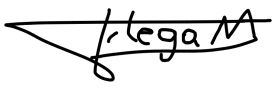
Colorado State University internship report

In Fort Collins, with Dr Francisco Ortega

Dataset segmentation and gesture recognition based on recurrent neural networks

March – July 2020

Erwan Le Pluard
erwan@lepluard.fr



Internship supervised by Cédric Buche and the Brest National School of Engineering (ENIB)

Summary

Acknowledgments

I would like first of all to thank Dr Francisco Ortega, my internship supervisor, for his trust, his help and his implication before and along my whole internship whether technically or socially. He took on his time to know my level and give me required documentation in order to learn what I needed about artificial intelligence, and until the end of the internship he helped and kept contact as much as possible during those complicated times of COVID crisis.

I also thank every members of the NUI Lab for welcoming me so nice and for their help, especially Dhruva Patil which gave me lots of answers I needed during my work.

I thank the entire staff and structure of the Colorado State University for its hospitality and the access to those buildings, as well as the Brest National School of Engineering for its link which made this opportunity possible.

And I would like finally to thanks Cédric Buche for being my school supervisor and especially for giving me the opportunity to do my internship here at CSU in the United States, thanks to his contact with the Dr Ortega.

Introduction

As part of my schooling that I want to orient towards Information Technology (IT), I have already completed my technician internship in the research & development department of a French company (at Micro Module, where I developed an electronic lock with synchronized access). I enjoyed the experience a lot and I wanted to continue my discovering of the research environment in a more academic way.

I also wanted this long internship experience to be abroad in order to get out of my comfort zone, strengthen my oral English skills, and work in a different cultural environment. I then looked for internship offers in the USA; and I eventually had this opportunity thanks to Cédric Buche to come with Auguste Cousin, another ENIB student to the Colorado State University.

After a long process of paperwork and a plane trip, I then joined Francisco Ortega and the rest of the Natural User Interaction Laboratory (NUI Lab) members for a great experience in Colorado. As this laboratory of the Computer Science department of CSU is mainly focused on research about user interaction, gesture recognition tasks represent therefore a big interest of the lab.

Since algorithms of gesture recognition often require for their training many labelled gesture videos well isolated, lot of time has to be involved to manually create datasets by cutting and labelling each gesture in a video. My internship mission was to automate this process which could represent a huge time saving.

I then developed with Auguste and the help of a NUI Lab member an AI-based classification algorithm to recognize the frames of a video which could represent a movement beginning or ending; to finally segment each movement between those frames and then use a recognition algorithm to identify the gestures.

CSU presentation

Colorado State University

The Colorado State University is one of the land-grant universities[1] founded to respond to the industrial revolution needs in 1870, by focusing on the teaching of science, military science, engineering, and agriculture.

From initially an only agricultural college in 1879, it is now a huge university covering more than 60 academic fields of research with 2,000 faculties in 8 colleges. It has a growing number of around 34,000 students, and a huge campus to host them.

Figure 1, photo took just before the COVID-19 stay-at-home order

I was impatient to discover this typical American university in terms of scale compared to what I'm used to, and I have to admit I wasn't disappointed on that point. When I first arrived in the campus I was completely lost.

While used to the ENIB campus and its 3-4 buildings, I discovered at CSU a whole city in the city. Dozens of buildings (see figure 3). One (or more) for each of the university research fields, 2 huge stadiums, a student center with many fast foods in it… There is even a whole pub restaurant with a local brewery in the basement of the Student Center (see figure 2).

I also took a gym membership (that I could only use two weeks before the university closed because of COVID-19) and the Practice Facility amazed me once again by its huge size, bigger than any gyms I could see in France, and an incredible amount of equipment.

[1] State university of higher education which benefits of the Morrill Acts of 1862 and 1890, initially signed by Abraham Lincoln to found modern educational institutions by granting land to the states, with precise goals about modern skills and new technologies.

Figure 3, local brewery at Lory Student Center's Ramskeller Pub

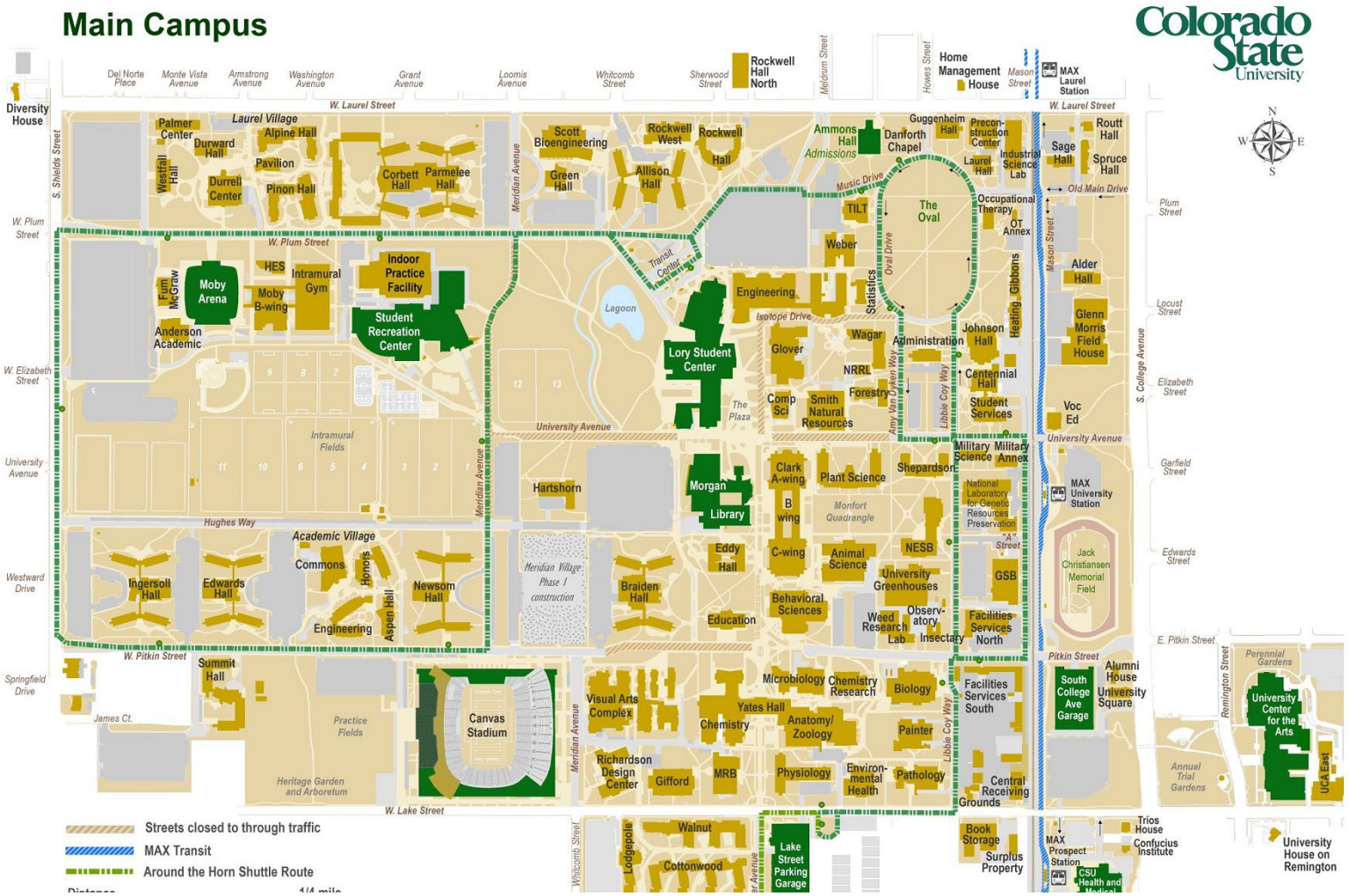


Figure 2, Colorado State University Main Campus map

Department of Computer Science

My internship took part in the department of Computer Science. It is a very active department of CSU engaged in transformative innovation and interdisciplinary research. Those research areas cover computer networks, data security, software engineering, bioinformatics, big data, artificial intelligence (AI)…

The department laboratories thus regularly publish valuable research publications (with the NUI Lab about user interaction for instance), which are for some awarded or founded by external organisms.

Figure 4, photo of the Computer Science building, source: compsci.colostate.edu

This is where I spent most of my time at CSU. This building counts many well equipped computers and meeting rooms, as well as desks on which I loved to work in the first part of my internship with an amazing view in front of the mountains behind the university.

Figure 5, photo of mountains view from the second floor of the Computer Science building

NUI Lab

Originally founded by Francisco R. Ortega in January 1015 at Florida International University, the Natural User Interaction Lab is now a laboratory part of the Department of Computer Science of the CSU.

It focuses on fields of research about 3D user interfaces. Some examples are gesture interfaces with gesture recognition and elicitation, multi-modal interaction, or virtual & augmented reality interfaces. I could for instance test impressive experiences of another NUI Lab member about virtual avatar assistant with gesture recognition.

Figure 6, NUI Lab banner, source : nuilab.org

Here are some examples of the NUI Lab projects:

- **Multi-Modal Gesture Recognition**

  This project focuses on the research about gesture recognition with speech and how those data can be combined to improve recognition accuracy[2]

- **Gesture User Preference and Elicitation**

  The user interactions are not always obvious when it comes to gesture commands. Multiple gestures or speech commands can be natural for different subjects for instance to flip a cube. Having a better understanding of average user preferences and elicitation for gesture interaction is the goal of this project.

[2] A predictive model accuracy is the measurement of the difference between its actual output and the expected results.

AI training

Hands-On Machine Learning

When Auguste and I firstly talked with Francisco about our internship project and the skills that it would imply, it appeared clearly that we would in a first time need just to train ourselves on machine learning. I did already look into artificial intelligence and machine learning but only by curiosity and personal interest, never in a practical way or in school at ENIB, I thus didn't have the level to begin a project with neural networks goals.

Francisco gave us a book to get start with machine learning in a practical way, the excellent *Hands-On Machine Learning with Scikit-Learn and TensorFlow* written by the French Aurélien Géron (which I still read in English for the vocabulary). It's a great book to discover artificial intelligence, machine and deep learning, as it covers the theoretical notions and concepts of the different types of machine learning systems as well as the technical aspects through an end-to-end project with Python[3].

Machine Learning Fundamentals

To explain my internship work and the use of machine learning, it is necessary to define it. A good general definition has been given in the early ages of computer science by Arthur Samuel in a paper published by the IBM journal[4]:

> *"field of study that gives computers the ability to learn without being explicitly programmed."* – Arthur Samuel*, 1959*

In our time where data are becoming more and more important, bulky and tough to manually analyze and treat, we need algorithms to do the work more than ever. Machine Learning is a field of study which responds to those needs by introducing the concept of computer algorithms which could learn by themselves on given data.

[3] Python is a programming language widely used by AI researchers.
[4] Source : https://ieeexplore.ieee.org/abstract/document/5389202.

Supervised, unsupervised and semi-supervised Learning

Machine learning (ML) works by setting up a *training set* of data that you feed into an algorithm to train it, and get output predictions. In function of the type of supervision given during training, AI models can be classified into different major categories. Here are the most used ones.

- *Supervised Learning* – most common

When the ML program is fed with *labeled* data – data with the wanted solution included – the algorithm will try to find a correlation between the inputted data and the given label, in order to be able to generalize it and predict as well fresh unlabeled data. We call that kind of training *Supervised Learning*, and it is the most common as it applies lots of problems.

For instance, if you want to make a program which identifies on a bird picture whether it is seagull or not, you will feed the program with hundreds of bird pictures labelled as "Seagull" or "Not a seagull" for each picture. During his training, the machine learning algorithm will then try to find a correlation between the inputted pixels and the being-a-seagull fact to predict it.

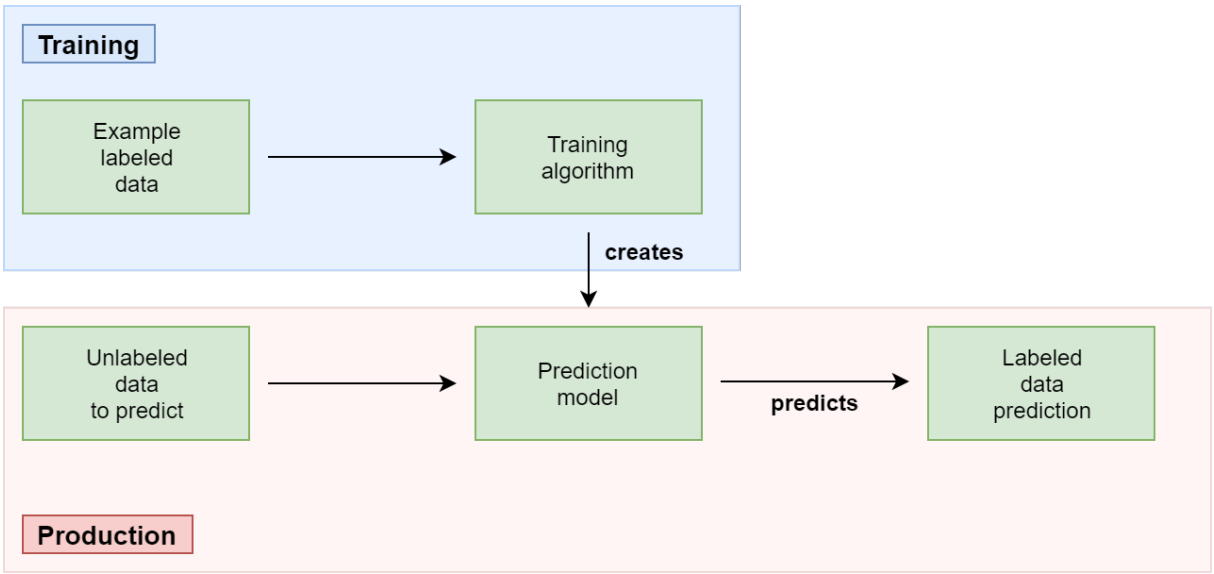


Figure 7, diagram about Supervised Machine Learning

- *Unsupervised Learning*

This type of training is unsupervised in the fact that you do not provide labels with the given input. There are multiple types of unsupervised learning algorithm, used for instance when you want to *group* (or *cluster*) your data by finding connections between them.

