter.

First, let's look at the accuracy and loss curves of the model :



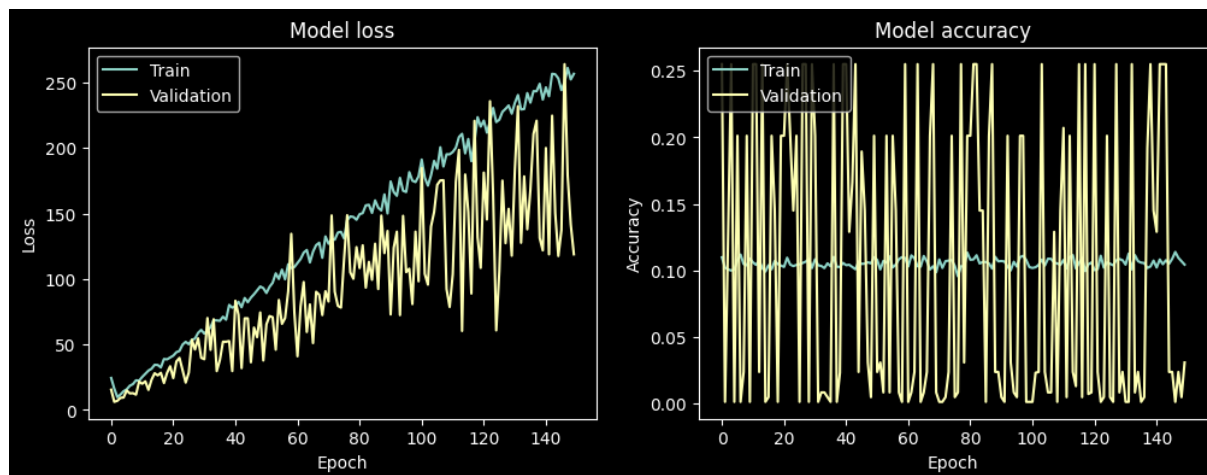


Figure 6.2: Accuracy and loss curves of the submovements classification model

It is clear that the model is not learning. The accuracy is stuck at 0.1. Since we have 10 classes, it is similar to a random classification. In other words, the model is not learning anything and keeps predicting the same class for every frame.

The loss is deteriorating linearly. It is another sign that the model is not learning anything. If the model was learning, the loss would decrease and then stabilize.

To confirm that the model is not learning, we can look at the confusion matrix of the model :

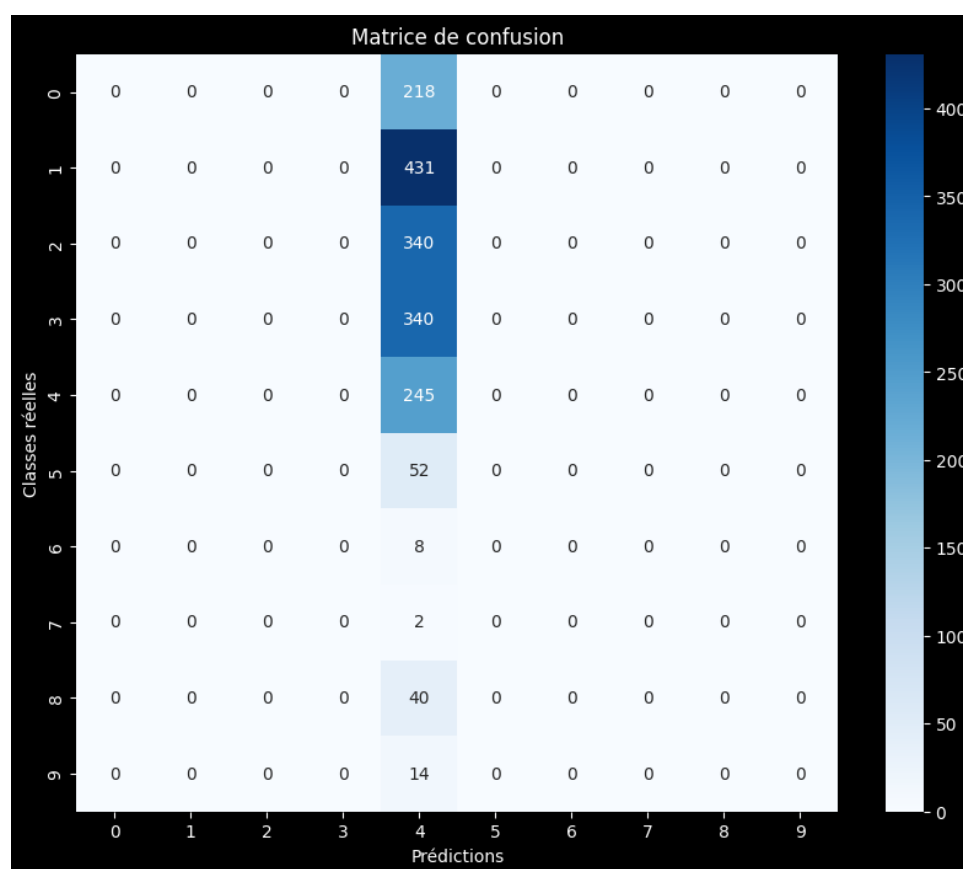


Figure 6.3: Confusion matrix of the submovements classification model

As we can see, the model is always predicting the same class. In this case is the class 4, but it could have been any other class.

First, I thought that the model was not learning simply because its structure was too simple. So I tried to add more layers to the model, to make it more complex and giving it more chances to learn more complex patterns. But it did not change anything, the model was still not learning. Actually it was even worse since the loss would now grow exponentially, as shown in the figure below :

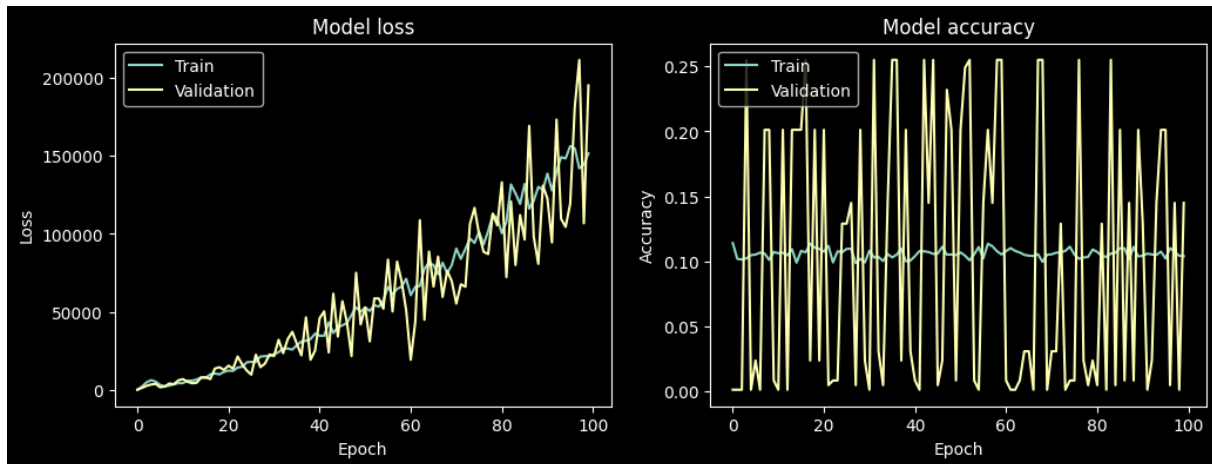


Figure 6.4: Accuracy and loss curves of the submovements classification model with more layers

I also tried to use a different optimizer and loss function, but it did not help, the model was still not learning.

One more thing to mention is that I tried to train the model with the first version of the dataset, the one with the vectors of zero values instead of the phantom (when the hand or the object is not detected) and I got very similar results.