For instance, if you are a cinema owner, you have a bunch of data about your customers and you want to know more about similarities between them, you could feed them into that kind of algorithm. You could then notice for example that 30% of customers are men coming for romance movie on week days, while 20% are women coming on week-ends for action movies.

- *Semisupervised Learning*

Semisupervised Learning is a compromise between the 2 previous types of learning, when your ML system tries to group data based on their features, and then needs a user to label each group.

Good examples of semi-supervised learning algorithms are social network facial recognition ones, as they try to detect similar faces in your photos grouping them by person, and asks you the name if unknown.

- *Reinforcement Learning*

This type of training is also often used as it responds to specific needs. We talk about Reinforcement Learning when we define different possibilities of action for an algorithm, and *rewards* based on rules which will tell whenever the algorithm actions impact positively or not on a situation. The algorithm will then try during the training to find the best winning strategy, called *policy*, based on the rewards it gets while performing possible actions.

Those types of algorithm are for example widely used in video games artificial intelligences. The training of a chess AI would be for instance to define whenever the game is won or lost, define what pieces the algorithm can move and in which way, and then let it play thousands of games against humans/other AIs/itself.

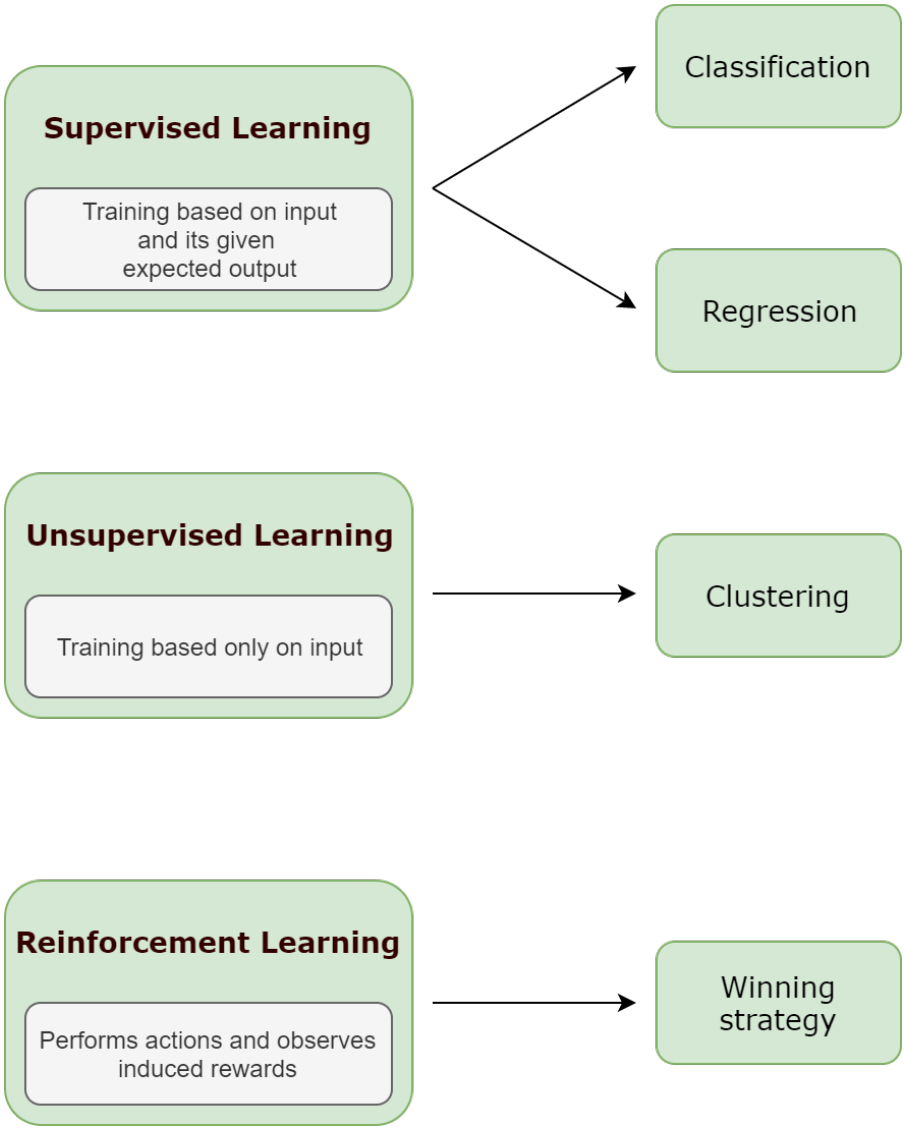


Figure 8, diagram about different types of Machine Learning

Classification & Regression

As you can see on figure 8, the given example with bird classification is not the only type of Supervised Learning task. When it comes to output a prediction of a label for an inputted data, there are two main types of output we could want.

- When we want to classify input between preset categories like for the example of "Seagull" and "Not a seagull" category; we call it a *classification* task
- When the seek output is rather a continuous value like if we wanted to predict the weight of a person based on his height, his age, etc.; we call it a *regression* task

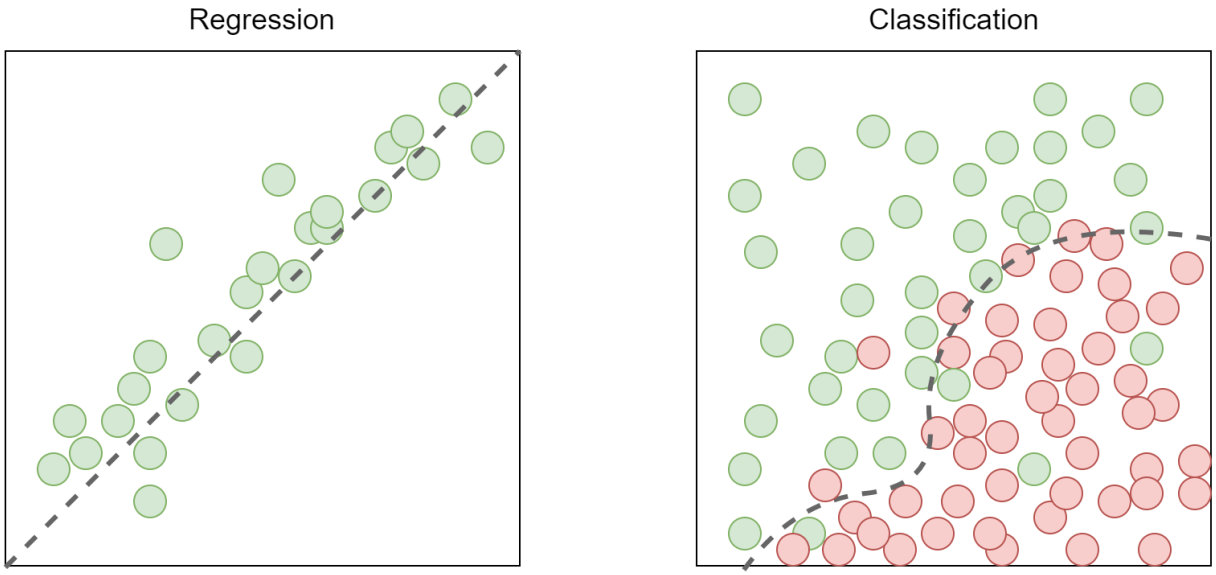


Figure 9, diagram illustration of differences between regression and classification tasks

A classification task will often be simpler (easier to reach a good accuracy) as the output possibilities are in a limited amount while with a regression task you have an infinite amount of possible outputs with sometimes very wide value range and diversity.

Some classification algorithms are however based on regression one, *Logistic Regression* is for example commonly used to determine the probability of a sample to be from one category or another.

While learning about Machine Learning and the practical aspects in Python, I for instance developed a digit classification algorithm based on Logistic Regression model, using the well-known Mixed National Institute of Standards and Technology (MNIST) dataset[5] made by Yann Le Cun for AI learners.

[5] Dataset of size-normalized handwritten digits pictures – source: http://yann.lecun.com/exdb/mnist/

Once I knew how we could classify the machine learning algorithms, I exercised many little projects to discover what were their different types by implementing them. I will not go in details on how the different training and decision algorithms work, but for instance with the MNIST dataset classification it was a logistic regression used to compute each picture probability to be one digit or another.

As many of the typical "classic" machine learning algorithms, *logistic regression* is a binomial regression model which estimates as best as possible a given mathematical model by considering every data point.

Artificial Neural Networks

Neural networks on another hand, do not try model a mathematical model but is instead inspired of the concept of human biological neural system. The structure has much changed since the first neural networks models, but the idea is still based on neurons activating in function of their connections to other neurons, with a huge number of connected neurons to simulate a real decision-making model.

Each neuron has one or more input, usually numbers between 0 and 1, and a linked output by a specific relation (which can be for instance a linear equation), with associated tunable weight for each input. During training, the neuron will adjust its inputs weights (as the human brain does by strengthening synapses) according to the validity of its output versus the expected one for a considered input.

Each neural network can be composed of two or more *neural layers*, each neuron able to be connected to each of the previous layer nodes. The inputs are thus representing the first layer of the network, the last being the output of the model. The intermediate layers between those are called the *hidden layers*.

To classify birds pictures with such a system, we could for example set up a neural network with a first layer of 256 inputs representing the gray-levels of the pixels from a 16x16 picture ; have a second layer of 20 neurons (arbitrary choice) ; and one output neuron which would represent from 0 to 1 the probability of the picture to represent a seagull.

During the training, each time we would feed this algorithm with a labeled data, the neural network would try to predict the output and in case of good result, would strengthen the neural connections that permitted this result, and vice versa. See Figure 10.

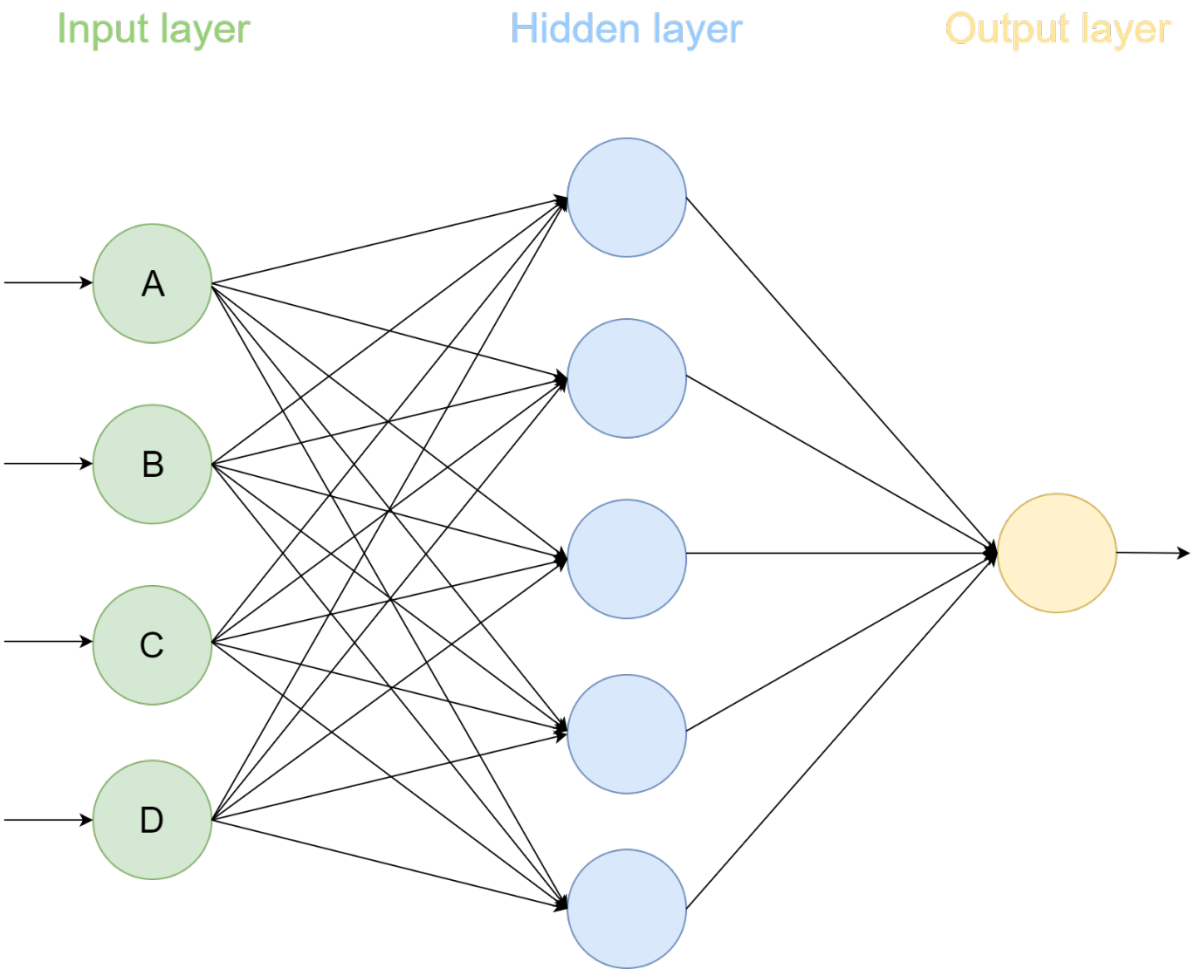


Figure 10. Representation of a simple artificial neural network with single output

We can evaluate the performance of a model while he is training by its *penalty loss*, which is a kind of score which can be computed by many different algorithms to estimate how bad is the model for a single prediction. A perfect model would thus have a loss of 0.

In practice, when it comes to the training of a ML model, a good practice is to split the dataset in two parts:

- The *training set* which is used to train the model and adjust the weights of each neuron connections according to the accuracy of the predictions.
- The *validation set* which is not fed into the algorithm but rather used to measure the model accuracy, showing thus the *validation loss* that is useful to determine if an algorithm is overfitting[6] the training set when it diverges from the *training loss*.

[6] A machine learning algorithm is *overfitting* when it learns to fit too precisely to the expected predictions on the training set, in a way that it is not able to generalize its "decision strategy" to new data outside of the training set. It often happens when the dataset is not big or diverse enough.

Neural networks with multiple dense layers can be used to model complex problems and predict labels with good accuracy, but the number of total connections growth exponentially with the number of layers, increasing thus drastically the needed computing power and training time.

The number of intermediate layers and their number of composing neurons are parameters that can be tweaked a lot when looking to improve the model accuracy. Usually those kinds of parameters that control and impact the learning process are called *hyperparameters*. Once a neural network is designed his hyperparameters are usually modified during the *hyperparameter tuning* (or *optimization*) to find the combination matching the best the expected results.

Gesture segmentation project

Introduction

While training myself on machine learning, the original project for my internship was about gesture recognition in a virtual reality environment. But before I finished my training part to start the project, the university buildings closed and the NUI Lab as well one week after, because of the coronavirus pandemic.

Working during COVID-19 was not an easy task, especially on an abroad internship. Despite the support from Francisco and the meetings we had every week, it was hard to keep a stable work schedule during a stay-at-home order in a little apartment room when you cannot really separate work and “free time” space.

Figure 11. Example picture of me working during the stay-at-home order

But I finally managed to find a rhythm in my everyday routine, and worked efficiently even if my work schedule was a bit shifted on night schedule as I prefer to work late and get up late than the opposite. While the “work” part of my internship was thus not so much affected as well as my training about ML, the rest of my abroad experience has been more limited by the pandemic outcomes in a kind of frustrating way.

The initial project about gesture recognition in virtual environments was impacted too. As research in those fields requires corresponding equipment such as virtual reality headset, the stay-at-home order and the inability to access the lab made my original project impossible to realize.

I still wanted to work about machine learning so Francisco then talked about the Easy Automatic Segmentation Event Labeler (EASEL) project[7], a software realized in order to help to the creation of gestures datasets.

EASEL

EASEL is a computer assisted dataset creation tool. The idea behind this software is to help users while the creation of a gesture dataset from raw videos, by auto segment and recognize human actions instead of manually editing the video, splitting the gestures at the right timestamps, and label them.

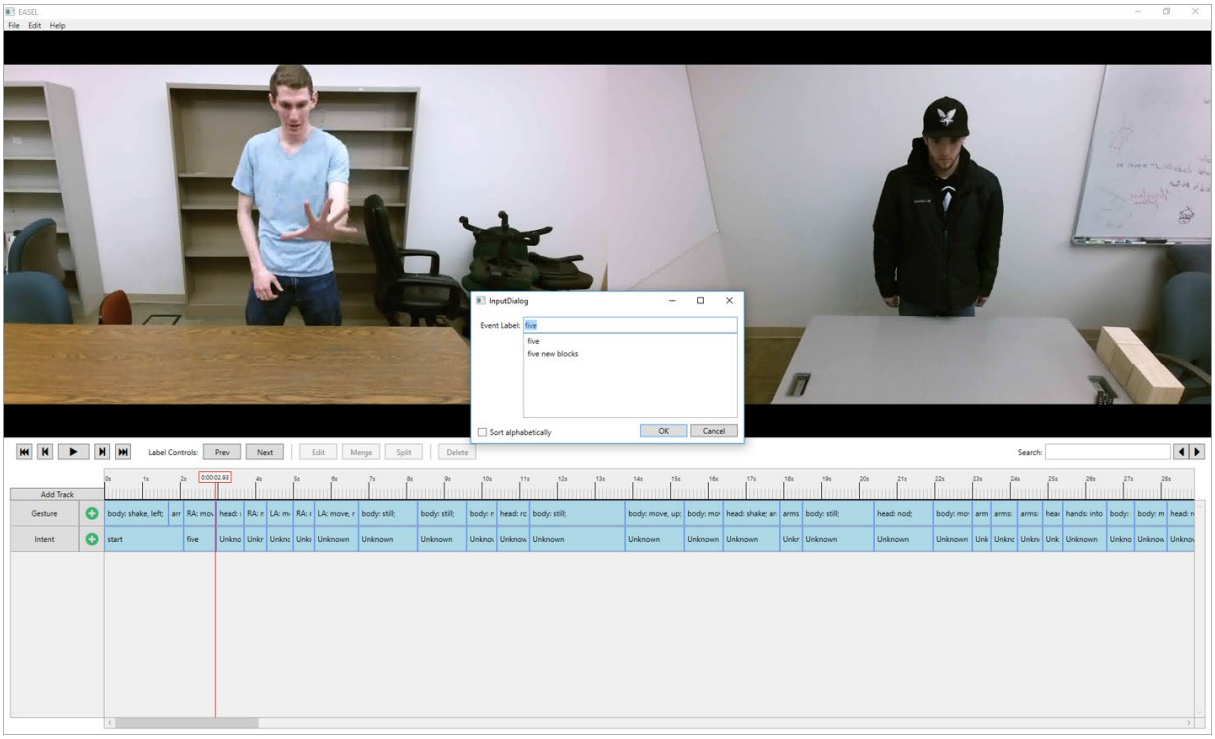


Figure 12, EASEL example screenshot

EASEL thus pre-segment the videos and tries to recognize each segmented gesture, and then allows the user to correct the wrong annotations. This tool showed pretty good results with the gesture recognition: only 32% of labels needed to be corrected manually; while 72% of the gestures start/end times required adjustments. A better gesture segmentation would then be a good improvement for the tool.

The auto-segmentation of the gestures is based on the "ACE-PC" technique by Arn et al.[8], which consists in estimating curvature in highs dimensional spaces. Basically, it is like

[7] Easy Automatic Segmentation Event Labeler - https://dl.acm.org/doi/10.1145/3172944.3173003

looking for acceleration in body joints coordinates to estimate movements start/end. This technique is convenient in this particular case as it requires no previous training and will only compute on inputted data.

The body joints coordinates along time are extracted from Kinect skeleton data, which is very convenient as it avoids computing those coordinates from the RGB video. Kinect data with depth sensors are also more reliable than a flat RGB video.

For the gesture recognition, EASEL team implemented a version of Dynamic Time Warping (DTW) which is an algorithm able to measure similarities between to temporal suites. It also requires no previous training and allows to add and classify new gestures on-the-fly.

Francisco wanted Auguste and me to continue EASEL work if possible, or start from scratch another solution with the same goals. I looked into the code, set up the environment, required libraries to make it compile and eventually ran it. Unfortunately, the software relied on an old remote database which cannot be accessed anymore.

As reverse-engineer the database structure would have been a time-consuming task as well as analyze the code to be able to continue the work; we agreed with Francisco that it would be more interesting and enriching to start from scratch with our own approach.

Own approach

While EASEL relies on non-AI algorithms which require no training, our approach is based on the idea that a ML algorithm would be able, with enough data diversity in its training, to generalize on every human subject (based on its skeleton data) and to detect movement start/end for segmentation, as well as recognizing same segmented gestures.

The EASEL most improvable aspect seemed to be the gesture segmentation so we decided to focus on that only point first. The final choice has been to set up a recurrent neural network with a logistic regression which would compute the probability of each frame – given the coordinates of each body joint – to be a movement start/end or not.

While we would still talk and think together as well as for making the technical decisions, we then split the work between Auguste and me to begin the same project but with each of us with our own experience. He would more focus on the recurrent neural network (RNN) model development while I would rather focus on the data preparation. That way each of us could study different topics and explain to the other what he learned.

[8] Robert Arn, Pradyumna Narayana, Teegan Emerson, Bruce Draper, Michael Kirby, and Chris Peterson. Motion Segmentation via Generalized Curvatures. Under review in *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

For the technical aspects, I chose to use Python for the data preparation as well as for the model development, because it is by far the most used language for this purpose and has already lots of documentation and libraries available with very useful tools for data science and machine learning. NumPy and Pandas are two popular libraries I used for the data transformation and preparation.

We would then have to do data processing and model training on huge datasets which represent lot of computing power and memory required. To make things work faster and be able to run our algorithms with no worries about our computer limitations, I installed Python and a Jupyter Notebook to use it on one of my private dedicated servers in a French datacenter.

Jupyter Notebook is an open-source software that allows users to edit code online through the web browser, run it and share its results easily.

We also decided to use TensorFlow for the predictive model development, the most suited and advanced library for our uses. It is also well designed for multi-processor computing and well optimized which is an important point when dealing with big datasets of hundreds of videos.

Figure 15. Python logo

Figure 14. NumPy logo

Figure 13. Pandas logo

Figure 16. Jupyter Notebook logo

Figure 17. TensorFlow logo

Choice of dataset

Once technical aspects defined and working environment set up, I just needed one more thing to start: a dataset. This choice represents an important decision for the rest of the project as it will be the raw matter which I will rely on to try to extract valuable data, and produce a meaningful result.

Given the allotted time and my level in machine learning, it also reasonable to target too complicated goals like extract body joints from raw RGB video[9] instead of Kinect skeletons, or segment gestures of multiple subjects at a time.

Thus, the videos included in the dataset need to respond to multiple criteria to be usable in our project:

- Each video must only contain human gestures performed by only one subject with the least parasitic gestures possible, we would else need to first detect if multiple gestures are occurring simultaneously, treat each independently…
- The videos have to contain various numbers of gestures of various length for the model to be able to generalize to any gesture.
- Corresponding body joints coordinates have to be already extracted and given with videos, saving us lot of times.
- Videos have to be labeled with timestamps of each gesture start/stop in order to feed them to our model by supervised learning.

[9] RGB videos are natural human videos composed by Red-Green-Blue images.

When I was looking for available gesture datasets, I came across a survey report about *RGB-D-based Action Recognition Datasets*[10] which includes a lot of pretty complete ones. A deep analysis and comparison of them is also made which allows to have fast and good insights on the datasets.

I was thus able to find some good candidates like the UTD-MHAD[11], which has a solid number of different gestures with good quality inertial sensor and depth data, a good diversity in its subjects… But the problem for this one as well as for many others is that it is a final reviewed dataset; which means that each gesture has already been segmented and isolated in independent files without previous and next frames.

Our model is based on RNNs, so it needs the previous frames joints coordinates in order to tell if a frame is a movement start/end. We thus require continuous unsegmented gestures for our system to be able to train to segment them.

I then looked to the dataset which had been used by the EASEL, as they had the same constraints as we had. The dataset they used to test their tool is the EGGNOG[12] dataset. This dataset made under the NUI Lab supervision seemed to be perfectly suited to those needs and moreover we had access to the original files which include long videos with lots of continuous gestures.

Figure 18. EGGNOG video sample screenshot

[10] Jing Zhanga, Wanqing Lia, Philip O. Ogunbonaa, Pichao Wanga, and Chang Tanga. *RGB-D-based Action Recognition Datasets: A Survey* – https://arxiv.org/abs/1601.05511

[11] Chen Chen, Roozbeh Jafari, and Nasser Kehtarnavaz. *UTD Multi Modal Action Dataset* – https://personal.utdallas.edu/~kehtar/UTD-MHAD.html

[12] Isaac Wang, Mohtadi Ben Fraj, Pradyumna Narayana, Dhruva Patil, Gururaj Mulay, Rahul Bangar, J. Ross Beveridge, Bruce A. Draper, and Jaime Ruiz. *EGGNOG: Elicited Giant Gallery of Naturally Occurring Gestures Dataset* – https://www.cs.colostate.edu/~vision/eggnog/

The EGGNOG dataset includes arounds 24,000 gestures labels across more than 350 videos. The dataset being 21 Gigabytes heavy, it was very convenient to have a dedicated server on which download and process it.

As you can see on Figure 18 subjects were standing behind a table so every gesture are upper body ones (mainly with head or arms), but it is a good base to start with. Videos have been captured with a Kinect which also includes each body joint coordinates through *skeleton data* files.

Skeleton data extraction

The first step was thus to extract the Kinect skeleton data for each video in an array, and removing the unneeded/unwanted data.