All those tests lead me to think that the problem was maybe not coming from the model itself, but from the data. This is only a hypothesis, since I do not have a lot of experience and I did not have time to investigate more deeply. Multiple potential solutions to put in place are discussed in the chapter 8.2.

## 6.4 Binary classification model

Since I did not want to end on a failure, I decided to try to do a binary classification. The idea was, before trying to classify the different submovements, it would have been nice to be able to detect if there is a submovement or not.

### 6.4.1 Dataset modification

To create this dataset, I used the same dataset as the one used for the previous model. But I removed the 10 classes and replaced by a single class. The class is called "submovement" and contains a "1" when there is a submovement and a "0" when there is not.

After creating this dataset, I realized that the dataset was not balanced anymore. Indeed, the video contains a lot more frames with submovement than frames without submovement. So I

had to reduce the number of frames with submovement to have a balanced dataset. This lead me to a final dataset of shape (4144, 177).

### 6.4.2 Model architecture

The model architecture I used is a bit more complexe than the one used for the previous model. It is composed of 5 layers :

```
model = Sequential()  
model.add(Dense(units=64, activation='relu', input_shape=(X_train  
    .shape[1],)))  
model.add(Dropout(0.3))  
model.add(Dense(units=64, activation='relu'))  
model.add(Dropout(0.3))  
model.add(Dense(1, activation='sigmoid'))
```

The first layer is a dense layer with 64 units and a relu activation function. The second layer is a dropout layer with a dropout rate of 0.3. The third layer is a dense layer with 64 units and a relu activation function. The fourth layer is another dropout layer with a dropout rate of 0.3. The last layer is a dense layer with 1 unit and a sigmoid activation function. The sigmoid activation function is used because we are doing a binary classification.

To compile the model, I used the Adam optimizer and the binary crossentropy loss function which is the most common loss function for binary classification.

Finally I used the following code to train the model :

```
history = model.fit(X_train, y_train, epochs=200, batch_size=16,  
    validation_data=(X_test, y_test))
```

All the hyperparameters have been chosen after runing a grid search, just like for the previous model.

### 6.4.3 Model results

The model results are much better than the previous model, since this time, the model is learning something. However, the results are not amazing. Let's look at the accuracy and loss curves of the model :

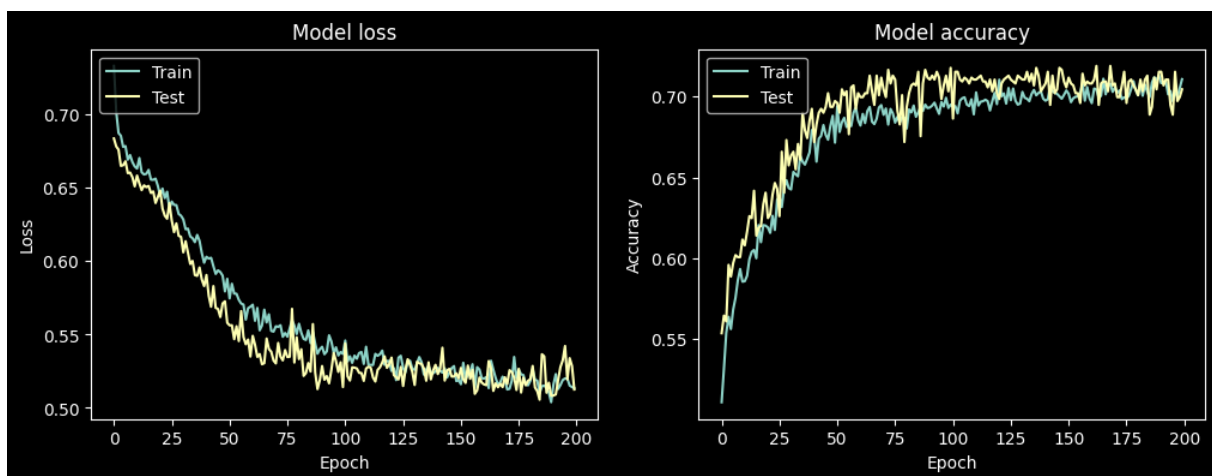


Figure 6.5: Accuracy and loss curves of the binary classification model

As we can see, the model is learning, but it is not learning very well. The accuracy is stuck around 70%. It is better than a random classification, but it is not enough to be considered as a good model.