As the Kinect sensors do not have a view on the lower body joints, all corresponding coordinates had to be dropped as well as outliers like some joint orientations. Code available on Annex 1.

EventIndex	Time	SkeletonId	HandLeftConfidence	HandLeftState	HandRightConfidence	HandRightState	Joints					
0	60878	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05344743	-0.1622912	1.434072	-0.02257698
1	390144	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05411628	-0.1622954	1.433774	-0.01796596
2	1060801	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05393379	-0.1623337	1.433705	-0.01082574
3	1390146	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05389345	-0.162351	1.43371	-0.004227282
4	1720109	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05391142	-0.1623924	1.433695	0.003328574
5	2060914	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05397823	-0.1624121	1.433701	0.01312614
6	2719906	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05411291	-0.1624568	1.43367	0.02412818
7	3060726	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05454592	-0.162537	1.433553	0.03678077
8	3389968	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05533413	-0.1626774	1.433231	0.04956678
9	3719935	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.05675653	-0.162883	1.432816	0.06182534
10	4390067	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06019704	-0.1631215	1.432132	0.07915752
11	4719937	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06203945	-0.1634849	1.431693	0.08823433
12	5060756	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06310227	-0.1636941	1.431412	0.09531873
13	5391303	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06404428	-0.1639106	1.431126	0.1013687
14	5719889	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06527883	-0.1641235	1.43091	0.107498
15	6060711	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.0663107	-0.1642423	1.430423	0.112544
16	6390039	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.0670608	-0.164325	1.43019	0.1158322
17	6719932	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06761459	-0.1643433	1.430075	0.1181567
18	7060969	7,20575940379289E+016	INFERRED	OPEN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06815704	-0.1643798	1.430104	0.1201091
19	7390129	7,20575940379289E+016	INFERRED	UNKNOWN	NOT_TRACKED	UNKNOWN	SpineBase	TRACKED	0.06837951	-0.1643925	1.430084	0.1208795

Figure 19. Example of raw Kinect skeleton data

Index	Time	SpineBaseLocX	SpineBaseLocY	SpineBaseLocZ	SpineBaseOrW	SpineBaseOrX	SpineBaseOrY	SpineBaseOrZ	SpineMidLocX	...
0	60878	0.053447	-0.162291	1.434072	-0.022577	-0.023818	0.998039	-0.053311	0.040736	...
1	390144	0.054116	-0.162295	1.433774	-0.017966	-0.023970	0.998193	-0.052089	0.041452	...
2	1060801	0.053934	-0.162334	1.433705	-0.010826	-0.023720	0.998357	-0.051018	0.041578	...
3	1390146	0.053893	-0.162351	1.433710	-0.004227	-0.025078	0.998382	-0.050858	0.041006	...
4	1720109	0.053911	-0.162392	1.433695	0.003329	-0.026128	0.998366	-0.050714	0.040675	...
...	...	...	...	...	...	...	...	...	...	...
392	148060801	0.045580	-0.108119	1.455182	0.145699	-0.032601	0.985284	-0.083215	0.035697	...
393	148390054	0.047760	-0.108602	1.454057	0.146402	-0.034822	0.985216	-0.081875	0.036714	...
394	148720241	0.053461	-0.110056	1.451808	0.143861	-0.042106	0.985681	-0.077232	0.038356	...

Figure 20. Example of extracted useful skeleton data after unnecessary columns dropping

Once skeleton joints extracted and available for analysis, they could be plotted for visualization. Like every time I have to deal with temporal data, I thought it could only be a good idea to plot and visualize some example data to get first insights on its nature.

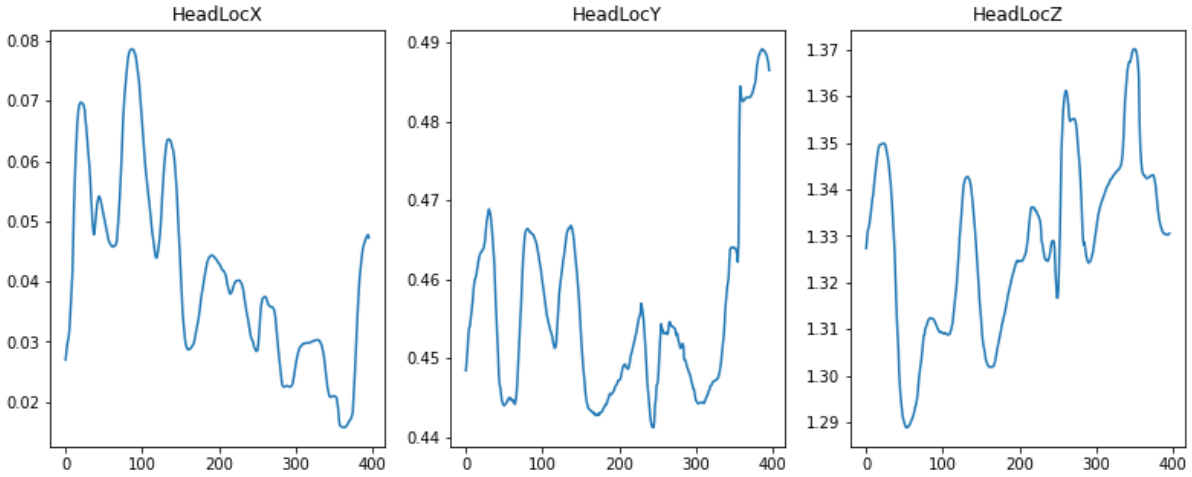


Figure 21. Head coordinates over time of a random video

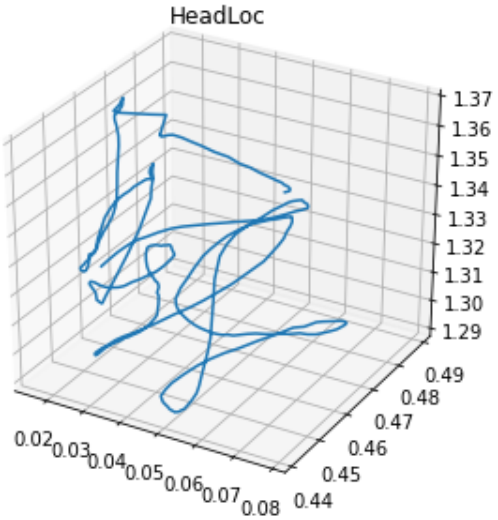


Figure 22. 3D representation of head coordinates over time of a random video

As videos are 30 FPS[13], each frame represents roughly a 1/30th of a second. So with the fact in mind that 90 frames would correspond to 3 seconds of video, we can notice on Figure 19 that coordinates data seem to be well continuous and not very noisy, at least for this joint.

[13] Frames Per Second

But when we take a closer look at each joint coordinate, we can see that some are much noisier than others, that's for instance the case of the hand joints (wrist, thumbs...). You can easily notice the difference in terms of noise for example on the left thumb coordinates of the same video, see Figure 23 and Figure 24.

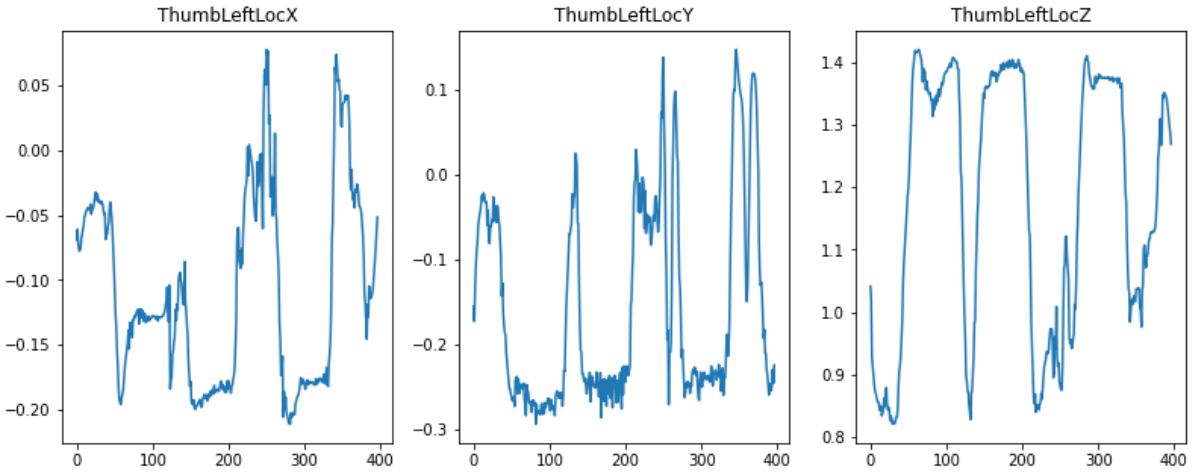


Figure 24. Left thumb coordinates over time of a random video

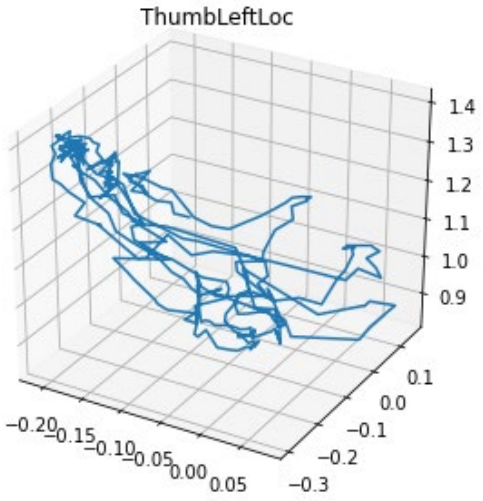


Figure 23. 3D representation of left thumb coordinates over time of a random video

I looked at several different random videos and it appeared that some joints were regularly noisy, which would amplify the difficulty of getting valuable results from those data. After a discussion with Dhruva who confirmed the potential improvements it could lead to, I decided to implement a filtering algorithm to remove the noise while keeping the whole data complexity and its curvature.

Data smoothing

Multiple reliable algorithms exist to filter a noisy signal. They all work by updating each point location with a new computed one based on the around points locations.

The *moving average algorithm* is one of the simplest techniques for smoothing signals in a software, it consists in converting each point value into a new value averaged on *n* points around the considered location. That number of points is usually named the *filter width* or the *window length*.

The greater the window length the more intense is the impact of the smoothing on the data. It has to be long enough to have a meaningful impact and flatten the noise while being short enough to not flatten the gesture peaks (limit the original information *loss*).

Another well-known and widely used filter technique is the *Savitzky-Golay algorithm*. It is considered as a much better approach with most of signals, including our continuous gestures ones. Instead of simply averaging points in a window, it performs an estimation of a polynomial fitting to a set of consecutive data points, and then computes the central point of the fitted polynomial curve as the new smoothed point. It thus also has a window length parameter, as well as the polynomial order to use.

Here are two illustrations of those two algorithms on a simple example set of random points, each picture shows one algorithm updating step that will be repeated for each point. The window length used for both filters is the 9 shown points, and the polynomial order of the Savitzky-Golay filter is set to 3.

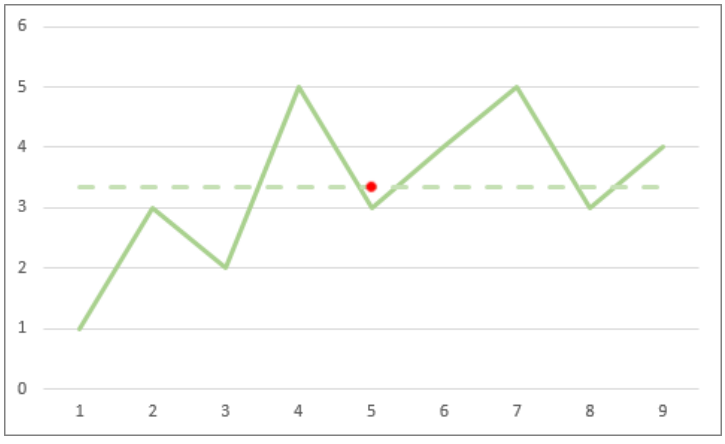


*Figure 25. Representation of a single point update with the **moving average** algorithm (red dot is the smoothed value)*

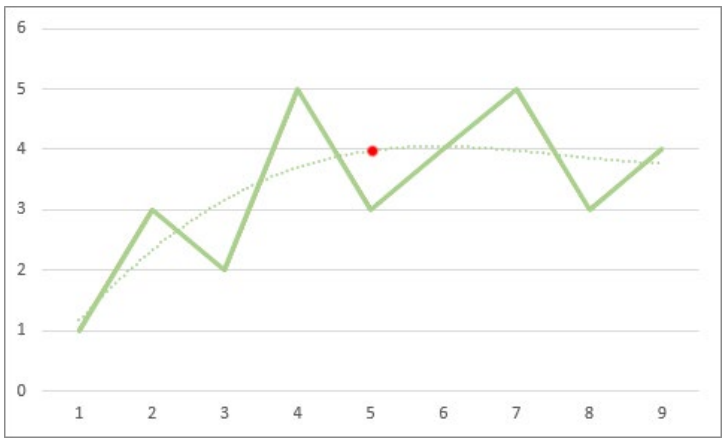


*Figure 26. Representation of a single point update with the **Savitzky-Golay** algorithm (red dot is the smoothed value)*

Depending on the kind of data we are dealing with, the available computing power and the expected algorithm speed, you might choose one filter type or another. In this context with the given "shape" of the continuous coordinate curves, it seems clearly that the Savitzky-Golay filter would be a better approach and would have much more chance to restore an unnoisy signal. That is therefore the filtering algorithm I used to smooth the data.