The loss is decreasing and then stabilizing. It is a good sign, it means that the model is learning. However, the loss is still quite high since it is stuck around 0.52. At least, there is no big difference between the training loss and the testing loss, which means that the model is not overfitting.

Here is the confusion matrix of the model :

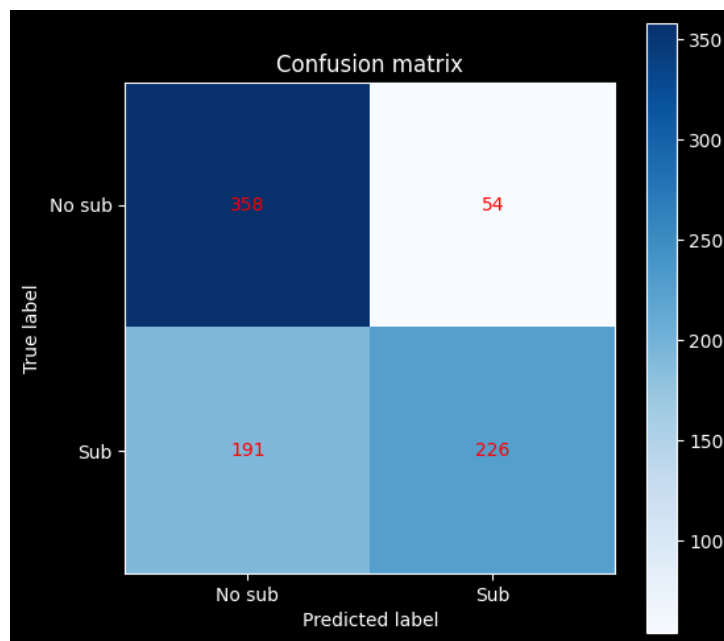


Figure 6.6: Confusion matrix of the binary classification model

Those results were obtained with the dataset using the phantom. I also tried to train the model with the dataset using the vectors of zero values and I was surprised to see that the results better, as shown in the figure below :

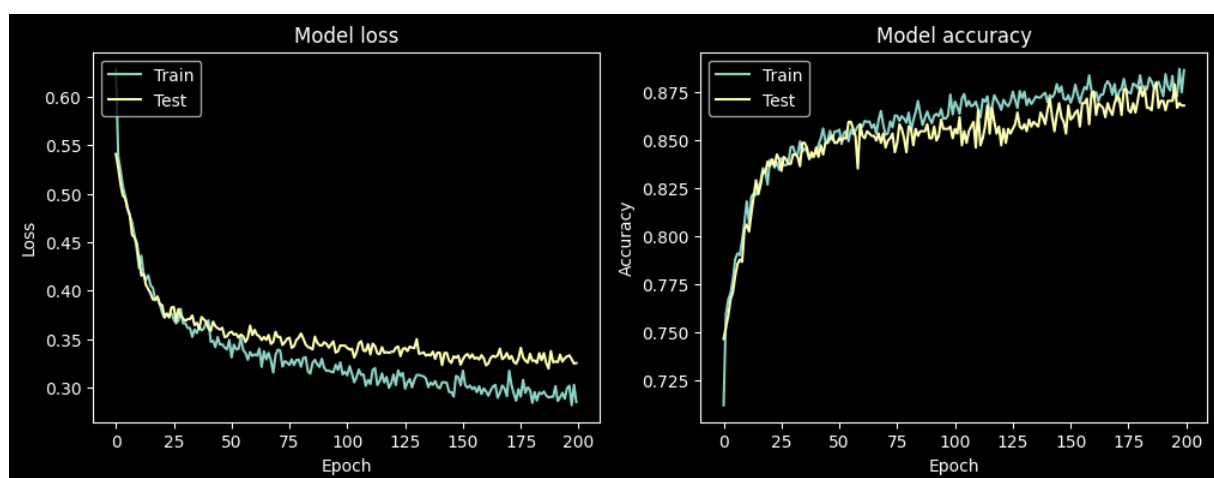


Figure 6.7: Accuracy and loss curves of the binary classification model with zeros values instead of the phantom

As we can see, the accuracy was much better, it reached 88% with the training set and get stuck around 87% with the testing set. The loss was still not very good, but it was better than the previous model, since it was stuck around 0.35.