In order to choose the two filter parameters (window length and polynomial order), it is important to keep in mind that they also impact the performance of the algorithm. A good way to fix those values is to adjust the window length while keeping a low polynomial order, until the data is smoothed just as much as needed.

After a long process of implementation and parameter tuning to find good ones, here are the results of the smoothed data with a Savitzky-Golay filter of a 13-points window length and polynomial order to 2.

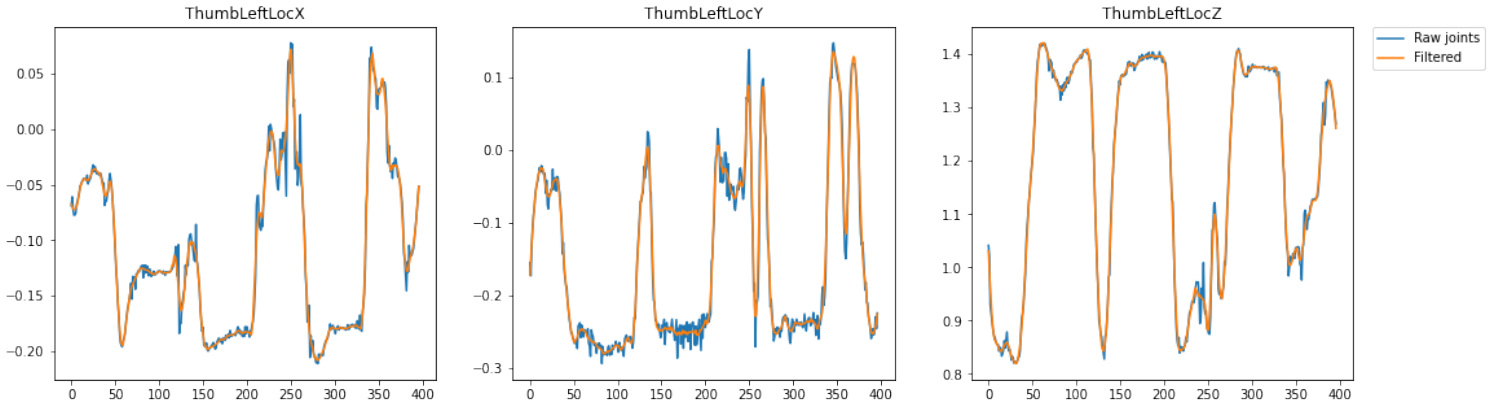


Figure 27. Filtered left thumb coordinates over time of a random video

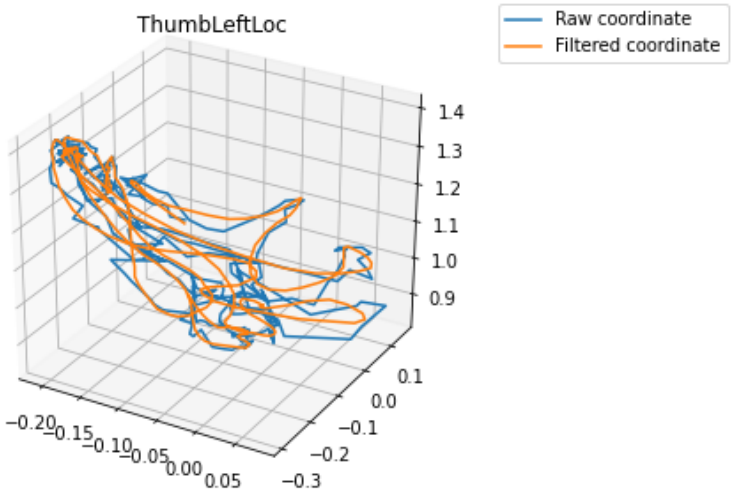


Figure 28. 3D representation of filtered left thumb coordinates over time of a random video

The smoothing filter seems to perform well by reducing in a strong way the useless complexity of some of the coordinates curvature induced by sensor noise. We can see that every meaningful peak is conserved as well as the global “path shape”, guarantying a reduced information loss.

Analyzing the unnoisy body joints is also another way to be sure that this filter keeps the integrity of data we don’t need to filter. We can thus see that the shown clean head coordinates in Figure 21 are not very affected by the Savitzky-Golay algorithm, as expected.

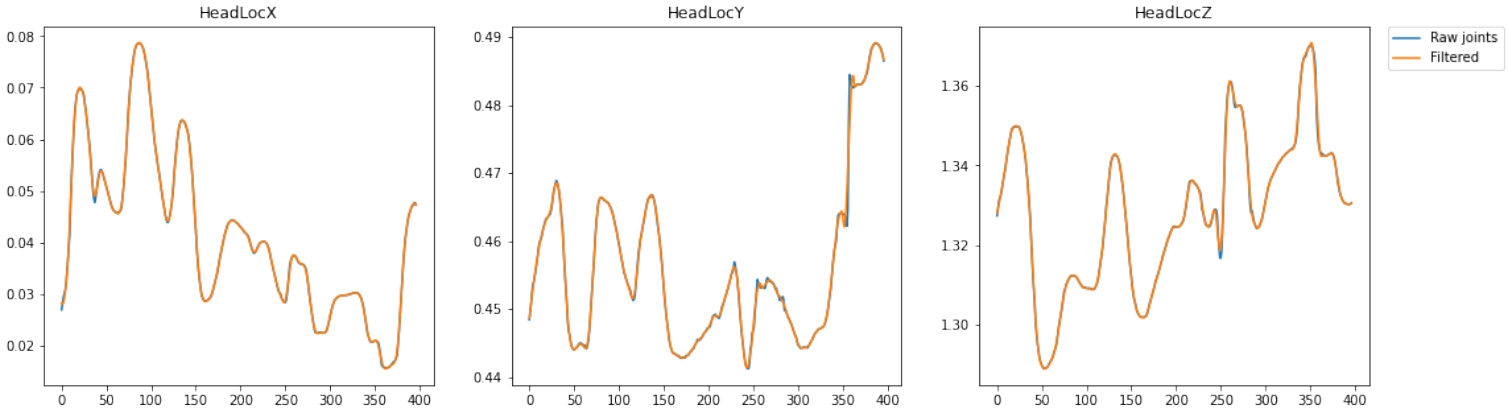


Figure 29. Filtered head coordinates over time of a random video

Labels extraction

While the body skeleton coordinates were at this point usable and smoothed, they did not include the labels needed for the supervised learning. The next thing I did was thus to iterate over the ~ 24,500 gesture labels from a "Labels.csv" file, and mark the temporal corresponding skeleton data as movement start/end.

File Name	Start Frame	End Frame	Label
s01\part1_layout_p02\20151105_191251_00	2	40	Unknown
s01\part1_layout_p02\20151105_191251_00	40	98	body: move, front;
s01\part1_layout_p02\20151105_191251_00	98	117	Unknown
s01\part1_layout_p02\20151105_191251_00	117	134	hands: into L;
s01\part1_layout_p02\20151105_191251_00	134	154	arms: move, front; hands: L;
s01\part1_layout_p02\20151105_191251_00	154	210	arms: rotate; hands: L;
s01\part1_layout_p02\20151105_191251_00	210	265	arms: rotate; hands: L;
s01\part1_layout_p02\20151105_191251_00	265	296	body: still;
…	…	…	…
s01\part1_layout_p02\20151105_191251_00	11760	11794	hands: into interleave;
s01\part1_layout_p02\20151105_192144_00	0	8	Unknown
s01\part1_layout_p02\20151105_192144_00	8	38	body: move, front;
s01\part1_layout_p02\20151105_192144_00	38	53	arms: move, left; hands: into claw, down;
s01\part1_layout_p02\20151105_192144_00	53	85	arms: together;

Figure 30. Sample of the labels file

It seems like an easy task at first look, as I had a list of skeleton frames and frames number corresponding to gestures start/end ; but, after few attempts and unsuccessful results, I realized that the frame numbers given by the labels file were sometimes out of the range of the skeleton data list, and they didn't seem to really match together.

The problem was that the frames numbering from the labels file does not correspond to the skeleton frames, for instance when the labels file states that there's a movement in video A from frame 30 to 54; it can actually correspond to the 34th and 52th frames of the skeleton data.

To match the labels with the skeleton data, I therefore needed to synchronize them by their timestamp. Skeleton raw data files include timestamp for each row as you can see on Figure 17, but the labels file does not.

My first solution was then to compute the labels timestamp based on the video framerate, as while looking EASEL code it appeared that labelling data were linked to raw video frames number. All videos being 30 FPS, a simple proportional calculation can give us in theory each frame corresponding timestamp.

But the results appeared inconsistent, and when I deeply analyzed the raw RGB video frames I realized that I couldn't rely on the framerate as it was not stable at all, some frames were skipped, some lasting longer… I talked with Dhruva about this problem as he had work on the EGGNOG dataset creation, and he talked me about ".frames" files mapping video frames number and their timestamp, exactly what I needed.

Frame	Timestamp
0	0
1	333333
2	666666
3	1333333
4	1666666
5	2000000
6	2333333
7	2666666
8	3333333
9	3666666
10	4000000
11	4333333
12	4666666
13	5000000
14	5666666

Figure 31. Sample of a random video ".frames" file

The public EGGNOG repository that I downloaded did not include those files that's why I was on the wrong way, so I took them from the original lab server.

At this point I thus just had to match each movement start/end timestamp with the skeleton frames ones; but as they were not simultaneous I needed to arbitrary define a time interval around each label timestamp, in which skeleton frames would be marked as movement start/end frame.

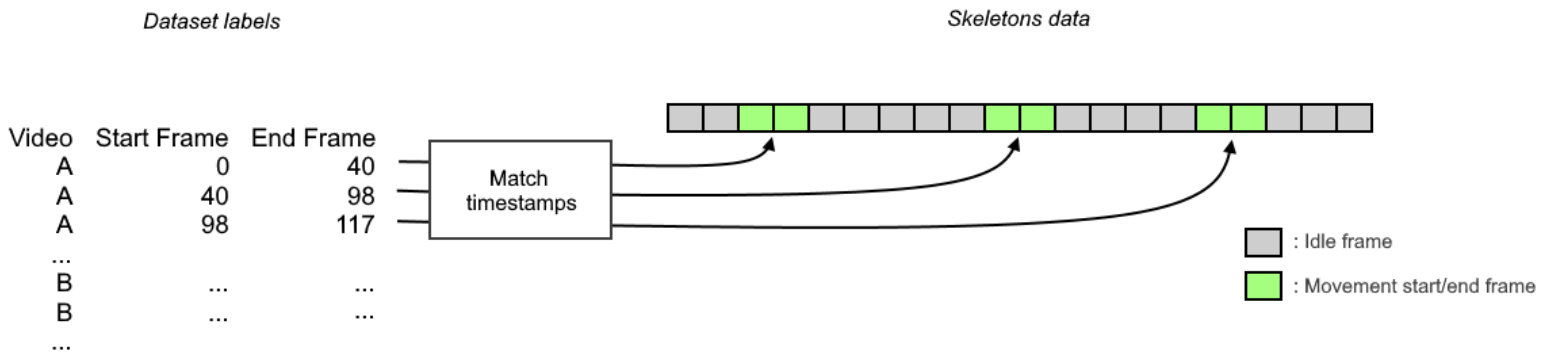


Figure 32. Labels and skeleton data matching illustration

I tweaked my interval duration until I ended up by just setting it to the length of a frame period, 1/30th of a second, which leads to between 1 and 2 frames marked for each label, which seemed satisfying.

The result is about 6% of the frames marked as gesture start/end; by iterating over ~150 files of the 450 total files, because 200 of them are missing the ".frames" file, which prevents me to map the labels.

Definitions of the functions used to do this operation are available on Annex 2 and Annex 3. As I also define an arbitrary time interval in which frames will be considered as movement end ones, it represents a tunable *hyperparameter* which impacts the output division of gesture and non-gesture frames.

The beginning of a movement represents something about few hundreds milliseconds, but it depends on the human subject speed and other factors, that is why I tweaked this interval to include more than only one frame for each label.

Thus, I do not think that this algorithm is the best way to measure if a frame is a start/end of a movement or not, but we didn't have any more information which could let me do a deeper analysis.

Samples equalization

During the first runs of our segmentation algorithm, we had an unexpected amazing accuracy of 94% good predictions of our software about each frame label. When looking to the training and validation loss during training, they seemed to converge which also is a good point.

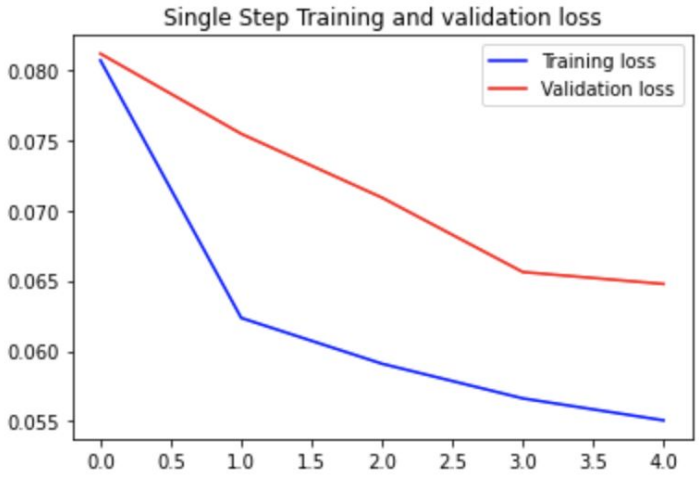


Figure 33. Training and validation loss of the model training with raw data

But such a success for a first try? I was very sceptic and my doubts confirmed when I took a look deeper into the results and the *confusion matrix*.