I am not sure to understand why the model is performing better with this dataset than the one with the phantom. My hypothesis is that model is learning to detect the zeros values and not really trying to learn the actual hands or object positions. If it is the case, it would explain why the model is performing better since when a frame is labelled as "no submovement", it means there is no interaction between hands and objects in the frame. And with the dataset using the phantom, we still compute distances between hypothetical hands and objects positions. This distance is the maximal one, so we could think that the model would take it into account and learn that high distances means no interaction. But my guess is that the model is not learning that, it is just checking if the distance is equal to zero or not to help him determine if there is a submovement or not.

## 7 Problems encountered

This chapter gathers all the problems I encountered during this project and their solutions (as far as they have been found).

### 7.1 Missing frames

The "missing frames" problem is a problem that occurs when I extract the ground truth from the video. The ground truth is the label of each frame of the video.

The video is a bit more than 8 minutes long, which correspond to exactly 28'854 frames. Each single frame is labelled with 10 labels. So, in theory, the ground truth should contain 28'854 labelled frames.

However, when I extract the ground truth from the video, I only get 28'826 frames. This means that 28 frames are missing.

I tried to re-extract the ground truth from the video multiple times, with different tools, but I always get the same result. At this point, I don't know why these frames are missing. I tried to find a pattern in the missing frames and it seems that the missing frames are always at the end of the video.

Having 28 missing frames is not a big problem, since it only represents 0.1% of the total number of frames. However, it is still a problem we need to solve, since it means that the ground truth shape does not match the dataset shape (which contains all the 28'854 frames).

The easiest solution I found is to simply remove the last 28 frames of the dataset. This way, the ground truth shape matches the dataset shape. This allows us to merge the dataset and the ground truth together, in order to create a single dataset to train our model.

### 7.2 Missing computing power

I quickly realized that my computer was not powerful enough to train a model on the entire dataset. I first tried to split the dataset into multiple smaller datasets, but in addition to being time-consuming, this solution was not very efficient.

To overcome my lack of computing power, I asked for help to my supervisor, who gave me access to a server of the School of Engineering and Management of Vaud (HEIG-VD). To use this server, I have to install an application called "GlobalProtect". This application allows me to connect to the HEIG-VD network, via a VPN. Once connected, I can access the server via SSH.

### 7.3 Unbalanced datasets

As a reminder, we have defined 5 different sub-movements (see chapt. 1.2.2.2). Each sub-movement can be performed with the left hand or the right hand. This means that we have a total of 10 different labels.

The egocentric video used to create the dataset shows a person performing various cooking tasks. Those tasks are, for example, washing vegetables and cutting them, taking a plate out of the

tackle , or washing dishes. Most of the tasks are performed with the right hand since the person is right-handed. In addition, some tasks take considerably more time than others. For example, the person spends just a few seconds taking a plate out of the tackle, while she spends several minutes cutting vegetables.

All these factors lead to an unbalanced dataset. Indeed, some labels are more represented than others. For example, the "manipulation" label is more represented than the "placing" label. This is because when the person is cutting vegetables, she is performing the "manipulation" sub-movement. However, the "placing" sub-movement is only performed when the person is putting the vegetables or dishes on the worktop, an action that is not time-consuming.

This unbalanced dataset is a problem since it can lead to a biased model. Indeed, the model will be more likely to predict the "manipulation" label than the "placing" label, since the "manipulation" the label is more represented in the dataset. The disparity between the number of frames of each label is shown in fig. 5.1.