The confusion matrix is a 4-number matrix including true/false positives and true/false negatives. In this case I could see that every frame that was not a movement start/end was well labeled as “not a gesture frame” (true negatives), but every frame that actually was a movement start/end one (true positives) was mislabeled also as “not a gesture frame”.

In fact, the algorithm just learned to output “not a gesture frame” for 100% of the frames, and as we only had 6% gesture frames it could still reach a 94% accuracy.

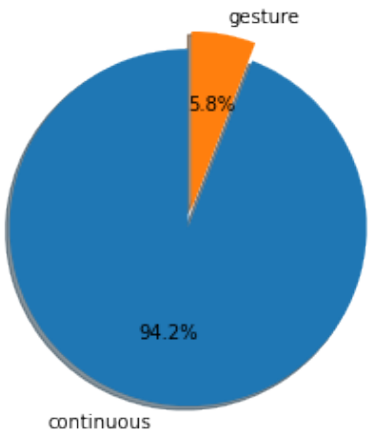


Figure 34. Frame labels repartition ("gesture" representing the movement start/end frames)

The first solution to this problem has been to randomly duplicate the gesture movement frames in order to equalize the division of the whole sample set, thus avoiding our model to look for a unique answer to output each time.

We’ve been able to get much more consistent data with this approach by equalizing to 50% of non-gesture data and 50% gesture, but we had to duplicate a lot the few amount of gestures we had compared to the number of “idle” frames.

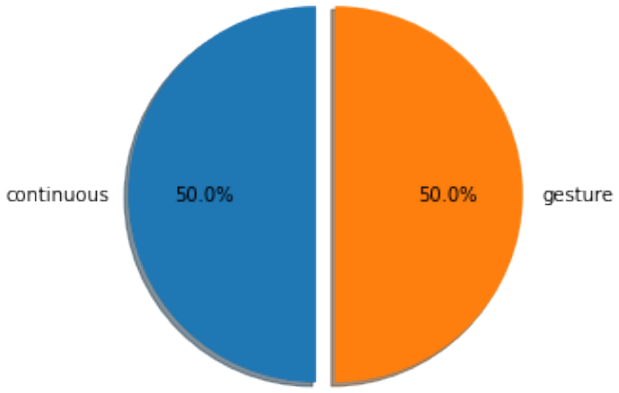


Figure 35. Frame labels repartition after the sample equalization

The fact that this duplicity gets the dataset further from a real-conditions data representation can be a problem which impacts the performance of the training and thus the whole system. Such a duplication could for instance leads the model to memorize the data and thus to *overfit* its predictions in a way.

Through searches and discussions with Dhruva and Francisco, the way to reduce this approach bad impact has been to oppositely remove some of the "idle" frames which would have the same effect on the equalization while having a lot less to duplicate data.

One could consider this technique as a loss of data, but as our model goal is to predict "gesture frames" more than the opposite, it should not affect the algorithm learning in a negative way.

The results then obtained on the training and validation loss with equalized samples fed into the algorithm reached a 39% movement detection accuracy[14] and an 84% overall accuracy[15] (accuracies computed by testing the trained model on a test set including around 30,000 unequalized frames).

When at this point I added my smoothing algorithm based on a Savitzky-Golay filter (see Data smoothing), the performance went down for unknown reasons. See Figure 37.

[14] The movement detection accuracy is the number of well detected movement frames divided by the total number of movement frames.

[15] The overall accuracy represents how many good predictions were made by the model in total on the tested set. In this particular case, this value can be misleading as it is computed on the validation set which contains lots more "idle frames" than movement ones.

Figure 37. Training and validation loss of the model training with equalized data

*Figure 36. Training and validation loss of the model training with **smoothed** equalized data*

While I was expecting the data smoothing to simplify the data without losing useful value, simplifying thus the task for the model to find good correlations and do correct predictions, it actually decreased in practice the performances of the model. When I asked Dhruva about this point, he was not able to give me an answer as he'd also expected the smoothing to increase the system accuracy.

He advised to continue the work on the data and the predictive model as it was an early stage of it, and to look deeper in the smoothing process later if I had time and this anomaly still appeared.

Final system

Towards the end of my internship, after Auguste made lots of improvements for the model based on Dhruva's advice, we ended up with a well-prepared dataset and a model which seemed to produce pretty good results.

The system structure (see Figure 38 on next page) consists in 6 consecutives neural layers:

- The input layer is made of the 99 body joints coordinates fed into the algorithm. In fact, as we are using LSTMs[16] the input layer is not just one skeleton frame but a batch of the actual tested frame and the 9 previous ones with their joints coordinates (not represented on Figure 37 for reasons of simplification).

- A first LSTM layer which takes the inputted batch of frames and outputs a same shape result to a second LSTM. The use of LSTMs instead of regular neural networks allows the system to benefit from their internal feedback with memory which seemed us a good choice because the sequential form our data, every joints coordinates being the continuation of previous ones. We used LSTMs with 160 hidden nodes (best hyperparameter found during the tuning) and batches of 10 frames.

- The second LSTM output is fed into a classic dense neuronal layer of also 160 neurons.

- A second dense layer takes the output of the previous layer, this one only contains 2 neurons acting as a *feature extractor*[17].

- The final output node is then passed through an *argmax* function which keeps the max value of the inputs, and in a softmax activation function, a common way to return a probability distribution in a classification problem.

The outputted result can then be used as the probability of each frame to be a movement start/end one, or classified using a probability threshold to consider the frame as a positive result.

[16] Long short-term memory is a type of recurrent neural networks which are neural networks with feedback connections, able to take sequences of sample instead of one sample at a time with classical neural networks, enabling memory mechanism very relevant in our case of continuous skeleton coordinates frames.
[17] Feature extraction is a way to reduce the dimension of data, it consists in building derived values (called "features") intended to reflect the inputted information in a smaller dimension, facilitating the learning and generalization process.

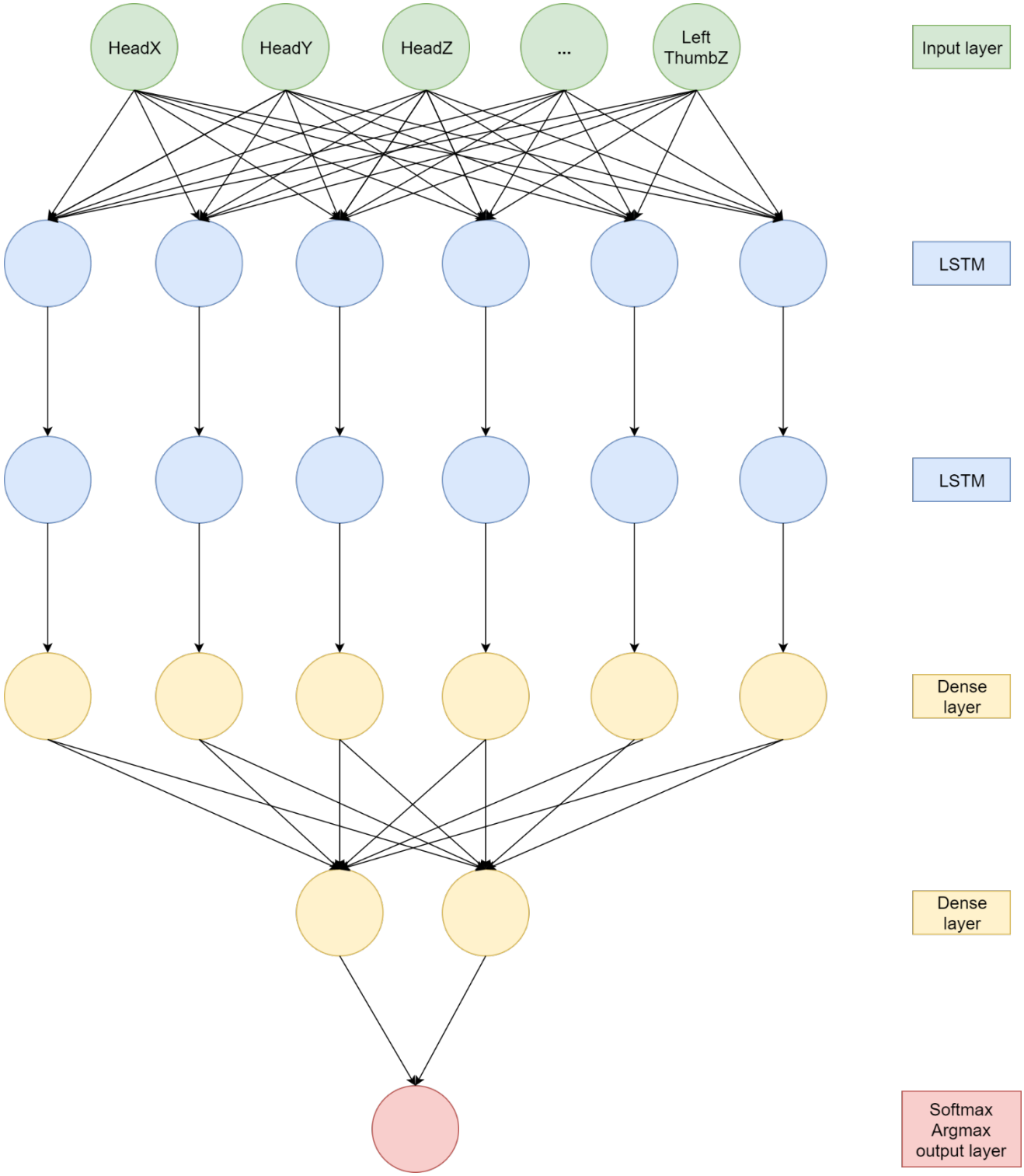


Figure 38. Basic diagram of the final gesture segmentation predictive system

Our model shown at first pretty good results, predicting most of the movement frames with at worst a bit of delay or marking excess (marking more frame as movement one around the label than expected) which can be a logic result : a movement beginning/ending does not last exactly one or two frames as labeled (see Movement detection interval).

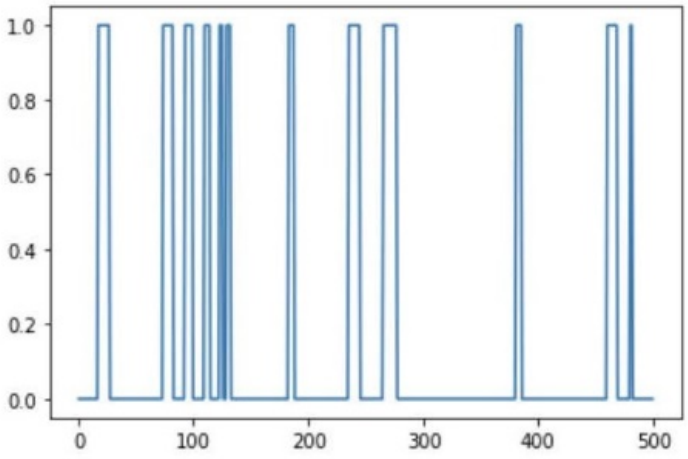


Figure 39. Example sample of gesture frames predictions for a random dataset video

This comparison between gesture predictions and labels is the one we first looked at, and it seems to confirm that our model is working as expected. When looking at the training and validation losses, they also seem to converge together which is usually a sign of a good generalization ability (see Figure 41).

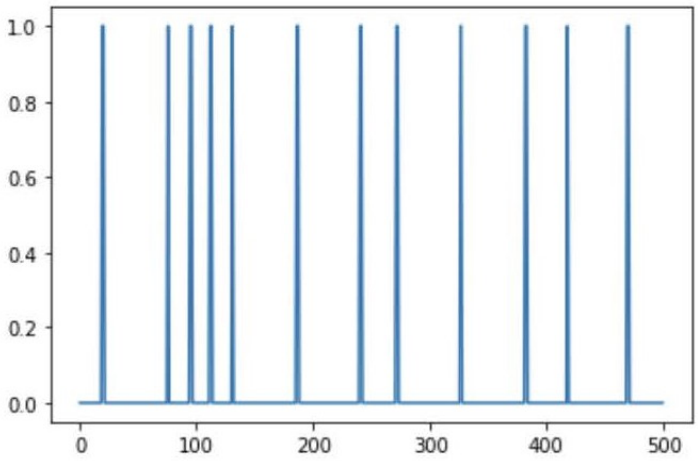


Figure 40. Corresponding sample of gesture frames labels

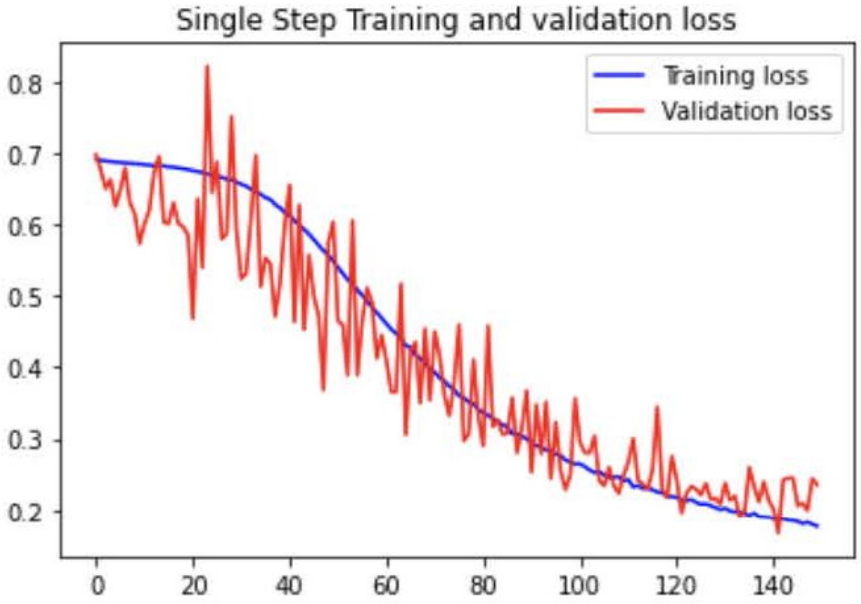


Figure 41. Training and validation loss of final algorithm training with smoothed data

Everything seemed to be alright but then I looked into the algorithm which splits the dataset into validation and training set, and something looked wrong. We suddenly saw that when Auguste did a modification to this algorithm to limit its computer memory impact (which was saturated crashing the process) early in the project, he introduced a coding mistake which led to merging some of the training data in the validation data.