Multiple solutions can be used to balance a dataset. A common solution is to use oversampling methods. This method consists in duplicating the under-represented classes, to have the same number of frames for each class. However, this method is not very efficient, since it can lead to overfitting.

Another solution is to use undersampling methods. This method consists in removing frames from the over-represented classes, to have the same number of frames for each class. This is the method I used to balance the dataset. The hardest part of this method is to find the right number of frames to remove from each class. Indeed, if we remove too many frames, we will end up with a dataset that is too small to train a model. On the other hand, if we remove too few frames, the dataset will still be unbalanced. The technique I used to find the right number of frames to remove is explained in chapter 5.3.

In the end, I managed to create a quite well-balanced dataset, since the ratio between the number of the less represented class and the number of the most represented class fell from 1:10 to 1:2, as explained in fig. 5.3.

## 8 Conclusion

This chapter concludes this documentation by setting out a number of points such as:

- Time management
- Futur work
- Personal conclusion
- A word of thanks
- A declaration of honour

### 8.1 Time management

I have not done any Gantt chart or other time management tool, at the beginning of the project. Partly because all the objectives were not clearly defined and we did not know exactly how we would proceed. I think this was a mistake. Indeed, now that I am at the end of the project, I realize that I have spend too much time on some parts of the project and not enough on others.

The figure below resumes the time repartition of the project, in a very simplified way :

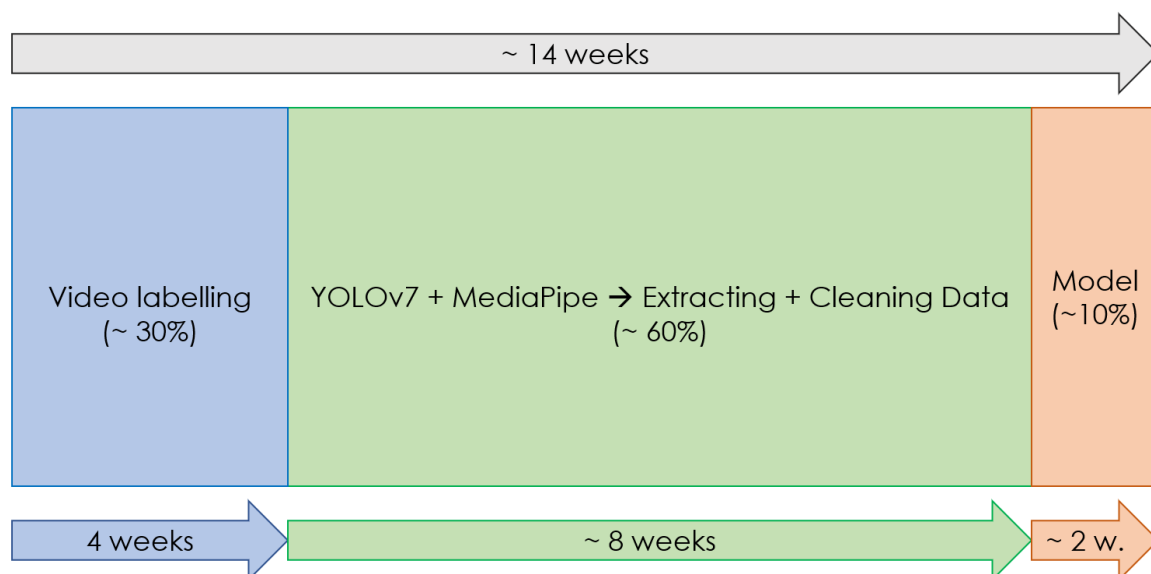


Figure 8.1: Simplified time repartition

As we can see, the total duration of the project is around 14 weeks. I spend one month (around 30% of the time), on the video labeling part. This is a lot of time, but I think it was necessary to do it properly because we can not have a good model without a good dataset.

Then, I spend around 2 months (more than 60% of my time) working with MediaPipe and YOLOv7 models. This work includes learning how to use the models, data extraction, and data cleaning.



Finally, I had less than two weeks to work on the classification model design, training, and testing. This is not enough time, and I think it is one of the reasons why I did not get good results with my actual models.

I am not trying to say that having better time management would have led to amazing results, but I am sure that it would have helped me to better manage my time and to have a better overview of the project.

One last thing to take into account is that it is the first time I am working on a project of this size in this field. I had no experience at all with data liberalisation or with MediaPipe and YOLOV7 models. So, I had to learn everything from scratch, and this took me a lot of time.

## 8.2 Futur work

As I said in the previous section, I did not have enough time to work on the classification model. If I had more time, here are the things I would have done:

- Try different models
- Try to optimize the hyperparameters by running complexe grid search
- Try different data normalization techniques
- Try to get more data

Firstly, I would have tried different models. I focus my work on the CNN model because in my opinion and according to the knowledge I learned during this year's lessons, it is the best model for image classification and therefore, probably the best model for this project. However, I think it would have been interesting to try other models such as SVM or Random Forest. I am not sure that they would have given better results, but it would have been interesting to try.

Secondly, I would have tried to optimize the hyperparameters by running more complex (and time-consuming) grid search. This would have allowed me to find the best hyperparameters for my model and therefore, to improve its performance.

Thirdly, I would have tried different data normalization techniques. I choose to use the Min-MaxScaler because it is the one I know the best. However, I think it would have been interesting to try other techniques such as the RobustScaler.

Finally, if all the previous steps did not lead to better results, I would have tried to get more data. This could be done by labeling more videos or using techniques of data augmentation such as flipping, rotation, or cropping.

With all these steps, I am sure that we could have improved the performance of the model.

## 8.3 Personal conclusion

To conclude this report, I would like to begin by saying that I am proud of the work that has been done as part of this in-depth project.

Although the results are not what I had hoped for, I am satisfied with the work I have done. I've been able to put into practice the knowledge I've acquired during the school year and I've learned a lot about the subject of data liberalization and object detection models detection models (MediaPipe and YOLOv7).

I must admit I'm a bit disappointed that I didn't have more time to play around with building my classification models. of my classification models. Despite this, I really enjoyed working on a project of this scale and in this field. project on this scale and in this field. I think I learned a lot from it, and I'm sure that if I I think I've learned a lot, and I'm sure that if I had to do this project again, I'd be able to get better results and, above all, I'd be able to do it in less time.

## 8.4 Acknowledgement

I want to thank my supervisor, Mr. Perez Uribe, for his help and his availability throughout the project.

I also would like to thanks Ph.D. student Yasaman Izadmehr for her precious help and her advices. She was a great help in determining the ideal frame size to use for the YOLO and MediaPipe algorithms and, more generally, throughout this project.

## 8.5 Declaration of honor

I, the undersigned, Sam Corpataux, declare on my honor that the work submitted is the result of my own work. I certify that I have not resorted to plagiarism or any other form of fraud. All sources of information sources used and author citations have been clearly stated.

# Bibliography

- [1] MathLab community. Matlab video labeler : getting started. <https://www.mathworks.com/help/vision/ug/get-started-with-the-video-labeler.html>. Access the: 14.03.2023.
- [2] Tarang Shah. About train, validation and test sets in machine learning. <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>. Access the: 18.03.2023.
- [3] V7 Labs. Yolo : Algorithm for object detection. <https://www.v7labs.com/blog/yolo-object-detection/>. Access the: 22.03.2023.
- [4] MediaPipe community. Official github repository of mediapipe. <https://github.com/google/mediapipe/>. Access the: 21.03.2023.
- [5] A. Perez Uribe Y. Izadmehr. Enhancing hand and object detection for monitoring patients with upper-limb impairment: A study on the impact of input size in foundation models.
- [6] FFmpeg community. A complete, cross-platform solution to record, convert and stream audio and video. <https://ffmpeg.org/>. Access the: 27.03.2023.