The validation loss is an indicator for the model of being able to generalize and do good predictions on data out of the training set, the validation set intends to be isolated and unique in order to evaluate the model on data he could not train with, avoiding memorization of each training set sample label. This error thus compromised this principle.

When we realized this impactful error, I corrected it by making sure that the validation set would not contain any video or even any same human subject from the training set, which intends to avoid model overfitting.

When we then ran a training of the model with the corrected dataset, it appeared that our results were not as good as expected.

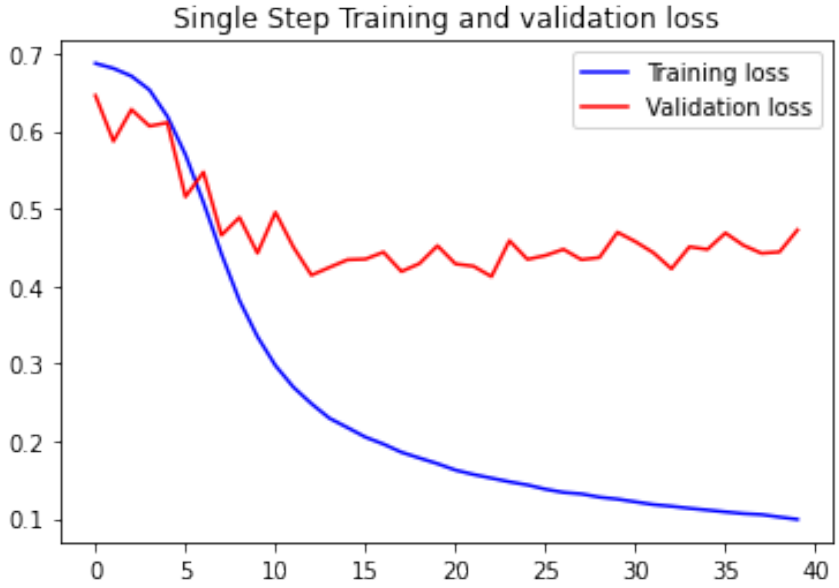


Figure 42. Training and validation loss of final algorithm after validation set split correction

As you can see on Figure 42, the training loss converges to a very low point which is supposed to be a sign that the algorithm manage to find a good correlation between given input and expected output. The problem is that the validation loss does not converge anymore when no sample of the training set is included in the validation one; which means that the model is overfitting our training data and does not manage very well to generalize its "decision-making strategy" to detect movement frames with a good accuracy.

On the internship end, time was missing to do so, but I believe that we could have made some improvements to avoid this problem and reach usable results.

Ways of improving

The final can be improved in many ways as we did not have enough time to explore every ideas and improvement corrections we thought about. Some of them would consist in:

- Increase the data diversity by generating noised duplicates for instance, increasing thus the dataset size.
- Expend the dataset by finding a way to generate the ".frames" files which would allow to add 200+ labeled videos to the dataset.
- Change the labels form with for instance a "movement score" instead of a simple "movement end" true/false value.
- Differentiate each joint movements by implementing a specific recognition model for each, and use the EGGNOG included labels descriptions to train each model with its corresponding movements.
- Do a more precise hyperparameters analysis and tuning, we couldn't try everything we had in mind as it demands a lot of computation power and training time when it comes to heavy layers.
- Preprocess some of the training work by adding new useful considered features, based on the raw coordinates. For instance, we could compute each body joint speed/acceleration and add it to the inputs.
- Convert the dataset coordinates and labels to an absolute form which could be summed with other converted datasets to increase the total size in a significant way, as increasing the dataset diversity should lead to a more generalizable system decision strategy.
- Check closer to the misrecognized frames looking for correlations between outputted errors and inputted data in order to focus on the model flaws and correct them.

Various improvements – and in particular the fact that there was a mistake in the training/validation split algorithm which made the model overfitting – were brought to light by looking at our results through the visualization software that I wrote.

Visualization software

Software conception

While working on the model there was several times where we struggled to analyze its results, it wasn't very convenient to compare the dataset given labels and our model predictions in a chart without the video of the corresponding movements.

I thus came up with the idea of a visualization software, to see simultaneously the videos and the data. I used C++ which is a fast and compiled language very powerful and friendly when powered by Qt. It lets you quickly implement working User Interfaces (UIs) and portable softwares, with full access to the Operating System (OS) native libraries.

Figure 43. C++ logo

Figure 44. Qt library logo

I realized the interfaces with QML Quick, a feature of Qt that I knew by name and that I've been wanting to try since a few years. It is very convenient way to implement quick and portable GUIs, and simple of use as it's more a markup language like HTML-CSS[18] than a classic programming style.

[18] Hyper-Text Markup Language (HTML) is a simple web markup language used today for every website structuring.
Cascade Style Sheets (CSS) is the language used to style (color, positioning…) HTML websites

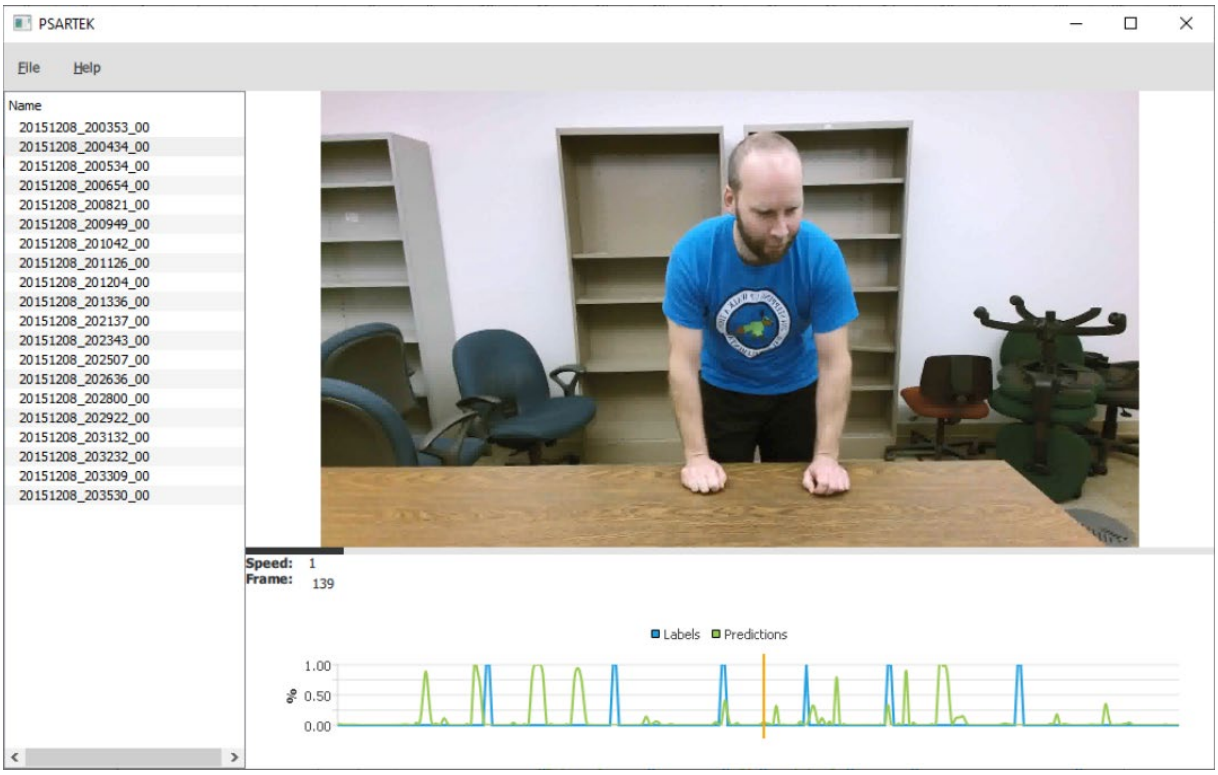


Figure 45. Example screenshot of the visualization software

I decided to keep the computation in Python scripts as I already did most of the work needed during the data preparation. The software would just use those Python scripts to generate 2 JSON[19] files (the EGGNOG labels, and our model predictions) and show them along the video.

I also added features like speed acceleration/deceleration or frame-by-frame to explore our results in a more precise way.

Results and interpretations

As we can see on Figure 45 (predictions made on a test video which hasn't been used during training, avoiding thus overfitted results), the results are not very accurate and we get a much more meaningful insight on our model predictions.

I could also spot certain videos with incredibly good results which proven then to be overfitted; and we've been able to significantly improve our predictive model thanks to that tool.

Another good improvement could be found by adding a visualization of the skeleton data, like a 3D representation of each joint coordinate according to the current video location; it could allow to see whenever there are outliers or errors in the data used as input to our system.

[19] JavaScript Object Notation (JSON) is a data format derived from JavaScript

Conclusion

Our whole predictive system needs thus further work to be really usable in a production environment, but it has good bases to do so. The objective of this internship was to get into machine learning and learn about artificial intelligence while trying to advance state-of-the-art in this field of study, and I am very happy with what I accomplished.

I learned a lot about Python especially with ML and data science libraries such as Pandas, NumPy and Tensorflow. I thus discovered those very useful features and will probably use them in most of my future Python projects, as well as Qt last versions and QML Quick for UI design which allows huge time savings.

The lab environment was really stimulating and challenging as I had to work with fields of study I wasn't used to but which were also exciting. I'm learned a lot about machine learning and artificial intelligence overall, and I'm very excited to use those new skills again on new projects.

It was nice to work in some kind of freedom while still having a trace and advice from the lab team. I had a great support from Dhruva when it came to Machine Learning and topics that I didn't mastered well, and Francisco was also really helpful and careful to everything that I needed. I thus want to thanks them again and every other member of the lab for their support and kindness towards me.

Appendices

1. Raw data kept columns header construction

```
# 1. Keeping only usable and useful features from the raw data

# Returns the list of used columns (dropping lower joints, useless features...)
def computeUsedColumns() :
    # Header computed on one frame row (skipping rows 2-6) with following replace regex:
    #
    # ([A-z]+)\s*\w+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+
    # $1\t$1Status\t$1LocX\t$1LocY\t$1LocZ\t$1OrW\t$1OrX\t$1OrY\t$1OrZ\t

    header = ["Index", "Time", "SkeletonId", "HandLeftConfidence", "HandLeftState", "HandRightConfidence", "HandRightState",
              "SpineBase", "SpineBaseStatus", "SpineBaseLocX", "SpineBaseLocY", "SpineBaseLocZ", "SpineBaseOrW",
              "SpineBaseOrX", "SpineBaseOrY", "SpineBaseOrZ", "SpineMid", "SpineMidStatus", "SpineMidLocX", "SpineMidLocY",
              "SpineMidLocZ", "SpineMidOrW", "SpineMidOrX", "SpineMidOrY", "SpineMidOrZ", "Neck", "NeckStatus", "NeckLocX",
              "NeckLocY", "NeckLocZ", "NeckOrW", "NeckOrX", "NeckOrY", "NeckOrZ", "Head", "HeadStatus", "HeadLocX", "HeadLocY",
              "HeadLocZ", "HeadOrW", "HeadOrX", "HeadOrY", "HeadOrZ", "ShoulderLeft", "ShoulderLeftStatus", "ShoulderLeftLocX",
              "ShoulderLeftLocY", "ShoulderLeftLocZ", "ShoulderLeftOrW", "ShoulderLeftOrX", "ShoulderLeftOrY",
              "ShoulderLeftOrZ", "ElbowLeft", "ElbowLeftStatus", "ElbowLeftLocX", "ElbowLeftLocY", "ElbowLeftLocZ",
              "ElbowLeftOrW", "ElbowLeftOrX", "ElbowLeftOrY", "ElbowLeftOrZ", "WristLeft", "WristLeftStatus", "WristLeftLocX",
              "WristLeftLocY", "WristLeftLocZ", "WristLeftOrW", "WristLeftOrX", "WristLeftOrY", "WristLeftOrZ", "HandLeft",
              "HandLeftStatus", "HandLeftLocX", "HandLeftLocY", "HandLeftLocZ", "HandLeftOrW", "HandLeftOrX", "HandLeftOrY",
              "HandLeftOrZ", "ShoulderRight", "ShoulderRightStatus", "ShoulderRightLocX", "ShoulderRightLocY",
              "ShoulderRightLocZ", "ShoulderRightOrW", "ShoulderRightOrX", "ShoulderRightOrY", "ShoulderRightOrZ",
              "ElbowRight", "ElbowRightStatus", "ElbowRightLocX", "ElbowRightLocY", "ElbowRightLocZ", "ElbowRightOrW",
              "ElbowRightOrX", "ElbowRightOrY", "ElbowRightOrZ", "WristRight", "WristRightStatus", "WristRightLocX",
              "WristRightLocY", "WristRightLocZ", "WristRightOrW", "WristRightOrX", "WristRightOrY", "WristRightOrZ",
              "HandRight", "HandRightStatus", "HandRightLocX", "HandRightLocY", "HandRightLocZ", "HandRightOrW",
              "HandRightOrX", "HandRightOrY", "HandRightOrZ", "HipLeft", "HipLeftStatus", "HipLeftLocX", "HipLeftLocY",
              "HipLeftLocZ", "HipLeftOrW", "HipLeftOrX", "HipLeftOrY", "HipLeftOrZ", "KneeLeft", "KneeLeftStatus",
              "KneeLeftLocX", "KneeLeftLocY", "KneeLeftLocZ", "KneeLeftOrW", "KneeLeftOrX", "KneeLeftOrY", "KneeLeftOrZ",
              "AnkleLeft", "AnkleLeftStatus", "AnkleLeftLocX", "AnkleLeftLocY", "AnkleLeftLocZ", "AnkleLeftOrW",
              "AnkleLeftOrX", "AnkleLeftOrY", "AnkleLeftOrZ", "FootLeft", "FootLeftStatus", "FootLeftLocX", "FootLeftLocY",
              "FootLeftLocZ", "FootLeftOrW", "FootLeftOrX", "FootLeftOrY", "FootLeftOrZ", "HipRight", "HipRightStatus",
              "HipRightLocX", "HipRightLocY", "HipRightLocZ", "HipRightOrW", "HipRightOrX", "HipRightOrY", "HipRightOrZ",
              "KneeRight", "KneeRightStatus", "KneeRightLocX", "KneeRightLocY", "KneeRightLocZ", "KneeRightOrW",
              "KneeRightOrX", "KneeRightOrY", "KneeRightOrZ", "AnkleRight", "AnkleRightStatus", "AnkleRightLocX",
              "AnkleRightLocY", "AnkleRightLocZ", "AnkleRightOrW", "AnkleRightOrX", "AnkleRightOrY", "AnkleRightOrZ",
              "FootRight", "FootRightStatus", "FootRightLocX", "FootRightLocY", "FootRightLocZ", "FootRightOrW",
              "FootRightOrX", "FootRightOrY", "FootRightOrZ", "SpineShoulder", "SpineShoulderStatus", "SpineShoulderLocX",
              "SpineShoulderLocY", "SpineShoulderLocZ", "SpineShoulderOrW", "SpineShoulderOrX", "SpineShoulderOrY",
              "SpineShoulderOrZ", "HandTipLeft", "HandTipLeftStatus", "HandTipLeftLocX", "HandTipLeftLocY", "HandTipLeftLocZ",
              "HandTipLeftOrW", "HandTipLeftOrX", "HandTipLeftOrY", "HandTipLeftOrZ", "ThumbLeft", "ThumbLeftStatus",
              "ThumbLeftLocX", "ThumbLeftLocY", "ThumbLeftLocZ", "ThumbLeftOrW", "ThumbLeftOrX", "ThumbLeftOrY",
              "ThumbLeftOrZ", "HandTipRight", "HandTipRightStatus", "HandTipRightLocX", "HandTipRightLocY", "HandTipRightLocZ",
              "HandTipRightOrW", "HandTipRightOrX", "HandTipRightOrY", "HandTipRightOrZ", "ThumbRight", "ThumbRightStatus",
              "ThumbRightLocX", "ThumbRightLocY", "ThumbRightLocZ", "ThumbRightOrW", "ThumbRightOrX", "ThumbRightOrY",
              "ThumbRightOrZ"]

    cols = dict()
    for i in range(len(header)) :
        cols[i] = header[i]

    # confidence
    for i in range(2,7) :
        del cols[i]

    # joint name
    for i in range(7,224,9) :
        del cols[i]

    # 8 lower joints
    for i in [116, 125, 134, 143, 152, 161, 170, 179] :
        for j in range(8) :
            del cols[i+j]

    # always null orientations
    for i in [39, 201, 210, 219, 228] :
        for j in range(4) :
            del cols[i+j]

    # remove tracking status features
    for i in [8, 17, 26, 35, 44, 53, 62, 71, 80, 89, 98, 107, 188, 197, 206, 215, 224] :
        del cols[i]

    return cols
```

Figure 46. Data header definition, unnecessary columns dropping

I firstly computed a header for the data, which includes a title for each column of the CSV[20]/TSV[21] file. This has been made by extracting the body joints names from a row of a random skeleton file and use them to title each file column, by executing the following replacement *regex*[22]:

```
Search:

([A-z]+)\s*\w+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+[\-0-9\.]+\s+

Replace:

$1\t$1Status\t$1LocX\t$1LocY\t$1LocZ\t$1OrW\t$1OrX\t$1OrY\t$1OrZ\t
```

SpineBase	TRACKED	0.05344743	-0.1622912	1.434072	-0.02257698	-0.02381788	0.9980385	-0.05331054	SpineMid	TRACKED	0.04073612
SpineBase	TRACKED	0.05411628	-0.1622954	1.433774	-0.01796596	-0.02396953	0.998193	-0.0520894	SpineMid	TRACKED	0.04145205
SpineBase	TRACKED	0.05393379	-0.1623337	1.433705	-0.01082574	-0.02371983	0.9983574	-0.05101767	SpineMid	TRACKED	0.04157773
SpineBase	TRACKED	0.05389345	-0.162351	1.43371	-0.004227282	-0.02507806	0.998382	-0.05085794	SpineMid	TRACKED	0.04100567
SpineBase	TRACKED	0.05391142	-0.1623924	1.433695	0.003328574	-0.02612835	0.9983658	-0.05071411	SpineMid	TRACKED	0.04067532
SpineBase	TRACKED	0.05397823	-0.1624121	1.433701	0.01312614	-0.02562695	0.9983707	-0.04926221	SpineMid	TRACKED	0.04122869
SpineBase	TRACKED	0.05411291	-0.1624568	1.43367	0.02412818	-0.02518147	0.9982469	-0.04782024	SpineMid	TRACKED	0.04183488
SpineBase	TRACKED	0.05454592	-0.162537	1.433553	0.03678077	-0.02508461	0.9979461	-0.04605954	SpineMid	TRACKED	0.042578
SpineBase	TRACKED	0.05533413	-0.1626774	1.433231	0.04956678	-0.02554756	0.9974548	-0.04443254	SpineMid	TRACKED	0.04338007
SpineBase	TRACKED	0.05675653	-0.162883	1.432816	0.06182534	-0.0268859	0.9968082	-0.04275667	SpineMid	TRACKED	0.04432496
SpineBase	TRACKED	0.06019704	-0.1631215	1.432132	0.07915752	-0.02908766	0.9956405	-0.03985262	SpineMid	TRACKED	0.04687395
SpineBase	TRACKED	0.06203945	-0.1634849	1.431693	0.08823433	-0.029258	0.9949452	-0.03798491	SpineMid	TRACKED	0.04871122
SpineBase	TRACKED	0.06310227	-0.1636941	1.431412	0.09531873	-0.02831137	0.9943694	-0.0366363	SpineMid	TRACKED	0.05032851

Figure 47. Sample of columns from 7 to 18 from a random skeleton file

This header allows me to then create a *Python dictionary* of those columns, keeping their indexes for further use and removing the columns I don't need/want. Those columns are then used by the skeleton-loading function (see Figure 34) to skip the unwanted ones.

```
# Loading of a random skeleton after columns computation
columns = computeUsedColumns()
skeleton = loadEggnogSkeleton("s09/part1_layout_p18/20151202_220007_00_Skeleton.txt", columns)
```

Figure 48. Example usage of the computed columns

[20] Comma Separated Values

[21] Tab Separated Values

[22] A *regular expression* or *regex* is a character sequence which defines a *search pattern*, usually used by string-search algorithms for operations such as "find" or "find and replace"

2. Data loading functions

```python
# 2. Data loading functions

# hyperparameters
# moveDetectionDuraction (s x10e-7) : window of time around which the frames are considered as
#      a start/end frame
#      Interval = [-moveDetectionDuration,moveDetectionDuration]

# Loads a skeleton dataframe from 'filepath' CSV, keeping only 'cols' columns
def loadEggnogSkeleton(filepath, cols):
    filepath = "datasets/eggnog.original/" + filepath
    dataframe = pd.read_csv(filepath, header=None, skiprows=[0], usecols=list(cols.keys()), names=list(cols.values()))
    return dataframe

# Loads frames corresponding timestamps from 'filepath' CSV
def loadTimestamps(filepath):
    filepath = "datasets/eggnog.original/" + filepath
    dataframe = pd.read_csv(filepath)
    return dataframe

# Return a function used to test if the frame "frame" matches one of the "eFrames" label frame within the
#      "moveDetectionDuration" time interval
def createFrameTest(eFrames, timestamps, moveDetectionDuration) :
    def getFrameLabels(frame) :
        isSF, isEF = False, False
        timestampsLen = len(timestamps)
        for f in eFrames :
            if f < timestampsLen and abs(frame[1] - timestamps[f][1]) < moveDetectionDuration :
                isSF = True
                break
        return isSF
    return getFrameLabels
```

Figure 49. Data extracting functions definition

I use the Python library *Pandas* to extract the data from the CSV files, and keep only the usable data columns by passing the previous computed columns header. The timestamp files (".frames" files) are clean and do not need such a filtering, so we can load and return them directly.

The third function is used during the dataset creation (see Figure 35) to make testing functions used to map each skeleton frame to its label "Start/end frame" or "Not a start/end frame".

3. Dataset array creation

```
# 3. Dataset creation & labels parsing

# Creates and returns the dataset as an array of [frames as numpy array, labels as list of booleans] loaded from the
#    "Labels.tsv" by matching each movement end label found with its corresponding frames (within a time interval)
#     marking them as "True"
def createDataset(columns, moveDetectionDuration = 333333) :
    labels = pd.read_csv("datasets/eggnog/Labels.tsv", sep='\t')
    eFrames = list()
    data = dict()
    lastFile = ""
    n=0
    ns, nf = 0, 0
    diag = True
    for i in range(len(labels)) :
        l = labels.loc[i]
        n+=1
        if lastFile != "" and lastFile != l["File Name"] :
            lastFile = lastFile.replace("\\", "/")
            eggSkel = loadEggnogSkeleton(lastFile+"_Skeleton.txt", columns)
            timestampsOk = False
            try:
                timestamps = loadTimestamps(lastFile+"_RGB.frames")
                timestampsOk = True
            except FileNotFoundError:
                try:
                    timestamps = loadTimestamps(lastFile+"_Video.frames")
                    timestampsOk = True
                except FileNotFoundError:
                    print(lastFile+ " ignored, no frame file found")
                    nf +=1
            if(timestampsOk) :
                frames = eggSkel.drop(["Index", "Time"], 1)
                labs = list(map(createFrameTest(eFrames, timestamps.values, moveDetectionDuration), eggSkel.values))
                data[lastFile] = [frames, labs]
                if diag :
                    a = np.array(labs)
                    movFramesCount = np.sum(a == True)
                    print(lastFile+" well processed: "+str(len(eFrames))+" labs => "+ str(movFramesCount)+"/"+str(len(labs))
                          +" frames marked.")
                else :
                    print(lastFile+" well processed !")
                ns += 1
            eFrames = list()
        lastFile = l["File Name"]
        eFrames.append(l["End Frame"])
    print(str(ns) +" files loaded for "+str(nf)+" fails (frames file not found), itered on "+str(n)+" total labels")
    return data
```

Figure 50. Dataset array creation function definition

This is the main function which uses the previous ones to return the dataset in a usable *Python dictionary* form. This function iterates over the labels file and:

- retains in an array “eFrames” each movement end frame number of a video (and not the movements start, as each movement end corresponds to the next movement start so I avoid redundancy)
- gets the corresponding timestamps (which can be in “_RGB.frames” or “_Video.frames” files), return an error if not found to continue to next file
- creates tests functions to map the frames to their corresponding label using the given time interval parameter used to control how many frames are marked by each label

- stores the results in an array of the frames and their corresponding labels, indexed in the *data dictionary* by current the file name
- empty its movements end list “eFrames” and continues to iterate on the next labels to get next video movements

While developing this algorithm, I discovered the *map* function of Python that lets you run a function on a whole list by replacing each value by the results of the passed function with the value as a parameter.

It is a very useful feature which can be used in many ways for data filtering or transformation, but I use it here (in a convoluted way maybe) to test each skeleton frame with a different function defined on-the-fly with the “eFrames” movement end frame list.